NASA Conference Publication 2491

First Annual Workshop on Space Operations Automation and Robotics (SOAR '87)

Proceedings of a workshop hosted by
Lyndon B. Johnson Space Center
Houston, Texas
August 5-7, 1987

NASA

NASA Conference Publication 2491

First Annual Workshop on Space Operations Automation and Robotics (SOAR '87)

Proceedings of a workshop sponsored by
National Aeronautics and Space Administration
and the U.S. Air Force
cosponsored by University of Houston-Clear Lake
and hosted by Lyndon B. Johnson Space Center
August 5-7, 1987

NASA
National Aeronautics and
Space Administration

Scientific and Technical
Information Branch

1987

PREFACE

This document represents the proceedings of the First Annual Workshop on Space Operations Automation and Robotics, otherwise known as SOAR '87, which was held at the NASA Johnson Space Center (JSC) on August 5-7, 1987.

This workshop was jointly sponsored by the National Aeronautics and Space Administration and the United States Air Force. It was cosponsored by the University of Houston Clear Lake. SOAR '87 helped to establish communications between individuals and organizations involved in similar research and technology. It brought together project/program managers in open exchange through presentation of technical papers and panel discussions. The objective of SOAR '87 was to provide a vehicle for engineers, scientists and managers of both the Air Force and NASA to come together in a workshop environment and exchange ideas, problems/problem solutions, and technical information on projects of mutual interest and, perhaps most importantly, to build a solid foundation for future interaction and cooperation. The workshop consisted of technical sessions emphasizing AI/Expert Systems, Human Factors, Environment, Robotics and Application Development and Transition. The workshop will rotate annually between NASA and an Air Force installation.

The papers included in these proceedings were published in general as received from the authors with minimum modification and editing. Information contained in the individual papers is not to be construed as being officially endorsed by NASA.

MESSAGE FROM THE GENERAL CHAIR AND ASSISTANT GENERAL CHAIR

The objective of SOAR '87 is to provide a vehicle for engineers, scientists, and managers of both the Air Force and NASA to come together in a workshop environment and exchange ideas, problems/problem solutions, and technical information on projects of mutual interest. Papers are invited based on the joint Air Force/NASA database of Space Operations Automation and Robotics projects. Attempts have been made to combine similar Air Force and NASA projects into joint sessions for maximum exchange of information. We believe SOAR '87 offers the best opportunity to ensure communications between the Air Force and NASA in an environment that meets the needs of all participants. I would like to express my appreciation to the various Air Force and NASA organizations that have worked so diligently with the belief that mutual benefit can be derived by coordination of mutual interest.

Robert H. Brown
NASA/Johnson Space Center

SOAR '87 is a unique opportunity for members of the Air Force and NASA technical communities to discuss problems of mutual interest and, perhaps most importantly, to build a solid foundation for future interaction and cooperation. SOAR '87 will present two distinct facets of Automation and Robotics. Sessions under each will focus on the fundamental issues that must be addressed to continue to advance the state-of-the-art. Additional sessions will emphasize the technical issues associated with the integration and transition of Automation and Robotics technology. Future applications that involve either or both of these technologies are dependent upon success in each facet. SOAR '87 will also feature sessions in Human Factors and Environment Technology. These sessions will contribute to the inter-disciplinary nature of the workshop. We look forward to your attendance and your participation in the technical program.

Capt. Gregory E. Swietek
HQ AFSC Andrews AFB

ACKNOWLEDGEMENT

Acknowledgements are due to all the personnel who provided the logistic support necessary to the success of this workshop. Thanks are also due to Computer Science Corporation, Lincom, Barrios Technology, McDonnell-Douglas Corporation, and Omniplan Corporation in extending the support of their personnel.

SOAR '87 ORGANIZING COMMITTEE

FIRST ANNUAL WORKSHOP ON SPACE OPERATIONS AUTOMATION AND ROBOTICS

General Chair	Robert H. Brown	NASA/JSC
Assistant General Chair	Capt. Gregory Swietek	HQ AFSC/DLAC
Executive Chair	Sandy Griffin	NASA/JSC
Technical Chair	Robert T. Savely	NASA/JSC
Technical Program Chair	Ann S. Baker	NASA/CSC
Assistant Program Chair	Dr. Timothy F. Cleghorn	NASA/JSC
Administrative Chair	Carol Kasworm	U of H/Clear Lake
Local Publicity Chair	Zafar Taqvi	NASA/LEMSCO
Exhibit Program Committee Chair	David W. Heath	NASA/JSC

TECHNICAL AREA DEVELOPERS

Application Development & Transition	Daniel C. Bochsler,	NASA/Lincom
	Maj. Steve Cross	USAF AFWAL
Robotics	Donna Pivirotto	NASA/JPL
	Maj. Steve LeClair	USAF AFWAL
AI Expert Systems	Dr. Jack P. Aldridge	NASA/MDAC
	Maj. Steve Cross	USAF AFWAL
Environment	Dr. Stuart Nachtwey	NASA/JSC
	Dr. Bruce Stuart	USAF
Human Factors	Dave Nagel	NASA/ARC
	Dr. Bob Bachert	USAF AAMRL

CONTENTS

APPLICATION DEVELOPMENT AND TRANSITION

SESSION 1: Automation of Checkout, Ground Support and Logistics
SESSION CHAIR: Mr. Robert Johnson,Wright-Patterson AFB

IMIS: Integrated Maintenance Information System 1

Space Shuttle Onboard Navigation Console Expert/Trainer System 11

MPAD MCC Workstation System: ADDAM and EEVE 15

Knowledge-Based Jet Engine Diagnostics 25

SESSION 2: Automated Software Development
SESSION CHAIR: Mr. Robert Hinson,NASA Johnson Space Center

Development of a Comprehensive Software Engineering Environment 31

Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project 39

The Knowledge Based Software Assistant 45

SESSION 3: Man-Machine Interfaces: Training and Tutoring
SESSION CHAIR: Lt. Col. Hugh Burns, Brooks AFB

An Intelligent Training System for Payload-Assist Module Deploys 53

Tutoring Electronic Troubleshooting in a Simulated Maintenance Work Environment 61

Intelligent Tutoring Systems as Tools for Investigating Individual Differences in Learning 71

An Intelligent Tutor for the Space Domain 77

SESSION 4: Neural Networks and Related Studies
SESSION CHAIR: Lt. Col. Daniel Biezad, Wright Patterson AFB,
Maj. Stephen E. Cross, Wright Patterson AFB

Neural Network Based Architectures for Aerospace Applications 85

Development and Experimentation of an Eye/Brain/Task Testbed 93

NASA JSC Neural Network Survey Results 97

Design of a Neural Network Simulator on a Transputer Array 111

SESSION 5: Functional Utilization
SESSION CHAIR: Lt. Roger Chilcott, NASA Johnson Space Center

Interchange of Electronic Design Through VHDL and EIS 119

Development of a Coupled Expert System for the Spacecraft Attitude Control Problem 125

User Interface Devices for Mission Control 133

The Desktop Interface in Intelligent Tutoring Systems 135

SESSION 6: Man-Machine Interfaces: Techniques and Methods
SESSION CHAIR: Mr. Robert F. Bachert, Wright Patterson AFB

Dynamic Human Dexterity and Control System (DEXDROID) (Paper not provided by publication date) 145

System Integrated Human Engineering (HE) on the Navy's RTIF/RAMP Project 147

The Ideal Modeling Methodology: Capabilities and Applications 153

The Rapid Intelligent Prototyping Laboratory: A Direct Manipulation Tool for Prototyping and Evaluating User-System Interfaces (Paper not provided by publication date) 159

SESSION 7: Systems Engineering and Distributed/Parallel Architectures
SESSION CHAIR: Capt. Richard Mraz, NASA Johnson Space Center

Cooperative Analysis Expert Situation Assessment Research 161

Performance Analysis of Parallel Branch and Bound Search with the Hypercube Architecture 165

Task Allocation in a Distributed Computing System 173

User Engineering: A New Look at System Engineering 183

SESSION 8: Validation and Verification
SESSION CHAIR: Glen Castore, Honeywell, Inc.

Approaches to the Verification of Rule-Based Expert Systems 191

A Formal Approach to Validation and Verification for Knowledge-Based Control Systems 197

Expert System Verification Concerns in an Operations Environment 203

Building Validation Tools for Knowledge-Based Systems 209

ARTIFICIAL INTELLIGENCE/EXPERT SYSTEMS

SESSION 1: Knowledge Acquisition I (Diagnosis/Knowledge Base Building)
SESSION CHAIR: Capt. James Ramsey, Kirtland AFB

Diagnosis: Reasoning from First Principles and Experiential Knowledge 217

Verifying Shuttle Onboard Software Using Expert Systems 223

Smart Bit (paper not written) 227

SESSION 2: Knowledge Acquisition II (Learning/Knowledge Base Maintenance)
SESSION CHAIR: Capt. James Ramsey, Kirtland AFB

A Multiexpert Knowledge System Architecture for Qualitative Process Theory (Paper not provided by publication date) 229

Teaching Artificial Neural Systems to Drive: Manual Training Techniques for Autonomous Systems 231

SESSION 3: Knowledge Acquisition III (World Modeling)
SESSION CHAIR: Capt. John V. Ferrante, Wright Patterson AFB

Design Knowledge Capture for the Space Station 239

Config: Qualitative Simulation Tool for Analyzing Behavior of Engineered Devices .. 247

A Situation-Response Model for Intelligent Pilot Aiding 253

A Human Performance Modeling Approach to Intelligent Decision Support Systems . 261

SESSION 4: Uncertainty
SESSION CHAIR: Mr. Robert N. Lea, NASA Johnson Space Center

The Empirical Accuracy of Uncertain Inference Models 269

Reasoning Under Uncertainty (Paper not written) 277

Decision-Theoretic Control of Planning Under Uncertainty (Paper not provided by publication date) 279

Autonomous Control Procedures for Shuttle Rendezvous Proximity Operations 281

SESSION 5: Reasoning I (Planning and Scheduling)
SESSION CHAIR: Nancy Orlando Sliwa, NASA Langley Research Center

Artificial Intelligence (AI), Operations Research (OR), and Decision Support Systems (DSS): A Conceptual Framework 287

Robotic Planner Expert System (RPLANES) 293

Expert Mission Planning and Replanning Scheduling System for NASA KSC Payload Operations 299

Trimodal Interpretation of Constraints for Planning 307

Range and Mission Scheduling Automation Using Combined AI and Operations Research Techniques 315

SESSION 6: Reasoning II (Explanation and Decision Making Applications)
SESSION CHAIR: Maj. Wesley M. Smith, Offutt AFB

Advanced Decision Aiding Techniques Applicable to Space 321

Automatic Routing Module 327

Robotic Air Vehicle Blending Artificial Intelligence with Conventional Software 335

SWAN: An Expert System With Natural Language Interface for Tactical Air Capability Assessment 341

SESSION 7: Monitor and Control
SESSION CHAIR: Dr. A. S. Dioxiadas, Ford Aerospace & Communication Corp.

Expert System Applications in Spacecraft Subsystem Controllers 349

The KATE Shell: An Implementation of Model-Based Control, Monitor and Diagnosis 355

An Expert System for Design of Digital Autopilots (Paper not provided by publication date) 361

Beyond Rules: The Next Generation of Expert Systems 363

SESSION 8: Architectures for Real-Time AI
SESSION CHAIR: Lt. Mark Peot, Wright Patterson AFB

A Conceptual Framework for Intelligent Real Time Information Processing 371

TDAS: The Thermal Expert System (TEXSYS) Data Acquisition System 375

Implementing CLIPS on a Parallel Computer 383

Real-Time Artificial Intelligence Issues in the Development of the Adaptive Tactical Navigator 389

ROBOTICS

SESSION 1: Kinematic, Dynamics, Mobility and Control
SESSION CHAIR: Dr. Richard Volz, University of Michigan

Software for Integrated Manufacturing Systems, Part I 397

Software for Integrated Manufacturing Systems, Part II 399

Communication and Control in an Integrated Manufacturing System 405

SESSION 2: Sensing and Perception (A)
SESSION CHAIR: Dr. Ronald C. Benton, Honeywell, Inc.

Development of Moire Machine Vision 413

Implementation of a Robotic Flexible Assembly System 421

CAD Based 3-D Object Recognition
(Paper not provided by publication date) 431

System Integration of a Telerobotic Demonstration S ystem (TDS) Testbed 433

SESSION 3: Sensing and Perception (B)
SESSION CHAIR: Dr. Rui J. P. deFigueiredo, Rice University

Vision Technology/Algorithms for Space Robotics Applications 441

Machine Vision by Optical Correlation, Programmable Spatial Light
Modulators, and Real-Time Image Warping
(Paper not provided by publication date) 455

SESSION 4: Manipulation, End Effectors
SESSION CHAIR: Dr. Robert H. Cannon, Stanford University

An Optimal Resolved Rate Law for Kinematically Redundant Manipulators 457

Telerobotic Research at NASA Langley Research Center 465

Manipulator Arm Design for the Extravehicular Teleoperator Assist Robot (ETAR):
Applications on the Space Station 471

Development of a Facility Using Robotics for Testing Automation of
Inertial Instruments 477

SESSION 5: Telerobotics
SESSION CHAIR: Mr. William Schneider, NASA Johnson Space Center

Space Station Assembly: Working with Robotics
(Paper not written) 485

Telerobotic Truss Assembly 487

Crew Interface with a Telerobotic Control Station 493

Telerobot for Space Station 497

The Use of Computer Graphic Simulation in the Development of Robotic Systems 501

SESSION 6: Task Planning and Reasoning
SESSION CHAIR: Ms. Edith Taylor, NASA Johnson Space Center

Intelligent Robotic Tracker 513

Calibration of Neural Networks Using Genetic Algorithms, with Application to Optimal Path Planning 519

An Efficient Representation of Spatial Information for Expert Reasoning in Robotic Vehicles 527

Task-level Robot Programming: Integral Part of Evolution from Teleoperation to Autonomy 533

HUMAN FACTORS

SESSION 1: Crew Integration and Protection
SESSION CHAIR: Robert F. Bachert, Wright Patterson AFB

Hyper Velocity Technology (HVT) Crew Escape 541

Aircraft Transparency Analysis Methods
(Paper not provided by publication date) 549

Surrogate Measures: A proposed alternative in Human Factors Assessment of Operational Measures of Performance 551

AUTHOR INDEX 559

IMIS
Integrated Maintenance Information System
A Maintenance Information Delivery Concept

Capt. Joseph C. Von Holle, Wright-Patterson AFB

Introduction

The Air Force Human Resources Laboratory (AFHRL), Logistics and Human Factors Division, is dedicated to improving the supportability of Air Force systems and the productivity of maintenance personnel. The Combat Logistics Branch of AFHRL is developing the Integrated Maintenance Information System (IMIS). The objective of IMIS is to improve the capabilities of aircraft maintenance organizations by providing technicians with a single information system for intermediate and organizational maintenance.

The modern maintenance environment is being increasingly inundated with additional information systems. Examples include the Comprehensive Engine Management System (CEMS), the Core Automated Maintenance System (CAMS), and the Automated Technical Order System (ATOS). Each new "maintenance aid" is a maintenance hindrance because it forces technicians to learn yet another system. To utilize the valuable information that these new systems offer, while eliminating the specialization required for each, AFHRL is developing IMIS. IMIS will utilize a very small portable computer/display to interface with on-aircraft systems and ground computer systems to provide a single, integrated source of the information needed to perform maintenance on the line and in the shop. IMIS will consist of a workstation for use in the shop, a portable computer for flightline use, and an aircraft interface panel for interacting with aircraft systems (Figure 1). The system will provide the technician with direct access to sev-

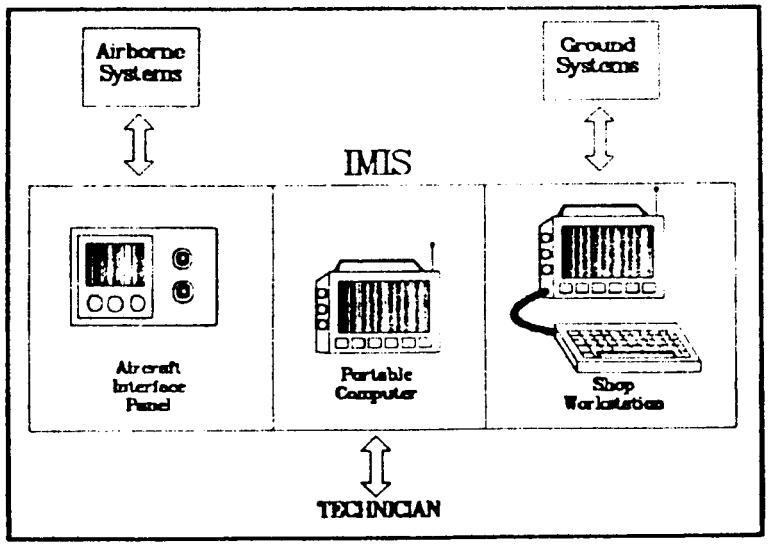


Figure 1: Integrated Maintenance Information System

eral maintenance information systems and data bases including CAMS, the supply system, ATOS, and an automated training data base. IMIS will process, integrate, and display maintenance information to the technician. The system will display graphic technical instructions, provide intelligent diagnostic advice, provide aircraft battle damage assessment aids, analyze in-flight performance and failure data, analyze aircraft historical data, and access and interrogate on-board built-in-test capabilities. It will also provide the technician with easy, efficient methods to receive work orders, report maintenance actions, order parts from supply, and complete computer-aided training lessons and simulations. The portable computer will make it possible to present quality information by taking advantage of the computer's ability to interact with, and tailor information to, technicians with varying levels of expertise.

Development is proceeding in three stages (Figure 2). Stage I, the Computer-based Maintenance Aids System (CMAS) established basic requirements for automated Technical Order (TO) data content, presentation formats, and basic delivery system hardware/software. Stage II, the Portable Computer-based Maintenance Aids System (PCMAS), is designed to implement the TO presentation specified in Stage I on the flightline, demonstrate interactive diagnostics and aircraft battle damage repair assessment, and test the feasiblility of these concepts during a field test. Stage III, Full IMIS Demonstration, will extend the concepts specified in Stages I and II, with an emphasis on information system integration throughout the maintenance complex. It will also incorporate state-of-the-art technology to reduce size and weight, while increasing capabilities.

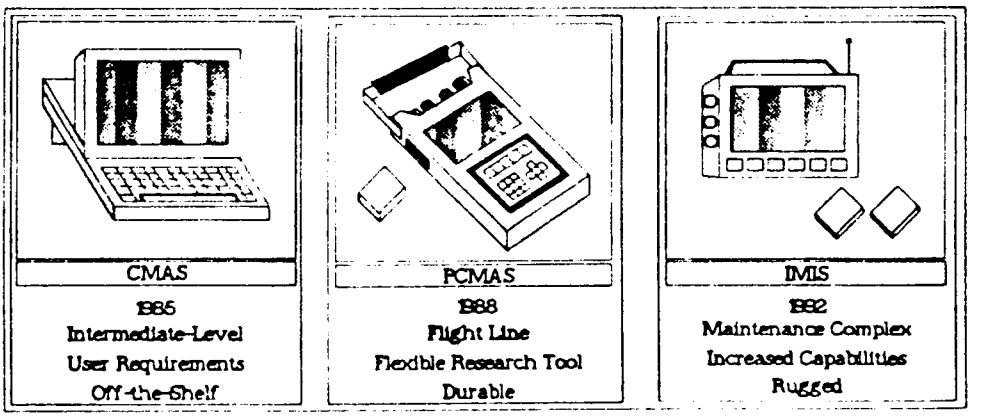


Figure 2: Three Stages of IMIS

STAGE I
Computer-based Maintenance Aids System (CMAS)

Due to the increasing complexity and number of modern weapon systems, the Air Force is faced with an ever growing number of paper-based Technical Orders (TOs). This has greatly increased costs and distribution problems. In addition, it has compounded problems with inaccurate data and lenghty correction times. To remedy the problem and provide improved techincal data, the Air Force is moving toward the digital storage and presentation of TOs. The Air Force Human Resources Laboratory believes the maintenance technician's needs should be considered first in the design of such a system. AFHRL has done extensive research to develop the technology required for an automated technical data system. The research has included a feasibility study, studies to develop the man/machine interface techniques required for an effective system, and studies to determine the information content requirements and presentation formats.

Two prototype systems were developed for intermediate level maintenance to test information presentation and man/machine interface techniques. Specific concepts tested were multiple levels of detail, random access to TO data, presentation of diagrams larger than the screen, function key utility, human interaction, and troubleshooting. The field evaluations established the feasibility and desirability of an automated maintenance system. The evaluations also demonstrated the importance of reformatting the data for automated presentation. Further analysis indicated that each paragraph should have associated tables and graphics and should not be dependent on the paragraphs before it. However, paragraphs should be linked in a hierarchical fashion so that the data can be reproduced as a paper TO, if desired. The data base must be in a neutral exchange format and should not contain code specific to screen presentation or other hardware limitations.

The first prototype system was tested at Offut AFB in December 1984. The development and evaluation of the system provided useful information with regards to computer size, response time, and color display. However, due to a number of problems, it did not gain user acceptance and was considered unsuitable for its proposed use. A second prototype was then developed based upon lessons learned from the first system.

The GRID Compass II computer was selected to host the second prototype. The GRID was chosen for its small size and its powerful capabilities which made it an ideal candidate for a CMAS prototype. The TO information used for the field test applied to the RT-728A/APX-64 radio receiver-transmitter. The checkout and analysis section of the data was analyzed to determine any additional sections needed to support the checkout. These additional sections included portions of Theory of Operation, Illustrated Parts Breakdown, and Troubleshooting. Additional troubleshooting routines were developed by an experienced technician. Figure 3 provides a sample screen presentation.

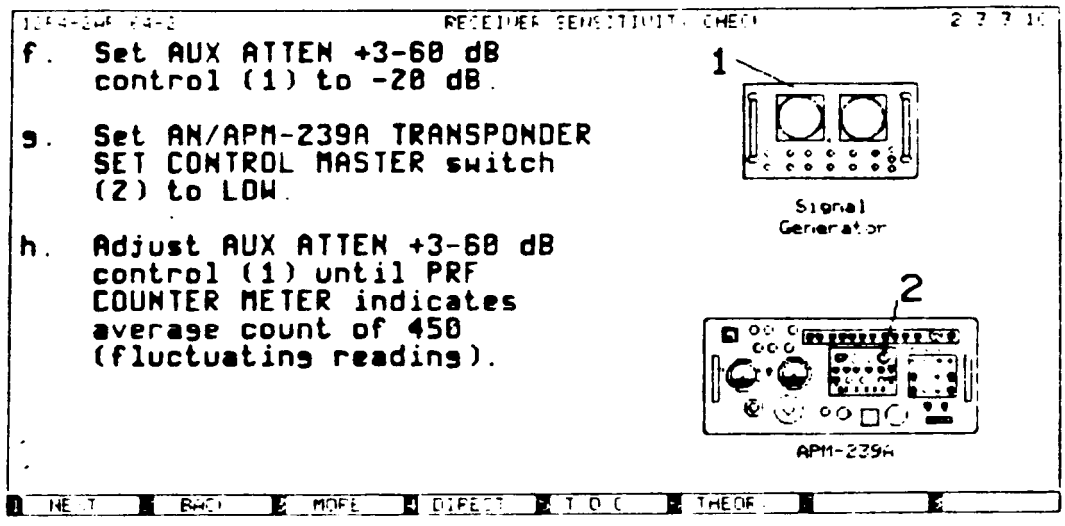


Figure 3: Sample Automated Technical Order Screen (reduced)

The results of the development and subsequent field test of the CMAS program were documented in two draft specifications. The Technical Data Content Specification established requirements for the content and formatting of data to be presented via electronic media. The Technical Data System Functional Specification established the system delivery functions, basic hardware/software capabilities, and the system performance requirements. A third specification, Technical Data Exchange Formats, currently being developed, will establish the coding techniques to establish a neutral format data base. These draft specifications, the first of their type, are the cornerstones for Stages II and III of the IMIS program.

STAGE II
Portable Computer-based Maintenance Aids System (PCMAS)

The PCMAS is an advanced development research prototype designed to demonstrate the concept of presenting automated technical data to maintenance technicians in a flightline environment. Field tests with PCMAS will examine problems involved in using a portable computer system in a flightline environment and establish requirements for a portable system for operational use. The PCMAS will demonstrate several concepts that are key to the successful implementation of IMIS.

In the shop, the PCMAS portable unit will be connected to peripherals to simulate a maintenance workstation to demonstrate exchange of information between ground based systems such as the Core Automated Maintenance System (CAMS), a base-level information management system, and the portable computer. The CAMS-like information will be stored on a large in-house computer system. Through the workstation, the technician will access information, such as the job location and work order, maintenance history of the aircraft, and equipment needed.

One of the biggest advantages to the technician will be the use of small memory cartridges to replace paper TOs. While present-day technicians may need to reference as many as ten paper-based TOs to perform a job, for example, removal and replacement of an F-16 engine, PCMAS users will require only two or three memory cartidges. The memory cartridges plug into the side of the PCMAS and may be swapped interactively as they are called upon by the program.

PCMAS will demonstrate interactive diagnostics. The PCMAS device will plug directly into the aircraft system bus, take over as bus controller, interrogate on-board systems for stored fault data, and run manual and built-in-tests. Efficient testing procedures will be maintained through generic diagnostic software which insures optimal use of tests based on their run-times and fault coverage. The software also examines diagnostic factors such as maximum aircraft downtime and available supplies.

PCMAS will provide specialized technical information to assist in Aircraft Battle Damage Repair (ABDR) assessment. This information will allow a single technician to accomplish the assessment task so that specialists in each area (structural, electrical, and airplane general) are not required. An expert system on an ABDR cartridge will supply the necessary task information. The PCMAS will also have peel-away graphics capabilities to allow the technician to determine what is behind the skin of the aircraft or behind different LRUs on-board the aircraft without removing them. This helps the technician identify mission-critical components such as subsystems, wire bundles, hydraulic lines, and structures in the path of the projectile and suggests quick checks to determine the status of those components. The time savings of looking for critical components with computer graphics versus manually cutting into the aircraft and removing hardware is obvious.

PCMAS will be used in a field test to help determine requirements for the use of a portable computer in the flightline environment. Hardware features such as size, weight, multiple power sources, power consumption, speed, screen resolution, and ruggedness will be evaluated.

STAGE III
Full IMIS Demonstration

The IMIS concept consists of four major subsystems: 1) the technician's portable computer/display; 2) an aircraft maintenance panel connected to on-board computers and sensors; 3) a maintenance workstation connected to various ground-based computer systems; and 4) sophisticated integration software which will combine information from multiple sources and present the data in a consistent way to the technician.

The technician's primary interface with IMIS will be the extremely portable, battery powered unit which is rugged enough for flightline use (Figure 4). A library of removable memory cartridges will store all the technical order information and diagnostic aids needed for one weapon system. The memory cartridges will be designed for fast, easy, and accurate updating. A high resolution, flat panel display will clearly display data under all lighting conditions. The man-machine interface will be designed for ease of operation to eliminate the need for the user to have typing skills. The portable computer will have the processing power to quickly display complex graphics and provide rapid response to the technician's requests. Interactive troubleshooting routines and artificial intelligence-based diagnostic aids will provide advice for difficult fault isolation problems. (It is important to point out that the portable computer will function independently to display most of the information the technician needs for on-equipment maintenance. Even if the base-level computer systems are unavailable or the aircraft systems are malfunctioning, the computer will be able to display technical order information and diagnostic aids to the technician.)

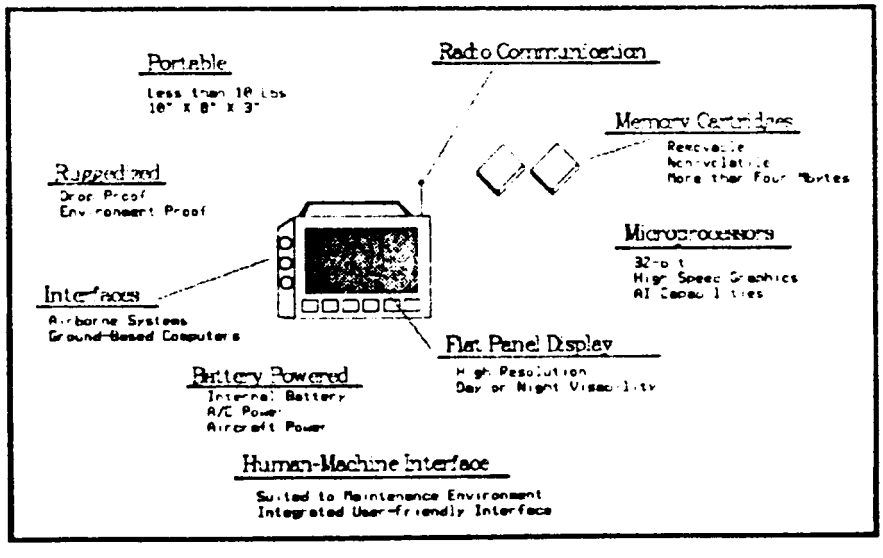


Figure 4: Portable Maintenance Computer Concept

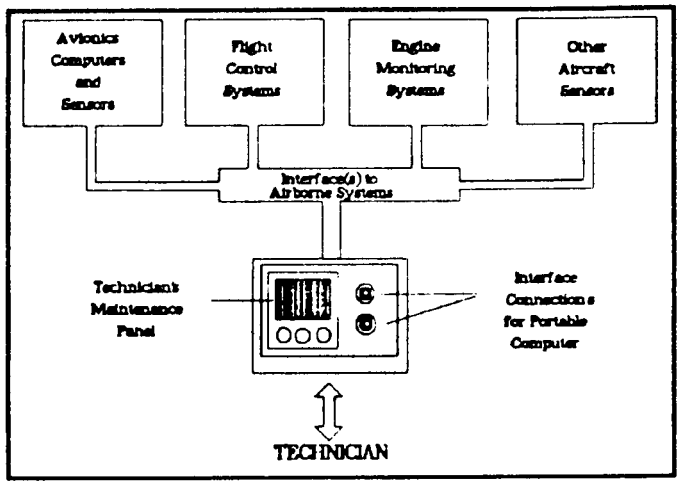


Figure 5: Aircraft Maintenance Panel

The technician will be able to perform most aircraft maintenance tasks without climbing into the cockpit. An aircraft maintenance panel on the outside of the aircraft will provide the interface with onboard systems (Figure 5). The portable computer will be able to retrieve and analyze flight information, interrogate or control available built-in-test systems, or input test signals for diagnostics. The interface panel will also be used to upload or download mission configuration/capability information.

The technician will interface with ground-based systems through a maintenance workstation (Figure 6). The desktop workstation will include a keyboard, a prin-

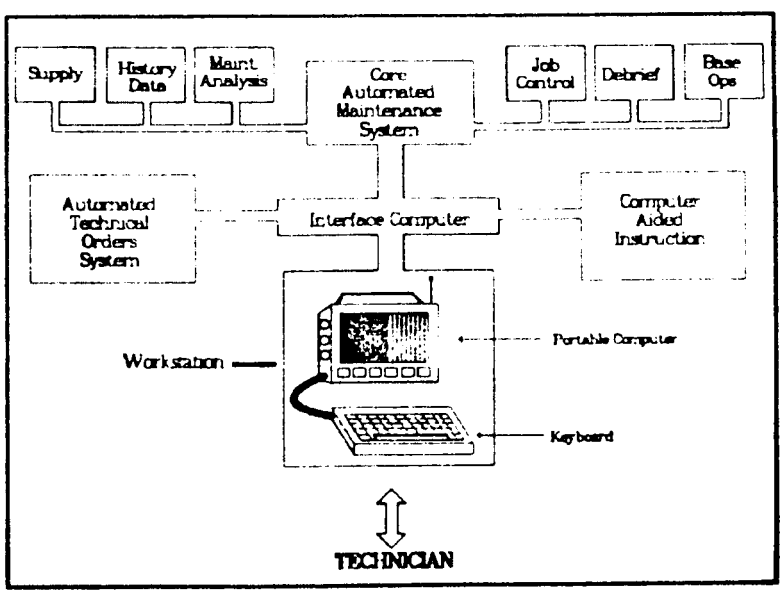


Figure 6: Maintenance Workstation

ter, and a computer interface. The interface will have the protocol software required to access the other available data systems. The portable computer will connect to the workstation and provide the display and processor for the workstation. The technician will then be able to access and exchange information with systems like the CAMS and ATOS.

The most beneficial feature for the technician will be the integration of information. Instead of dealing with several automated systems and accessing separate groups of information through several devices, the technician will access all information through one device (Figure 7). At a superficial level, the system will integrate information by employing standard commands and display formats. At a deeper level, through sophisticated software, the system will integrate information from all available sources to provide a coordinated maintenance package.

The development of the full IMIS demonstration will proceed in four phases. During the first phase, a structured analysis methodology will be used to determine an information system architecture. This architecture will define requirements for users' information needs, for interfaces, and for functional implementation. The second phase will be the hardware and software analysis, design, and review. Hardware fabrication and software programming, along with system tests and reviews, will occur during the third phase. Finally, in the fourth phase, the system will be evaluated in the operational environment by Air Force maintenance technicians. The product of the IMIS effort will be field tested and validated so that specifications for implementing this maintenance concept on Air Force weapon systems can be drafted.

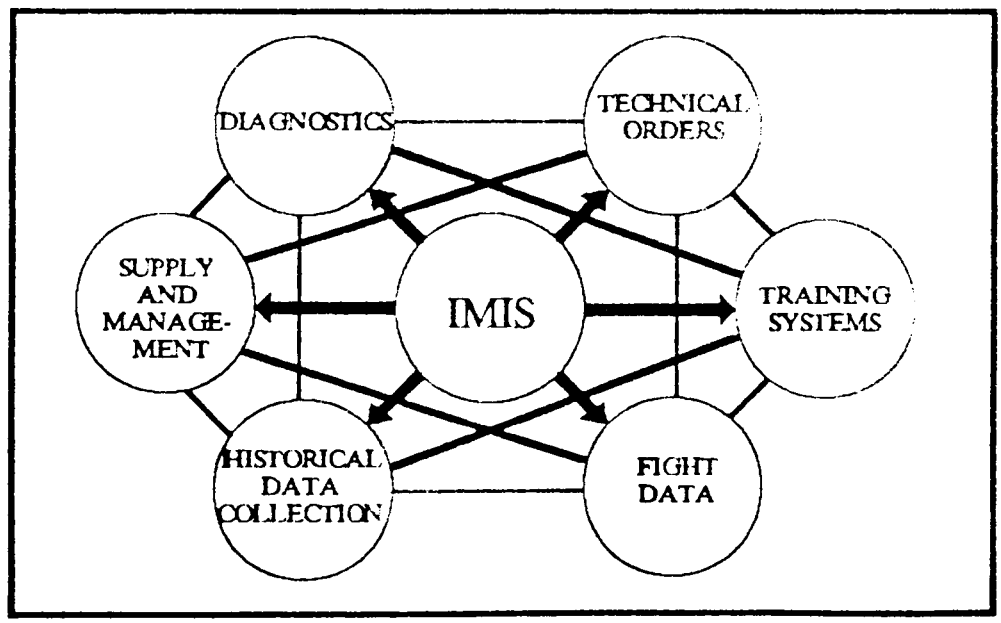


Figure 7: IMIS Information Integration

Conclusion

IMIS will be the culmination of a complex, thorough research and development project combining the skills and studies of numerous people and their projects. IMIS will be only the beginning for this new Air Force maintenance concept. IMIS will optimize the use of available manpower, enhance technical performance, improve training, and reduce the support equipment and documentation needed for deployment. It will serve as the technician's single, integrated source of all the technical information required to perform modern aircraft maintenance. The Air Force Human Resources Laboratory believes that IMIS will improve maintenance capability, productivity, and morale.

Air Force Human Resources Laboratory
Logistics and Human Factors Division
Combat Logistics Branch

AFHRL/LRC; WPAFB, OH 45433-6503
(513) 255-2606

SPACE SHUTTLE ONBOARD NAVIGATION CONSOLE EXPERT/TRAINER SYSTEM

Lui Wang
NASA/JSC
Mail Code FM72
Houston, TX 77058

Dan Bochsler
Lincom
18100 Upper Bay Rd.
Houston, TX 77058

ABSTRACT

This paper describes a software system (ONAV) under development at NASA's Johnson Space Center in Houston for use in enhancing operational performance as well as training ground controllers in monitoring onboard Space Shuttle navigation sensors. Previous expert system development work at NASA Johnson has shown that mainstream expert system development must follow a mix of software and system engineering procedures to insure operational success and effectiveness. ONAV development reflects this trend toward following a structured and methodical approach to development. The ONAV system must deal with integrated conventional and expert system software, complex interfaces, and implementation limitations due to the target operational environment. An overview of the onboard navigation sensor monitoring function is presented, along with a description of guidelines driving the development effort, requirements that the system must meet, current progress, and future efforts.

INTRODUCTION

This paper describes a Onboard Navigation (ONAV) software system under development at NASA's Johnson Space Center (JSC) for use in enhancing operational performance as well as training ground controllers in monitoring onboard Space Shuttle navigation sensors. Previous expert system development work at NASA JSC has shown that mainstream expert system development must follow a mix of software and system engineering procedures to ensure operational success and effectiveness. ONAV expert system development reflects this trend toward following a structured and methodical approach to development. The ONAV system must deal with integrated conventional and expert system software, complex interfaces, and implementation limitations due to the target operational environment. An overview of the onboard navigation sensor monitoring function is presented, along with a description of guidelines driving the development effort, requirements that the system must meet, current progress, and future efforts.

The Guidance and Onboard Navigation section (DM34) has a requirement to develop an expert/trainer system to assist in the training of Mission Control Center (MCC) onboard ONAV console operators during periods other than integrated simulations. This system is expected to evolve into a console assistant with the potential for increasing operations support effectiveness. The Space Shuttle Orbiter ONAV system is relatively stable and mature with limited new design/developments anticipated. The ONAV expert/trainer involves numerous aspects of current and future MCC functional elements including workstations, local area network (LAN) interfaces, telemetry (systems) and ground (trajectory) data, and crew and ground procedures. This situation ensures that the ONAV expert/trainer system, providing experience with a majority of the aspects of anticipated MCC functions, will benefit future Space Transportation System (STS) and Space Station operations.

ONBOARD SPACE SHUTTLE NAVIGATION

The purpose of the ONAV system is to estimate the Space Shuttle Orbiter's position and velocity (called the state vector). This is done by computing or measuring vehicle acceleration and numerically integrating it to obtain velocity and position. At various times, position measurements from outside sources are used to improve the state vector estimate.

During the descent phase of Space Shuttle flight where the vehicle comes from Earth orbit down to a landing site, the navigation system uses several position measurements to improve position estimates. Drag altitude is a very rough measurement which uses inertial measurement unit (IMU) sensed acceleration and an atmosphere model to estimate the altitude. Tactical air navigation (TACAN) measures the slant range and magnetic bearing from a ground station to the Orbiter. The air data system uses air pressure probes to measure the static atmospheric pressure, and compute an altitude measurement called baro altitude. Finally, the microwave scan beam landing system (MSBLS) provides measurements of slant range, azimuth angles, and elevation angles from ground transmitter stations located near a runway.

To achieve some degree of fault tolerance, the Orbiter contains three redundant IMUs, three TACAN transceivers, four air data sensors, and three MSBLS transceivers. Each piece of hardware is called a line replaceable unit (LRU). For each of these hardware systems there is a redundancy management (RM) software program in the onboard Shuttle computers. The RM has the task of choosing one set of measurements from the available sources and detecting and isolating failures in the hardware.

THE ONBOARD NAVIGATION CONSOLE TASK

The job of the ONAV console monitor is to assess the health of the various components of the ONAV system, and recommend actions to improve or maintain navigation accuracy. In performing this task, at entry ONAV uses onboard navigation data telemetered to the MCC, and the "ground state" (an independent estimate of the orbiter state.) The ground state is computed using ground radar measurements. IMU monitoring is based on comparisons of IMU attitude and velocity data, as well as comparisons with ground computed values. Possible recommendations include deselecting or reselecting IMUs. TACAN monitoring is based largely on comparisons of LRU measurements with the ground and with each other. Possible recommendations include using or not using TACAN data, deselecting or reselecting a TACAN LRU, or switching to a different TACAN station. ONAV has very little visibility into the air data system, so comparison of the baro altitude measurements with the ground is the main monitoring tool. Possible recommendations include using or not using baro altitude data. MSBLS monitoring is based on comparisons of LRU measurements with the ground and with each other. Possible recommendations include forcing TACAN to override MSBLS, or powering off a MSBLS LRU.

DEVELOPMENT GUIDELINES

This section presents a brief description of the development guidelines that impact the ONAV expert system.

Problem Domain

The ONAV expert system will consist of four distinct components corresponding to the following four Shuttle mission phases: ascent, onorbit, deorbit, and entry.

Knowledge Base

Development of the rules for the ONAV system will be generated and documented as four separate knowledge bases corresponding to each mission phase. The expert system will incorporate the concept of modular design to logically partition both data and rules in order to promote and enhance program development and extensibility. The following sources of information will be utilized, as appropriate for expert system knowledge base development: ONAV console checklists, ONAV display user's guides, and ONAV console personnel.

Development Environment

The expert system application software will be executable in a language available in a workstation environment. However, an expert system shell written in C called CLIPS will be used.

Documentation

a) Expert system software code: The expert system software will include comment text, to the maximum extent practical, according to proposed documentation standards for expert systems being developed at the JSC. Further, the comments will be enhanced through the use of long, descriptive variable names, labels, etc.

b) Guidelines and system requirements: This document is a top level overview of the ONAV development effort. This information is critical to providing proper direction to the project. Availability of this information not only provides a means to communicate to others not involved in the project, but also serves as a historical document. For very complex and detailed efforts, such a document serves as the first step in maintaining traceability and configuration control of software products.

c) Knowledge requirements: The target audience for this document is the knowledge domain expert. It is a reflection of "what the system knows" in a form as close as possible to the expert's language.

d) Design: This document is intended for use by the implementers of the expert system and will serve as a guide for the coding effort. Contents will include such things as fact formats, data representation, rule groupings, control flow, execution flow, interfaces, etc.

e) User's guide: The user's guide will present procedures for preparation, operation, monitoring, and recovery of the expert system. The user's guide will be based upon the design specification and is intended for the specific use of the users. It will include procedures for system operation directly in support of operational tasks.

f) Test plan: The test plan defines the total scope of the testing to be performed. It identifies the particular levels of testing and describes the contributing role for ensuring the reliability and acceptance of the system. It identifies the degree of testing and the specific functions that are involved in the tests. The test plan is for

reviewing and ensuring that the technical requirements are met.

Operation Modes

The ONAV expert system will operate in either of two modes. In the first mode, referred to as "closed loop," the system will be used with operational data. The operations environment will consist of integrated simulations. When the expert system is certified as accurate and reliable, it will be used in the actual mission environment. In the second mode, referred to as "open loop," the system will be used for initial level training and familiarization purposes using existing data tapes from several sources.

Timing

The expert system will provide outputs in a real-time telemetry/trajectory environment and system timing will be structured accordingly. Real-time data rates will apply to the expert system as ported to the workstation environment and not apply to development machines, if different. The expert system will assume data is available at approximately 2 second intervals.

Display Definitions

The primary function of the display will be to depict recommendations from the ONAV expert system to a ground controller. The goal is to provide an easily interpreted, quick-look format that shows the current status of the overall system, the status of individual subsystems, and recommended navigation system actions.

DESIGN OVERVIEW

The overall environment in which the ONAV expert system will operate is illustrated in figure 1. Although different mission phases may have different functional structures, as an example, the ENTRY architecture for the expert system is depicted in figure 2. This structure results from the basic nature of the ONAV task at the descent phase of the mission. Four functional components of the expert system are identified: 1) fact assertion, 2) monitoring, 3) analysis, and 4) output. In addition, two non-expert system components which are a part of the overall ONAV system called "computations" and "data preparation" are also shown in both figures. The computations component receives information from the operational environment of the MCC LAN and performs various computations such as scaling, state vector propagation, coordinate system transformations, etc. The prime purpose is to make information uniform in time (i.e., homogeneous), which is not necessarily the case with raw data from the local area network. The data preparation component receives information from either the operational environment or training tapes and performs three functions: (a) collects the information required by the expert system, (b) performs any additional computations required on the

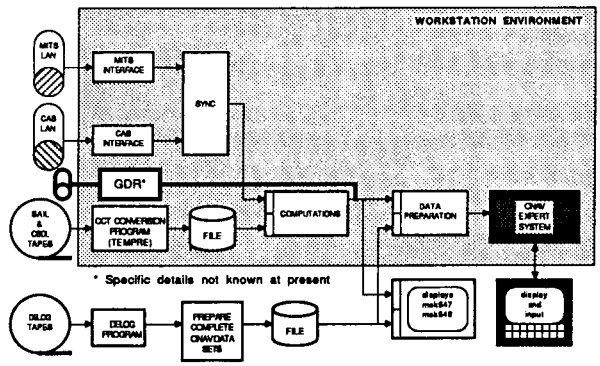
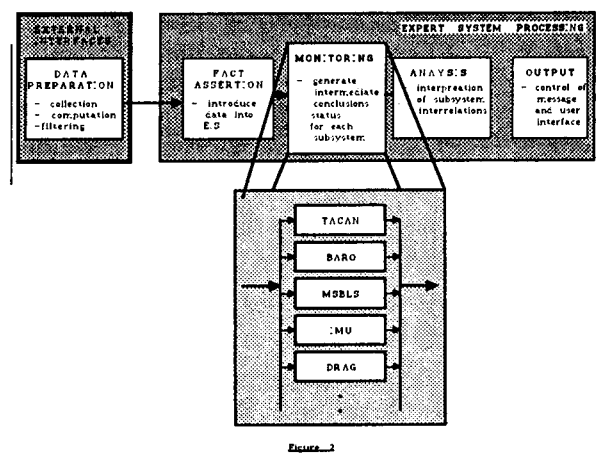


figure 1



data, and (c) filters and transforms that data into a form suitable for the expert system. Non-discrete numeric data are compared to thresholds and converted to "symbolic" forms whenever possible.

The fact assertion component takes the prepared data and puts that required by the expert system into the expert system fact base and the remaining data into the C software environment for background processing and reference. The monitoring component generates intermediate conclusions and statuses of the individual subsystems ONAV observes and manages. The analysis component performs an overall assessment of the current situation taking into account interrelationships between subsystems. The output component controls the sending of notices and/or recommendations to the ONAV expert system console.

CURRENT PROGRESS AND FUTURE PLANS

The ENTRY ONAV prototype was completed in June 87. A complete version of the ENTRY knowledge base design is currently under development. The ENTRY knowledge base line document was also completed and is waiting for final publication. The RENDEZVOUS ONAV is also well under way, a set of nominal rules are written and currently under design evaluation. Development work on the ascent phases of ONAV was initiated in July.

REFERENCES

1. Onboard Navigation Ground Support Expert/Trainer System, JSC Memorandum # DM3-86-32, J. Harpold, July 9. 1986.

2. "Guidelines and System Requirements For the Onboard Navigation (ONAV) Console Expert/Trainer System," Mission Planning and Analysis Division, Mission Support Directorate, NASA Johnson Space Center, Internal Note JSC-22433, December, 1986.

3. "Software Engineering Techniques Used to Develop an Expert System for Automated Space Vehicle Rendezvous," Bochsler, Daniel C., and Mary Ann Goodwin, Proceedings of the Second Annual Workshop on Robotics and Expert Systems, June 1986, Instrument Society of America, Research Triangle Park, NC.

4. "A Prototype Real-time Expert System to Monitor Space Shuttle Rendezvous Navigation," Goodwin, M. A., and B. I. Talisman, Proceedings of the Second Annual Workshop on Robotics and Expert Systems, June 1986, Instrument Society of America, Research Triangle Park, NC.

5. CLIPS Reference Manual, Version 3.0, July 1986, AI Section, NASA Johnson Space Center.

MPAD MCC WORKSTATION SYSTEM: ADDAM AND EEVE

Bill Kozick, Jr. Walt Reynolds

COMPUTER SCIENCES CORPORATION
Applied Technology Division
16511 Space Center Blvd.
Houston, TX 77058
(713) 486-8153

On behalf of NASA/JSC/MSD/MPAD/FM7/Roy E. Stokes

ABSTRACT

The flight control consoles in mission control at NASA's Johnson Space Center (JSC) have evolved in sophistication to the level where "expert sytems" are now being incorporated into them as "flight controller assistants."

This paper describes the evolution of a gateway node, designed to obtain and redistribute numerous kinds of data provided by Mission Control computers over a laser-optical network to enable rapid-prototyping development of the above application expert systems.

This automated data distribution and management system serves as an effective buffering system for assuring the necessarily separate requirements of the operational and developmental environments. This is accomplished through the evolutionary enhancement of the gateway's ancillary monitoring and control expert system that was originally designed to "watch and react" to system anomalies in the operational state, but whose role has been substantially expanded.

BACKGROUND

The Mission Control Center (MCC), at NASA/JSC has traditionally provided flight data via digital tape to applications subscribers outside of the center. Although called near-realtime data, the time delays required to prepare the tapes limited their use for playback and post-flight analysis modes. A growing demand for timely data during the flights necessitated the development of a local area network (LAN) to ship to sites remote from the MCC the kinds of data that had only been available to the MCC realtime support laboratories adjacent to the MCC.

A prototype laser-optical LAN was installed to test the feasibility of providing high-quality realtime and near-realtime data to remote subscribers. Known as the MITS LAN, its two-year test program has proven successful, enabling the system to be brought up to operational status. This means that remote nodes will be able to serve as both developmental and operational extensions of the MCC. A rigorous system of configuration management has been designed and is in the process of being installed to ensure that only properly verified and validated applications programs are maintained within the operational MCC environment.

Not surprisingly, the new concept of the MCC distributed LAN provided a resource that piqued the imaginations of Mission Planning and Analysis Division (MPAD) personnel working in the environment of developing expert systems for use as "assistants to" flight controllers. They came to view their node of the MITS LAN as a means to both develop their prototype programs and to eventually run them in actual operational states. At the same time the level of sophistication of the anticipated programs accelerated.

Other groups within MPAD were given the responsibility to ensure that data emanating from the MCC-distributed LAN node to MPAD were timely and of sufficient reliability to ensure meaningful development of expert systems and other application programs. A prototype data distribution and management workstation concept was devised in conjunction with the prototype of the MITS LAN. This paper describes the three phases of data workstation development for the automated data distribution and management (ADDAM) system and the design of an associated monitoring and control expert system, the Effective Evaluation Expert (EEVE), that is responsible for ensuring data integrity.

THE FIRST PROTOTYPE

Figure 1 depicts the initial ADDAM workstation concept devised in conjunction with the proof-of-concept MITS LAN. Workstation software to accommodate one "walk-up" user at a time was provided on a HP9000 minicomputer. Through a series of displays, menus and entryforms, the user selected types of data desired and a storage medium for that data (either disk files or digital data tapes). Gateway software was provided on the HP9000 that serves as one of the MITS LAN nodes. The gateway's function was to ensure that only appropriate data requests were transmitted to the MCC and within the correct design frequency load limits.

The idea for the addition of an expert system to this configuration resulted from certain anomalies within this earliest version of the MITS LAN. Frequent abrupt data terminations and dropouts necessitated development of a monitoring and control system to assess system integrity and provide remedial actions as required. Written in the programming language LISP, the Effective Evaluation Expert (EEVE) was run concurrently on a Symbolics 3640 tied to the gateway HP9000 via a direct cable/-RS232 interface. In this simplest of expert system configurations, EEVE provided what could be termed "operator emulation." Figure 2 shows "the face of EEVE" as manifested within the graphics environment available on the Symbolics computer. Status update windows and mouse activation fields were provided to enable observation of the performance of the expert system during operation.

Although both the MITS LAN and workstation system prototypes proved effective, both necessarily were limited by the proof-of-concept design constraints. The prototype workstation system could only serve one user at a time and data could only be stored before usage.

THE CURRENT PROTOTYPE

The increasing sophistication of the user/application community during the test phase of the MITS LAN prototype necessitated an expansion of the working concept of the MCC workstation system to accommodate their needs. Figure 3 shows the enhanced version of the workstation system prototype currently under assembly. Notable in this design are increased responsibilities and capabilities for all previous processors as well as the addition of new processors, specialized workstations, and improved techniques for moving data and information over the local MPAD LAN.

On the workstation side of the MPAD LAN, multiple MCC Application workstations are separate from the MCC Data Management workstation; the latter comprising a Britton Lee database machine hosted by a HP9000. The data management workstation is utilized to store data types and sequences for playback during specialized analyses or during flight simulations.

Each application workstation can provide data directly to automated applications, particularly realtime expert systems, on a datastream basis, through the new Enhanced Development Environment Networked Node (EDENN) processors in conjunction with the expanded capabilities of the ADDAM gateway.

On the gateway side of the local MPAD LAN, the ADDAM processors have been upgraded to service multiple workstations with an expanded list of data types and variations. Within the automated maintenance monitoring and control milieu, the EEVE systems design role now calls for hypothesis construction and testing beyond mere operator emulation. In addition, the EEVE system will be rehosted from the Symbolics 3470 (in LISP), to the same HP9000 containing the ADDAM processors (in the inferencing engine/language CLIPS), that also serves as the node terminal to the MITS LAN.

THE ADVANCED PROTOTYPE

Even as the current prototype is under construction, demands for increased performance have been requested from both the data provider side of the chain (the system herein described), and the data utilizer side (the application expert systems). Consequently the plan for an even more advanced operational structure is now also in preparation. Figure 4 illustrates the increased complexity over the current prototype.

It is important to note the addition of a generalized communications interface technology among the gateway and the workstations. Called the Remote Information Interchange Buffer (RIIB) processors, they intermediate among the data-oriented executives (ADDAM and EEVE) and the networks utilized to transfer data and advisories. As such, the RIIB's serve as "session managers" on behalf of the executives within a User Datagram Protocol/Internetwork Protocol (UDP/IP) networking environment. Also planned for the RIIB's are generalized display terminal support functions that allow for the dynamic reconfiguration of terminals within the Transmission Control Program/Internetwork Protocol (TCP/IP) network environment. Figure 5 illustrates that using the accepted International Standards Organization Open Systems Interconnection Protocols (ISO OSI) network commun-

ications scheme, both ADDAM and EDENN comprise Application Level 7, whereas the RIIB occupies the Session and Presentation Levels, 5 and 6.

This new uniformity in communications enables EEVE to take on the added task of network failure response. In this mode, EEVE can restart applications by rerouting the users' terminals via the network to the new hosting node. To accomplish this, EEVE accesses the Data Management workstation database for a local configuration management library.

Both the ADDAM and EEVE systems have full backup versions, as shown in Figure 4, that are fully ready to step in should the "master" gateway/monitor fail. When the backup EEVE senses that the current master has failed, it promotes the backup ADDAM to master status, reboots, and arranges for the switchover of data and advisories to any and all workstations waiting in abeyance. New backup versions of ADDAM and EEVE are then created and placed in readiness.

Specialized versions of EEVE, called EEVE II, have been added to the workstations to provide the levels of interpretation of the flight-specific environment for the online application expert systems. This focuses EEVE's duties on the gateway to levels of interpretation regarding flight control host activities and network and performance configuration.

Importantly, this design embodies two parallel and interdependent means of communication as symbolized by the solid and dotted lines. The solid lines, connecting the realtime data and advisory processors, including ADDAM and EDENN, symbolize the autonomic neuralnet system (ANS) side of the entire configuration. Conversely, the dotted lines represent a "virtual LAN" that connects EEVE, EEVE II and even the online application expert systems and, as such, symbolize the central neuralnet system (CNS) side of the configuration. This design allows the CNS to perform symbolic processing without hindering the realtime response of communications interfaces. It also enables the various expert systems to converse, negotiate, and even "argue" about network priorities and flight environment realities.

AN EXPERT CONVERSATION

The following scenario is presented to illustrate the levels of interpretation that will exist in the symbolic discourse between application expert systems and EEVE/EEVE II:

The application E/S innocently asks about the status of its flight (fearing the worst since data is missing) to the local EEVE II system. EEVE II discovers that there are holes in its fact database and makes further inquiries to EEVE at the gateway.

The application system would begin with a message to the EEVE II on the local workstation:

FROM: **Application E/S: USER43**
TO: **EEVE II @ MCC**
Workstation: FM8

(QUERY FLIGHT-STATUS)

(REQUESTOR SESSION USER43)

EDENN automatically adds the session identifier, USER43, as the inquiry is forwarded to EEVE II.

EEVE II consults its database,

(DATA-REQUIREMENT USER43 NRT)

(DATA-REQUIREMENT USER43 TRJ

(CYCLIC-STREAMS

(HIGH-SPEED-MISC))

(DEMAND-STREAMS

(ATTITUDE-TIMELINE

VECTOR-ADMIN-TABLE)))

(DATA-REQUIREMENT USER43 CAS)

(FLIGHT-LOGON USER43 FLIGHT 71-B)

and concludes that it needs to consult EEVE at the gateway about the status of the Near Realtime Telemetry (NRT), trajectory (TRJ), and Calibrated Ancillary Telemetry System (CAS) data streams for flight 71-B.

Thus EEVE II sends the following message to EEVE:

FROM: **EEVE II @ MCC**
Workstation: FM8
TO: **EEVE @ MCC Gateway**

(QUERY DATA-STREAM-STATUS FLIGHT

71-B NRT)

(QUERY DATA-STREAM-STATUS FLIGHT

71-B TRJ)

(QUERY DATA-STREAM-STATUS FLIGHT

71-B CAS)

(REQUESTOR WORKSTATION FM8)

The last fact is added by ADDAM as the message is forwarded to EEVE.

Correspondingly, EEVE consults its database at the gateway,

(DATA-STREAM NRT FLIGHT 71-B STATUS ACTIVE AT 17:08)
(DATA-STREAM TRJ FLIGHT 71-B STATUS UNKNOWN AT 17:05)
(DATA-STREAM CAS FLIGHT 71-B STATUS PENDING AT 17:07)

(CONTINGENCY FLIGHT 71-B TRJ NO-RESPONSE AT 17:05)
(CONTINGENCY FLIGHT 71-B TRJ CHECKPOINT-WARNING AT 16:58)
(CONTINGENCY FLIGHT 71-B CAS RESTART-REQUESTED AT 17:07)

and would have to shrug its allegorical shoulders if it were not for the sudden entry of a new fact and the expiration of a timer (all sensed by ADDAM) --

(CONTINGENCY FLIGHT 71-B CAS RESTART-ACKNOWLEDGED AT 17:10).

After a few simple rules fire EEVE then has the following facts with which to resolve the question:

(RESOLUTION FLIGHT 71-B CAS RESTART-CONFIRMED AT 17:10)
(RESOLUTION FLIGHT 71-B TRJ CHECKPOINT-ASSUMED AT 17:10)

(DATA-STREAM CAS FLIGHT 71-B STATUS IN-RESTART AT 17:10)
(DATA-STREAM TRJ FLIGHT 71-B STATUS (CHECKPOINTED .95) AT 17:10).

The rule operating for the pending status on the CAS data stream is clear enough. However, the assumed checkpoint rule is obviously one of interpretation -- guessing -- on the part of EEVE. Thus EEVE asserts that the flight control host system supporting the trajectory data stream must be in the grips of a checkpoint procedure. EEVE further notes that it does not know this with certainty, but rather that it feels reasonably sure (.95) that this must be the case since a checkpoint warning was received some 12 minutes previously.

It is not implied that EEVE is implemented using fuzzy logic. EEVE simply provides a confidence level for its deductions that may or may not be used by an EEVE II or application expert system when making further decisions.

At any rate, EEVE returns what it knows about flight 71-B to the FM8 workstation:

FROM: EEVE @ MCC Gateway
TO: EEVE II @ MCC Workstation: FM8

(RESPONSE-TO WORKSTATION FM8 AT 17:10)
(DATA-STREAM-STATUS FLIGHT 71-B NRT ACTIVE AT 17:08)
(DATA-STREAM-STATUS FLIGHT 71-B CAS IN-RESTART AT 17:10)
(DATA-STREAM-STATUS FLIGHT 71-B TRJ (CHECKPOINTED .95) AT 17:10)

The EEVE II at FM8 now has enough information to respond to USER43:

FROM: EEVE II @ MCC Workstation: FM8
TO: USER43 @ MCC Workstation: FM8

(RESPONSE-TO SESSION USER43 AT 17:10)
(FLIGHT-STATUS FLIGHT 71-B ACTIVE AT 17:10)

While the application system received an answer to its question, it turns out that USER43 is sophisticated enough to ask even more detailed questions:

FROM: USER43 @ MCC Workstation: FM8

TO: EEVE II @ MCC
Workstation: FM8

(QUERY DATA-STREAM-STATUS TRJ)
(QUERY DATA-TYPE-STATUS
(HIGH-SPEED-MISC
VECTOR-ADMIN-TABLE NRT))
(REQUESTOR SESSION USER43).

To which EEVE II can directly respond:

FROM: EEVE II @ MCC
Workstation: FM8
TO: USER43 @ MCC
Workstation: FM8

(RESPONSE-TO SESSION USER43)
(DATA-STREAM-STATUS FLIGHT 71-B TRJ
(CHECKPOINTED .95) AT 17:10)
(DATA-TYPE-STATUS FLIGHT 71-B
((HIGH-SPEED-MISC CYCLIC
HALTED AT 17:10)
(VECTOR-ADMIN-TABLE ON-DEMAND
HALTED AT 17:10)
(NRT ON-DEMAND ACTIVE AT
17:08)))!!

IN CONCLUSION

It is evident from the foregoing dialogues that much work has yet to be done to reach the level of sophistication implied by the symbolic information transfer that is taking place among the expert systems. It will be no small challenge to build the domains of interpretation required of the expanded EEVE and the new EEVE II's. Much proof-of-concept testing has yet to be inspired and embraced. Even before the bases for interaction are established among EEVE and the EEVE II's over the "virtual LAN," the servicing interfaces must be worked out between an EEVE II and the application expert systems that reside on its workstation. This, in itself, is a substantial undertaking because of the amount of time required to meet with the application expert system designers in order to understand their special needs and requirements. One such project, currently under way, has resulted in the requirement for a time synchronization processor to be added to an EDENN system.

Optimistically, once the central neuralnet is in place in even rudimentary form, thus linking all of the service and application expert systems together, the symbolic-oriented design will enable a continuous and evolutionary growth in capability as time and resources permit.

ABBREVIATIONS, ACRONYMS AND TERMS USED IN THIS PAPER

ADDAM - Automated Data Distribution and Management system
ANS - Autonomic Neuralnet System
ATT - Attitude Table data
CAS - Calibrated Ancillary System
CM - Configuration Management
CNS - Central Neuralnet System
COTS - Commercial Off-The-Shelf
DB - Database
DM - Data Manager
EDENN - Enhanced Development Environment Networked Node
EEVE - Effective Evaluation Expert
E/S, ES- Expert System
HP9000 - Hewlett Packard minicomputer
IP - Internetwork Protocol
IPS - Instrument Pointing System
ISO - International Standards Organization
JSC - Johnson Space Center
LAN - Local Area Network
LLA - Link Level Access
MCC - Mission Control Center
MITS - MOD-IPS-TACAN System (LAN)
MNV - Maneuver table data
MOC - Mission Operations Center
MOD - Mission Operations Directorate
MPAD - Mission Planning and Analysis Division
MSD - Mission Support Directorate
NASA - National Aeronautics and Space Administration
NRT - Near Realtime Telemetry data
OSI - Open Systems Interconnection Protocols
RIIB - Remote Information Interchange Buffer
TACAN - Tactical Air Control and Navigation system
TCP/IP - Transmission Control Program / Internetwork Protocol
TRJ - Trajectory data
UDP - User Datagram Protocol
VAT - Vector Administration Table data
XNS - Xerox Network System

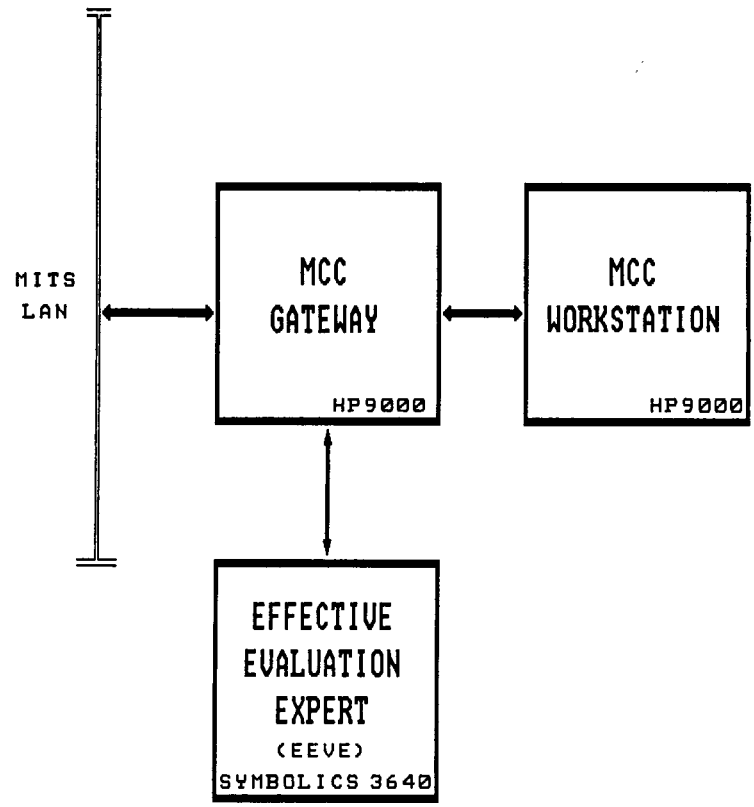


Figure 1 - The First Prototype

ORIGINAL PAGE IS
OF POOR QUALITY

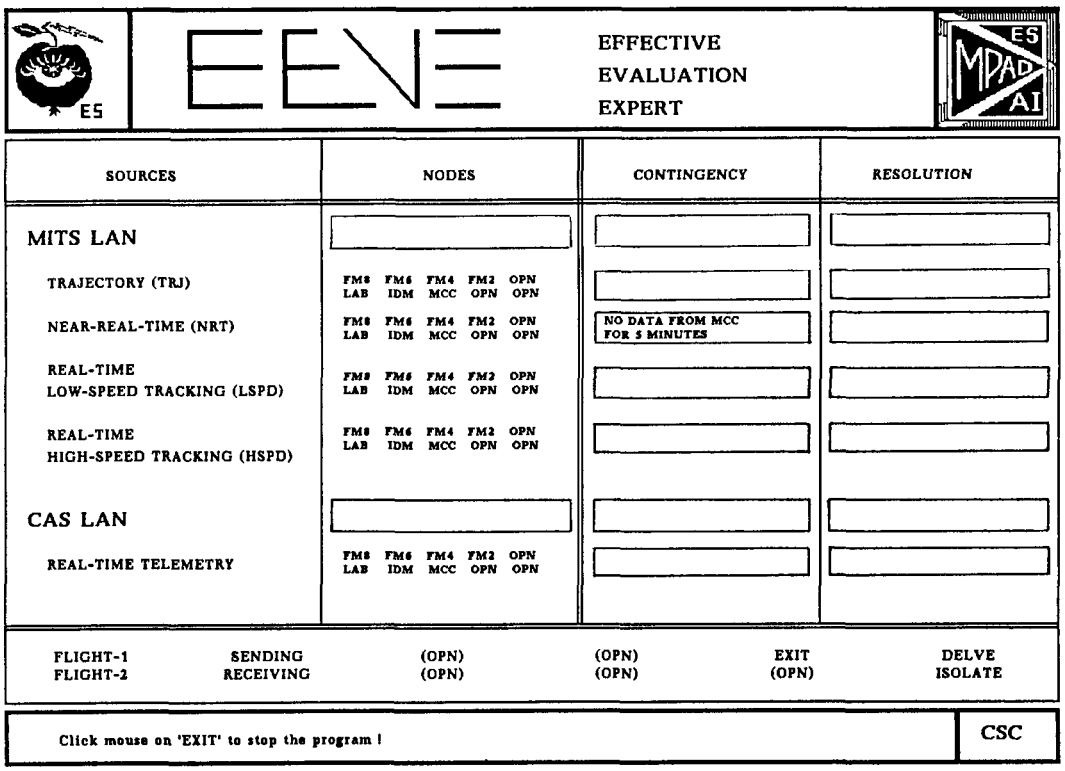


Figure 2 - The Face of EEVE

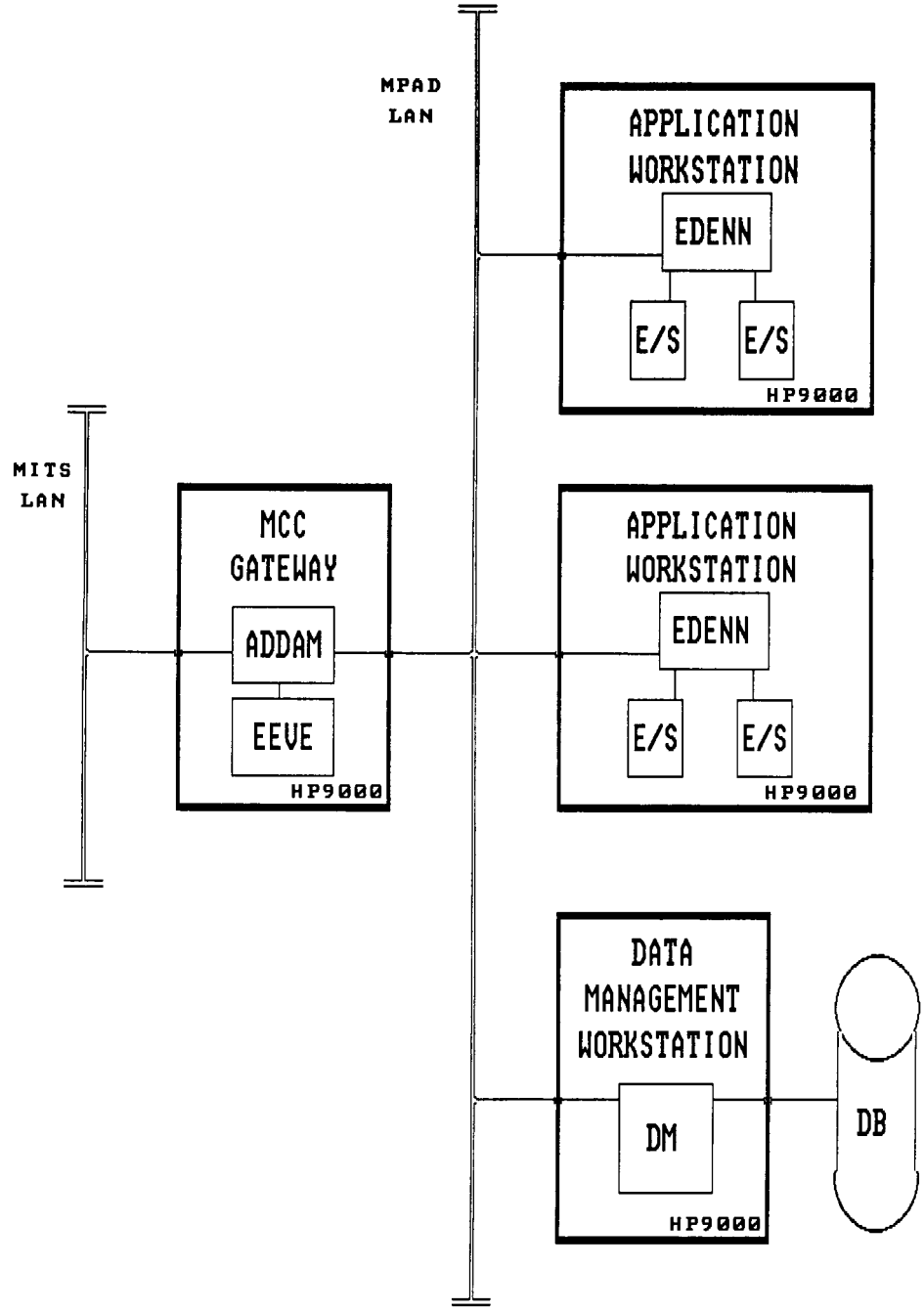


Figure 3 - The Current Prototype

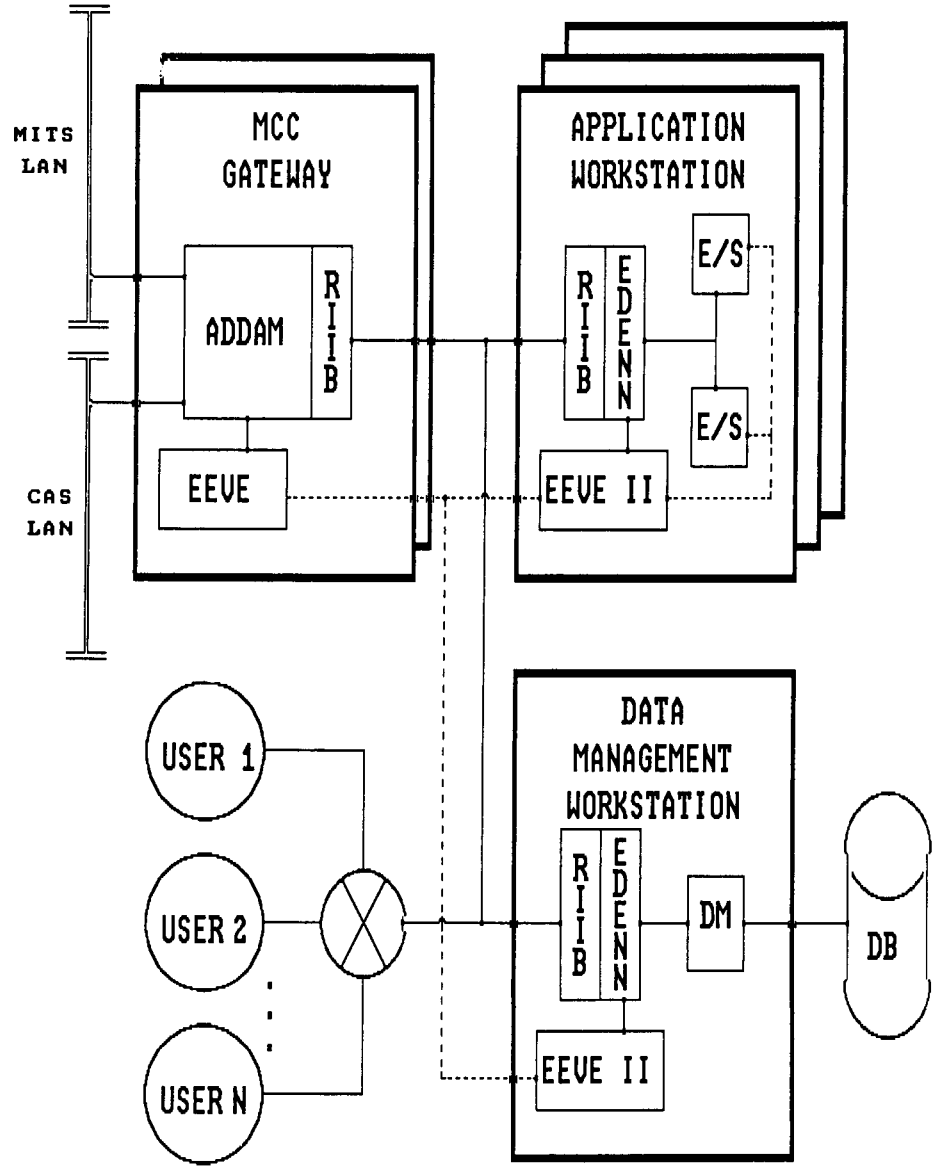


Figure 4 - The Advanced Prototype

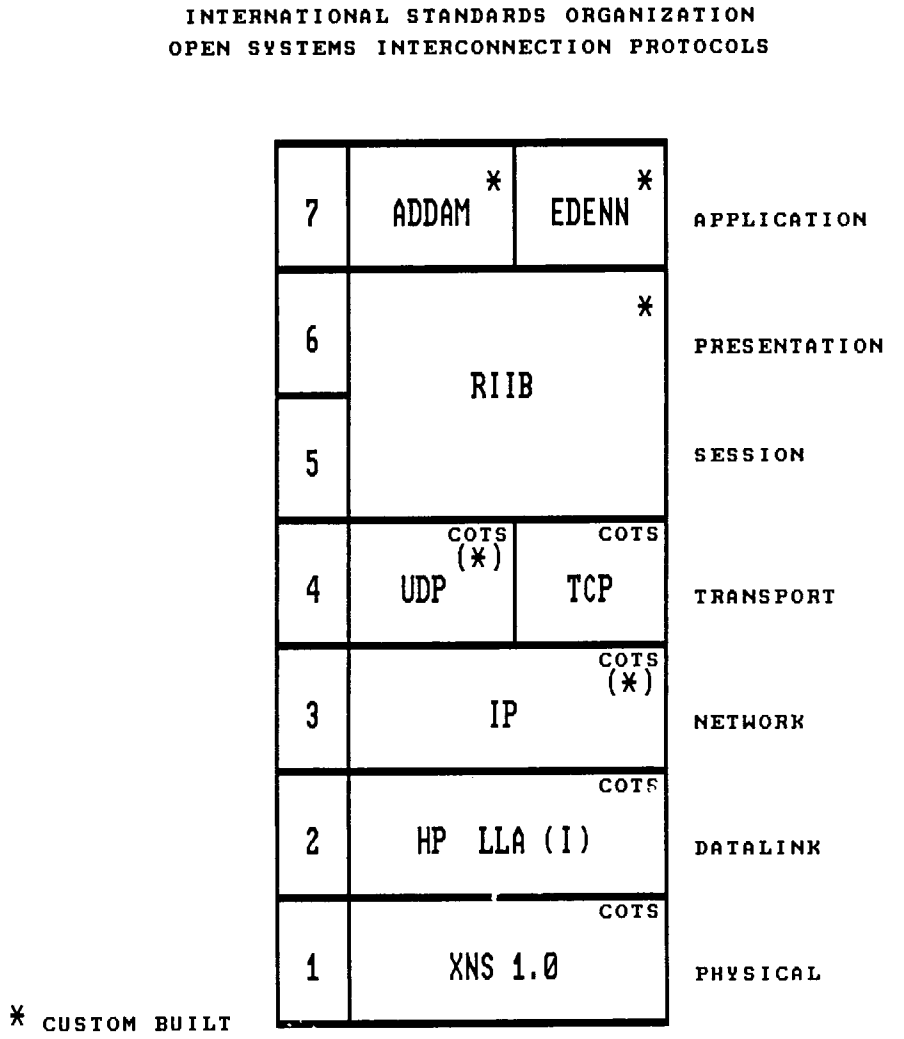


Figure 5 - The ISO OSI Network Communications Model

KNOWLEDGE BASED JET ENGINE DIAGNOSTICS

Timothy G. Jellison and Ronald L. De Hoff
Systems Control Technology, Inc.
Palo Alto, CA 94304
(415) 494-2233

ABSTRACT

A fielded expert system automates equipment fault isolation and recommends corrective maintenance action for Air Force jet engines. The knowledge based diagnostics tool was developed as an expert system interface to the Comprehensive Engine Management System, Increment IV (CEMS IV), the standard Air Force base level maintenance decision support system. XMAN™, the Expert Maintenance Tool, automates procedures for troubleshooting equipment faults, provides a facility for interactive user training, and fits within a diagnostics information feedback loop to improve the troubleshooting and equipment maintenance processes. The application of expert diagnostics to the Air Force A-10A aircraft TF-34 engine equipped with the Turbine Engine Monitor System (TEMS) is presented.

INTRODUCTION

XMAN is a knowledge-based software tool designed for advanced diagnostics support of complex aeromechanical equipment. Developed as an expert user interface to a large, historical maintenance database, XMAN automates procedures for troubleshooting equipment faults. The expert maintenance tool has been field tested and is operationally supporting flightline maintenance diagnostics for the Air Force A-10A weapon system.

XMAN represents a significant step forward in the evolution of maintenance information and integrated diagnostics systems. A means of improving the diagnostics process is provided through visibility into troubleshooting performed at the equipment level and feedback of information to the equipment manager. Interactive user training is provided in addition to the automation of the maintenance diagnostics function. The user remains a key factor in the equipment diagnostics process. Training and user acceptance of the expert system are facilitated by keeping the technician in the troubleshooting loop, while providing explicit diagnostics guidance and allowing ready access to data pertinent to the specific equipment fault.

Presented is a summary of troubleshooting performed by XMAN during its evaluation period and initial operations at eight Air Force A-10A bases. The specific application addressed is that of the TF-34 jet engine equipped with the Turbine Engine Monitor System (TEMS). Under Air Force contract, XMAN is the expert system interface to the Comprehensive Engine Management System Increment IV (CEMS IV), Engine Diagnostics and Trending (ED & T). This interface is discussed and a typical troubleshooting session presented. Design concepts underlying the expert system architecture are highlighted. The potential for expansion of XMAN to other aeromechanical equipment is addressed.

EVOLUTION OF XMAN AS AN AUTOMATED DIAGNOSTICS TOOL

The origins of XMAN date back to the early stages of the CEMS IV program (see Figure 1). CEMS IV is the standard Air Force jet engine management system for maintenance decision support. Systems Control Technology, Inc. (SCT) developed CEMS IV under Air Force contract to support the information intensive processes associated with On-Condition Maintenance (OCM). CEMS IV fuses data from a number of disparate DoD maintenance information systems including CAMS (Core Automated Maintenance System) and engine specific automated monitoring systems.

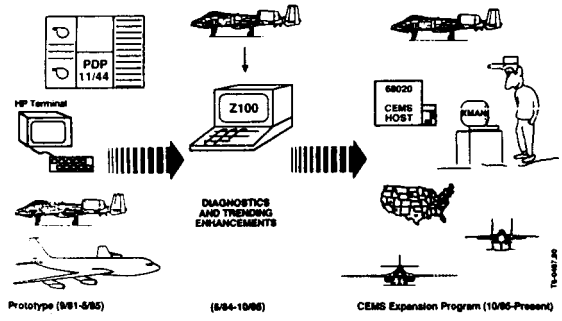


Figure 1 XMAN Development can be Traced Back to the Early Days of MIMS and CEMS IV.

CEMS IV is an outgrowth of SCT's Maintenance Information Management System (MIMS™). CEMS IV integrates data acquisition and processing functions to trend engine performance, display graphical diagnostics data products, and flag engine malfunctions (see Figure 2). Data are displayed to the flightline and intermediate maintenance technicians in a format which is easily usable and readily accessible. Only those data that pertain to that individual's work requirements are displayed. The prototype CEMS IV was evaluated in an operational environment at Barksdale Air Force Base, Louisiana (917th TFG, AFRES) for three and one-half years before Air Force direction was given to expand the system implementation. [1]

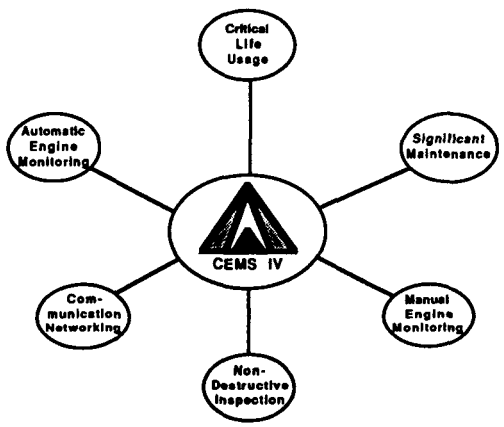


Figure 2 CEMS IV Integrates a Number of Disparate Maintenance Information Functions.

CEMS IV is currently undergoing expansion to twenty-one Air Force A-10A bases worldwide. Engine maintenance decision support for other aircraft types (e.g., F-16, B-1B, KC-135R, C-17) may be provided for up to sixty bases at Air Force option. Demonstrations of CEMS IV support have already been performed for the F100-220 engine (F-15 and F-16 aircraft) and the F108 engine (KC-135R aircraft). The CEMS IV software is approximately eighty percent generic, allowing flexibility in specific type-model-series (TMS) engine applications.

From the early stages of operational evaluation, it was evident that an expert system would enhance the CEMS IV man-machine diagnostics interface. SCT began the development of an expert system in early 1984. Under internal research and development funding, SCT produced an expert system kernel which serves as the XMAN software control system.

The software engineering principles guiding the development of MIMS (i.e., generic, table driven software, independent of specific hardware or equipment applications) also led to the development of a generalized expert system kernel. Tailoring is carried out through a process of knowledge engineering for the specific TMS engine or equipment application. XMAN is written in LISP and is resident on a microcomputer (PC compatible) operating under MS-DOS.

As a software refinement under SCT's CEMS IV contract with San Antonio Air Logistics Center (SA-ALC), XMAN's knowledge base has been tailored to troubleshoot and diagnose engine malfunctions on the TF-34 engine. The troubleshooting knowledge base is based upon the technical order (T.O.) logic trees developed by the engine manufacturer for analyzing engine malfunctions using CEMS IV. Thus, for the A-10A/TF-34 application, XMAN serves as a T.O. prompting system as well as an expert troubleshooting tool.

TYPICAL APPLICATION

The diagnostics procedures associated with interpreting CEMS IV data products, troubleshooting engine alarms, and recommending corrective maintenance action are automated by XMAN. The expert diagnostics function is a menu option on the CEMS IV workstation (see Figure 3).

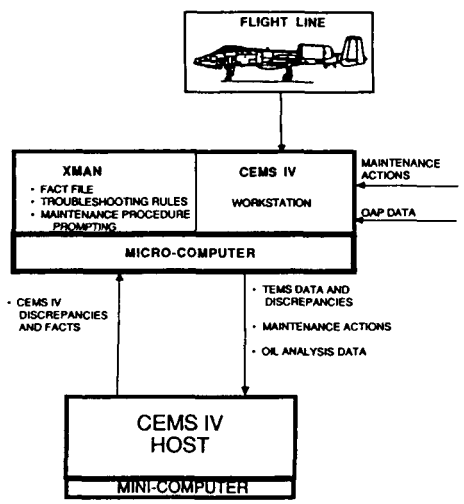


Figure 3 XMAN Resides on the Same Microcomputer as the CEMS IV Workstation.

XMAN troubleshoots engine discrepancies which are generated by both the on-aircraft engine monitoring system (TEMS) and CEMS IV. The discrepancies generated by TEMS are engine events (e.g., core overspeed, turbine temperature exceedance). TEMS-generated discrepancies are passed down to CEMS IV and stored in the host database. CEMS IV generates engine discrepancies based upon abnormal performance, wearmetal trends and parameter limit exceedances not flagged by the TEMS.

The CEMS IV host is a Motorola 68020, System V Unix™-based super minicomputer. The expert system, resident on the standard Air Force

The CEMS IV host is a Motorola 68020, System V Unix™-based super minicomputer. The expert system, resident on the standard Air Force microcomputer, accesses the CEMS IV database via a direct RS-232 link or a leased line modem. XMAN extracts facts from the CEMS database. These facts are transferred from the host system to the microcomputer and stored in a troubleshooting fact file. Troubleshooting facts and technician inputs are used by XMAN to diagnose engine discrepancies. Throughout the troubleshooting session, the technician may access the CEMS IV database directly and recall pertinent diagnostics data. The XMAN control system links the engine discrepancies present to the appropriate troubleshooting rule file (i.e., knowledge base). When troubleshooting is completed, XMAN prompts the user with corrective maintenance procedures. [2]

Troubleshooting a Rising Wearmetal

A typical equipment discrepancy summary produced by XMAN upon analysis of the CEMS IV database is shown in Figure 4. Shown in this summary are the following:

a. a reference number for each discrepancy;
b. the equipment serial number (ESN);
c. the aircraft location;
d. the equipment discrepancy; and
e. the date of the discrepancy.

In this example, an engine alarm generated by a rising oil wearmetal trend is analyzed. CEMS IV has forecast that the iron concentration in engine GE205293 is due to exceed the allowable limit.

```
E Q U I P M E N T   D I S C R E P A N C Y   S U M M A R Y

REF ESN    LOCATION          DISCREPANCY                    DATE
--- -------- --------- ------------------------------ -------
1 GE205012 A770211-1 NFTR FORCASTD BELOW LIMIT (CEMS) 87JUN01
2 GE205137 A750262-1 LEVEL 1 TEMS:NF VIBS - LEV 1     87MAY29
3 GE205293 A750300-2 FE   FORCASTD ABOVE LIMIT (CEMS) 87MAY29
4 GE205317 A760550-1 LEVEL 1 TEMS:SLOW START          87MAY22
5 GE206512 A750264-2 FFGT (FFG .TND) ABOVE LIM (CEMS) 87JUN01

Please select a REF number )
```

Figure 4 The Equipment Discrepancy Summary Indicates Alarms Which Require Troubleshooting.

The technician initiates the XMAN troubleshooting session by entering the equipment discrepancy reference number (in this case, 3). Once selected, XMAN activates the appropriate fact file and decision tree (i.e., rule file) corresponding to this engine problem.

An XMAN troubleshooting display is shown in Figure 5. In the middle window of this display, XMAN asserts facts derived from the CEMS IV database. Troubleshooting questions are posed along with answers automatically derived from the CEMS IV historical database and the user's affirmation or rejection.

```
GE205293          EXPERT TROUBLESHOOTER          A750300-2
ALARM:
        ED/CEMS DETECTED MALFUNCTION - C001          MODE: STEP
    IRON (FE) CONCENTRATION FORECASTED OVER LIMIT

ASSERTIONS:
1. Is it possible that FE DATA SCATTER caused the alarm ?     XMAN: NO
                                                              USER: NO

2. Was maintenance (for example, an oil change) done prior    XMAN: NO
to this alarm and is the FE concentration RETURNING to        USER: NO
the level found BEFORE the MAINTENANCE ?

3. Are CORE VIBRATION levels INCREASING in any of the         XMAN: YES
channels?                                                     USER: YES

ACTION:
      Borescope the engine compressor and high pressure turbine
      (HPT) according to the procedures contained in T.O.
      1A-10A-2-71MS-1. Return to XMAN upon completion.

)Press RETURN to continue.                          87JUN03(1746)
```

Figure 5 The Technician Interacts with XMAN Through the Expert Troubleshooter Display.

In this example, the XMAN fact generator has asserted that erratic data did not cause the alarm. Following the XMAN assertion, the technician is asked to accept or reject this automated analysis of CEMS IV data. Since the technician in this example is relatively new to automated engine diagnostics, he asks for help in interpreting data scatter. The user presses <HELP> on his keyboard. XMAN responds with the graphical display shown in Figure 6. Typically, several levels of HELP complexity are available to the maintenance technician. The complexity ranges from high-level descriptions to graphical displays highlighted by inverse video and pointing arrows. By pressing the <RETURN> key, the user cycles through the high-level to low-level HELP displays.

In Figure 6, a typical wearmetal pattern is displayed, and the text explains what to look for in distinguishing real data trends from erratic data. The technician presses a special function key to exit from the HELP utility and resume XMAN troubleshooting.

Fresh with new insight into data scatter, the user presses the <HOME> key to access the actual CEMS IV data. The <HOME> key is dedicated to spawning CEMS IV from the LISP environment. In this instance, the rising trend is clearly distinguished from data scatter (see Figure 7).

After returning to the troublesnooting session, the user concurs that iron data scatter did not trigger the alarm. XMAN proceeds with the next assertion,

i.e., that corrective maintenance has not had a positive effect on the iron trend. Again, the technician may ask for help in interpreting the data before responding to this question (HELP), and he may analyze the CEMS IV data directly (HOME). In this case, no significant maintenance action (e.g., an oil change) was performed on this engine recently.

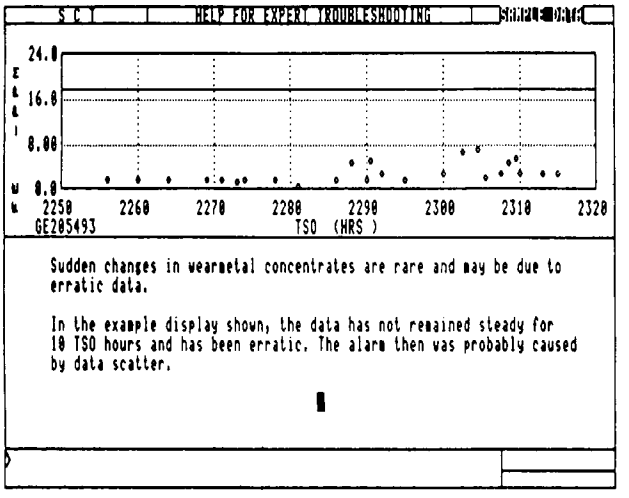


Figure 6 The XMAN Help Facility Guides the Technician Through the Diagnostics Process.

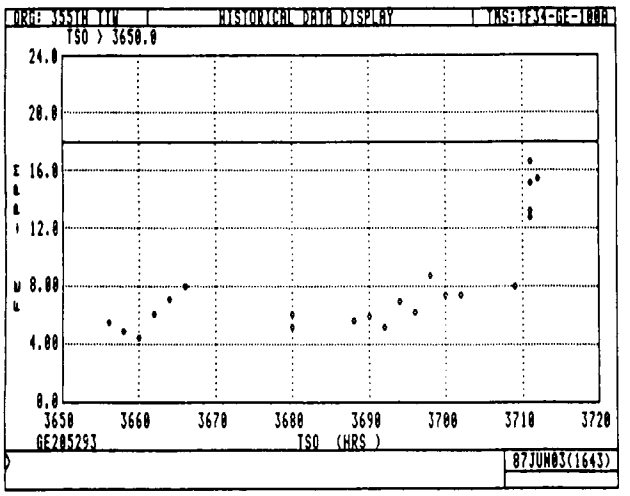


Figure 7 The User May View Pertinent Diagnostics Data by Pressing the <HOME> Key.

Next, XMAN asserts that core vibration levels are rising on the engine. Examination of the data indicates that front frame and exhaust frame vibration readings at core frequency are increasing slightly (see Figure 8). Based upon the analysis to this point, XMAN recommends that the technician schedule the engine for a borescope inspection. The user is instructed to return to XMAN upon completion of this task.

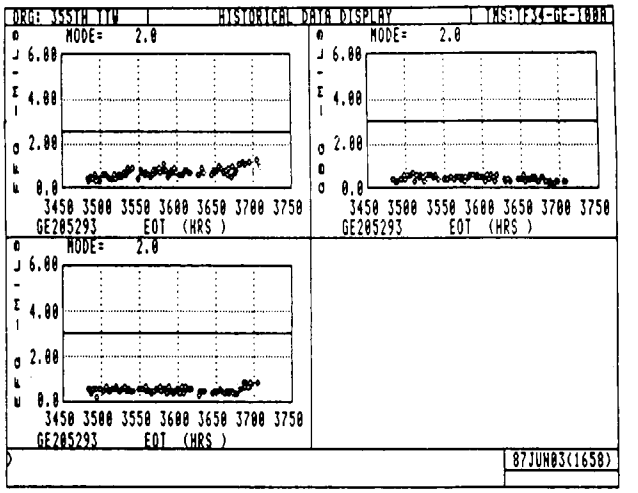


Figure 8 Core Frequency Vibration Levels Show Slightly Rising Trends in the Example.

After completion of the borescope inspection, the XMAN troubleshooting session is resumed, this time in the automatic processing mode. XMAN automatically scans down to the last assertion previously processed. The user is asked if the borescope revealed significant turbine or compressor damage, and the response in this case is negative. Figure 9 shows the recommended XMAN action for this engine discrepancy. The engine is to be placed on the CEMS IV WATCH list in order to keep the engine under surveillance. The WATCH list maintains a record of engines which require special attention or follow-up service. CEMS IV advises the user when a review or action is due. In order to facilitate access to the WATCH list, XMAN issues the CEMS IV WATCH command line when the user presses the <HOME> key. Further analysis is necessary if the wearmetal reading continues on its trend or is correlated with serious vibration increases.

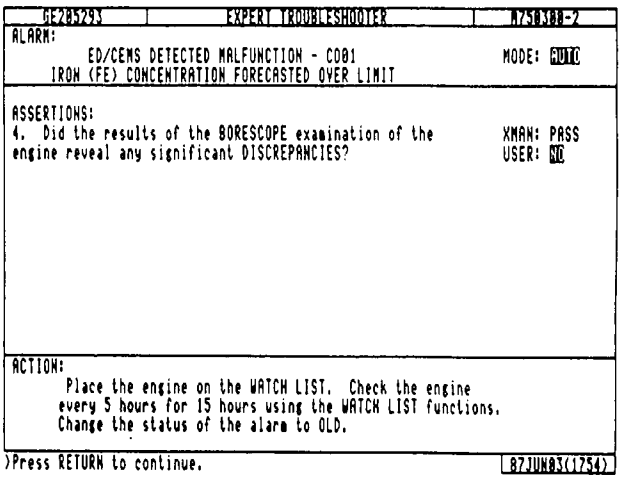


Figure 9 XMAN Recommends a Maintenance or Follow-Up Action Upon Completion.

At several points in the scenario described, the user may have responded "PASS" rather than entering a yes or no response. This response is programmatically equivalent to acceptance of the XMAN assertion. However, the audit trail or journal of the troubleshooting activity shows that the user entered an "I Don't Know" response.

OPERATIONAL EVALUATION

An audit trail of all engine troubleshooting performed allows visibility into actual discrepancies processed in the field. At each base, XMAN tracks all engine alarms analyzed locally including user responses and recommended maintenance actions. The entire XMAN session may be recreated using the troubleshooting results which are captured in an archived fact file. This archived file includes the engine discrepancy, time, date, and user stamps, facts generated, and user responses.

Archived engine fact files are received from the bases periodically. Results are assimilated from all bases in a central analysis location, and troubleshooting effectiveness summaries are produced. Although not yet formally implemented, this detailed usage information may be analyzed by the equipment manager (e.g., repair depot engine managers) to evaluate diagnostics procedure effectiveness on their fielded weapon systems. In addition to high-level summaries of recommended actions versus equipment alarms, lower level detail of user interaction with the expert system and CEMS IV are clearly traceable.

For the first time, the equipment manager has actual field level troubleshooting information available to allow improvement in maintenance diagnostics technical orders. Further closure of the maintenance diagnostics loop is possible through correlation of XMAN recommended actions with actual maintenance performed (i.e., AFTO 349 information available through the CEMS IV/CAMS (Core Automated Maintenance System) interface. This correlation is currently under investigation.

Troubleshooting Summary

XMAN was installed at Barksdale Air Force Base, Louisiana, for user evaluation and operational testing in October 1986. A six-month evaluation period preceded the release of XMAN to seven other Air Force bases. These bases included Davis Monthan AFB, Myrtle Beach AFB, England AFB, Glenn L. Martin Airport (Maryland Air National Guard), Nellis AFB, Sacramento Air Logistics Center, and San Antonio Air Logistics Center. XMAN was released for operational use in late April 1987. The evaluation results presented focus on the user evaluation operations at Barksdale AFB.

During eight months of use at the 917th TFG, XMAN aided processing of over 700 engine discrepancies. A summary of the most frequently occurring CEMS IV and TEMS alarms is shown in Figure 10. The three most frequently processed CEMS IV alarms include:

a. Fan speed trim margin forecasted below limit;
b. A below limit trend in corrected fan speed versus corrected temperature; and
c. Chromium forecasted over limit (an oil wearmetal abnormality).

Note that a large percentage of the alarms diagnosed did not require immediate maintenance attention. XMAN recommended alarm deletion in these instances. This insight into alarms inappropriately generated allows for the possibility of fine tuning of the CEMS and TEMS software to reduce this undesired characteristic. Other recommended actions based upon XMAN analysis of CEMS IV generated alarms focused on the scheduling of engine water washes (a problem typical of the A-10/TF-34 due to gun gas ingestion) and engine placement on the CEMS IV surveillance list (WATCH).

The most frequently analyzed TEMS engine discrepancies included shifts in the interstage turbine temperature and core flameout. Recommended actions concentrated on checkout of the monitoring system rather than actual engine repair. The results analyzed are indicative of the overall excellent health of the TF-34 engines at Barksdale which have been supported by TEMS and CEMS IV for over five years.

ALARM \ ACTION	DELETE	WATCH	TROUBLESHOOT OTHER ALARM	SCHEDULE WASH	REPLACE T5 AMP	REPAIR TEMS
NFTR FORECASTED BELOW LIMIT	234	19	5	17		
NFT5 TREND BELOW LIMIT	37	5	17			
CR FORECASTED OVER LIMIT	9	50				
LEVEL 1 TEMS:T5 SHIFT	18	10			2	
LEVEL 2 TEMS:NG FLAMEOUT	4					1

Figure 10 The XMAN Audit Trail Allows Insight into Actual Troubleshooting Performed in the Field.

CONCLUSIONS

Integrated diagnostics, performance reliability, and equipment maintainability are taking on increasingly crucial roles in Department of Defense weapon system support strategies. New methods for providing critical maintenance and logistics information in a timely manner and with limited user interaction are essential. Computer programs which embody forms of human expert problem solving abilities offer significant supportability benefits in an era of increasing equipment sophistication and decreasing service personnel availability.

XMAN offers enhanced diagnostics and interactive training in a commercially available

software package. As the services gear equipment diagnostics programs toward portable maintenance aids (e.g., IMIS (Integrated Maintenance Information System)), XMAN offers a proven operational maintenance diagnostics troubleshooting capability. As an integrated diagnostics tool, this expert system allows insight into actual troubleshooting performed, evaluation of results, and the feedback loop to improve diagnostics procedures.

REFERENCES

1. De Hoff, R.L. and L.J. Dolny, "Maintenance Information Management System (MIMS™) - Strategic Maintenance Decision Support", Maintenance Management International, Vol. 5 (1985), pp. 31-40.

2. De Hoff, R.L., Dolny, L.J., Jellison, T.G. and N.S. Pratt, "XMAN™ - An Expert Maintenance Tool", AUTOTESTCON '86 Proceedings, Sept. 1986, San Antonio, TX.

Development of a Comprehensive Software Engineering Environment

Thomas C. Hartrum
Department of Electrical and Computer Engineering
School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Dayton, Ohio, 45433

Gary B. Lamont
Department of Electrical and Computer Engineering
School of Engineering
Air Force Institute of Technology
Wright-Patterson AFB, Dayton, Ohio, 45433

Abstract

The generation of a set of tools for the software lifecycle is a recurring theme in the software engineering literature. The development of such tools and their integration into a software development environment is a difficult task at best because of the magnitude (number of variables) and the complexity (combinatorics) of the software lifecycle process. An initial development of a global approach was initiated at AFIT in 1982 as the Software Development Workbench (SDW). Also other restricted environments have evolved emphasizing Ada and distributed processing. Continuing efforts focus on tool development, tool integration, human interfacing (graphics; SADT, DFD, structure charts, ...), data dictionaries, and testing algorithms. Current efforts are emphasizing natural language interfaces, expert system software development associates and distributed environments with Ada as the target language. The current implementation of the SDW is on a VAX-11/780 under VMS. Also, a simplified version of the SDW has been hosted on personal computers. Other software development tools are hosted under UNIX and are being networked through engineering work stations. This paper discusses the various aspects of AFIT's development of software engineering environments.

Introduction

A software development environment is an integrated set of automated and interactive software development tools that aid the software engineer in the development of quality software products. The specific software products which are associated with the software life cycle include requirements definitions; design specifications; source and executable program codes; test plans, procedures, and results; as well as other associated documentation such as guides and manuals of operation and maintenance of the software. By definition, software only exists in its documentation! Thus, extensive records must be generated, maintained, and managed to properly fulfill the software engineering objectives. A well planned and implemented software development environment can effectively assist in the generation of reliable and maintainable computer software.

The typical software development environment includes both hardware and software tools to aid the software engineer in the production of programs. Software development environments may consist of a minimal set of tools, such as an editor, a compiler, and a link/loader, that support only the actual coding of software. However, the most effective environments are those with an extensive set of powerful interactive and integrated tools that support state-of-the-art methodologies for dealing with software from its very conception through its eventual termination. A specific software development environment consists of a process methodology along with given hardware and system software, manual procedures and support personnel. The process methodology usually involves a specific set of operations (steps) along with conceptual tools to support these steps within the software lifecycle phases mentioned previously.

The concept of an integrated software development environment can be realized in two distinct levels. The first level deals with the access and usage mechanisms for the interactive tools, while the second level concerns the preservation of software development data and the relationships between the products of the different software life cycle stages. The first level requires that all of the component tools be resident under one operating system and be accessible through a common user interface. The second level dictates the need to store development data (requirements specifications, designs, code, test plans and procedures, manuals, etc.) in an integrated data base that preserves the relationships between the products of the different life cycle stages. This integration of tools and techniques at both levels is a major objective of any software engineering environment development effort.

The major objectives are to provide a production software development environment for students and faculty and to generate a software engineering research testbed. Initially the SDW provided the overall architecture for a "complete" capability. Recently, efforts are focusing on a distributed version of the SDW concept called System 690 in support of the software engineering laboratory course, EENG 690.

Development Lifecycle Model

The definition of the software lifecycle as supported and used by the various environments consists of the standard six phases; requirements definition, preliminary design, detailed design, implementation (coding), integration and operation and maintenance. This general methodology is reflected in DOD Standard 2167 [1]. Documentation must be provided within each phase to support reviews (static) and testing (dynamic) of results associated with each activity. This capability can be provided through the use of a data dictionary and associated data base management system. Software system corrections and enhancements should flow through all previous phases for "proper" documentation.

Note that validation and testing is not a distinct stage in this lifecycle, but rather an activity that is performed along the entire lifecycle. This activity involves the testing of the products of each stage for internal consistency and completeness with the products of the previous stages. Furthermore, the products of each stage are validated against the user's perception of the requirements.

The Software Development Workbench(SDW) and the distributed environment, System 690, are developed using this software life cycle definition with the primary objective of providing integrated and automated support. Discussion of each environment follows the stages of its initial lifecycle. The objectives and accomplishments of each stage of the lifecycle development are presented. The requirements definition and preliminary design stages deal with a system as it should exist in its ultimate form, whereas the detailed design, implementation, integration, and operation stages emphasize a prototype environment with a menu driven interface and initial tool set.

SDW Requirements and Design

The first stages of the SDW development effort [2] emphasized the requirements definition and preliminary design of the ultimate SDW implementation. Due to the extensive scope of this task, the target was set at a fairly high level specification with the individual subsystems specified in greater detail with the use of recursive applications of the software life cycle.

The results of this task are a set of five primary objectives of the SDW, thirteen specific concerns for its development [3], a functional model and associated evaluation criteria, a hardware/software configuration model, and a structure model that identifies all generic component tool types.

Of the five objectives of the SDW, the reduction of software errors is the first [4]. This is to be achieved by supporting and enforcing the use of accepted software engineering principles, as well as by using the computer to augment different testing procedures.

The SDW must also be responsive to change. Realizing that software is a dynamic entity, the SDW must be able to support changing requirements for its operation.

The rapid assessment of design alternatives is also quite important. The use of simulation models and prototyping is selected more and more to assess design operations as well as to aid in determining the end user's true needs.

The SDW must also be capable of providing interactive and automated documentation support. This support must emphasize the recording, and maintenance of all software development associated data.

Finally, the SDW must provide mechanisms to assist the software manager in planning and tracking software development efforts.

The thirteen specific concerns also required to be addressed by the SDW development effort are: integration, traceability, user-friendliness, testability, pre-fabricated programming, support for the entire software lifecycle, flexibility, consistency and completeness, explicitness and understandability, documentation support, updateability, language independence, and early prototyping. The first five of these concerns are of special significance to the SDW effort. Integration is to be realized in terms of both accessing component tools and storing of the development data. Traceability must also be preserved between the products of the different stages of the development effort. User-friendliness is also a very significant concern. The SDW must utilize the latest concepts of ergonomics in the design of its human interface. This interface should be easily understandable with a simple logical structure, well laid out display, and a simple command input mechanism. Prefabricated programming, or the incorporation of existing software can improve development productivity. Flexibility is required at both the environment and tool level to allow users to operate in a mode comfortable to their knowledge and experience levels. That is, the operation of the SDW must allow the user to tailor the type of prompting, feedback, and structure [5].

A functional model of the software development process using SADT (Structured Analysis and Design Technique) diagrams [6] was developed in order to define the SDW process and to select those aspects of the process that could be automated. Furthermore, a set of evaluation criteria is established with which to judge the effectiveness of the environment in satisfying its requirements. However, for reasons of brevity, these topics are not discussed further.

The configuration model of the SDW shown in Figure 1 illustrates the basic hardware/software configuration for the environment. The SDW Executive is the primary interface and controller of the component tools. The SDW tool set is broken down into three tool categories; cognitive tools, that extend the powers of understanding for the software developer; notational tools, that assist in the production and maintenance of associated documentation; and augmentive tools, that use the powers of the computer to perform much of the tedious testing and updating activities involved with software development. The project data bases are the integrated data storage areas with one allocated to each development effort. Finally, the Pre-Fab Software Description and Product Data Bases hold the functional descriptions and program codes of existing software modules. This structure provides for easy retrieving and incorporation of modules into development designs and implementations.

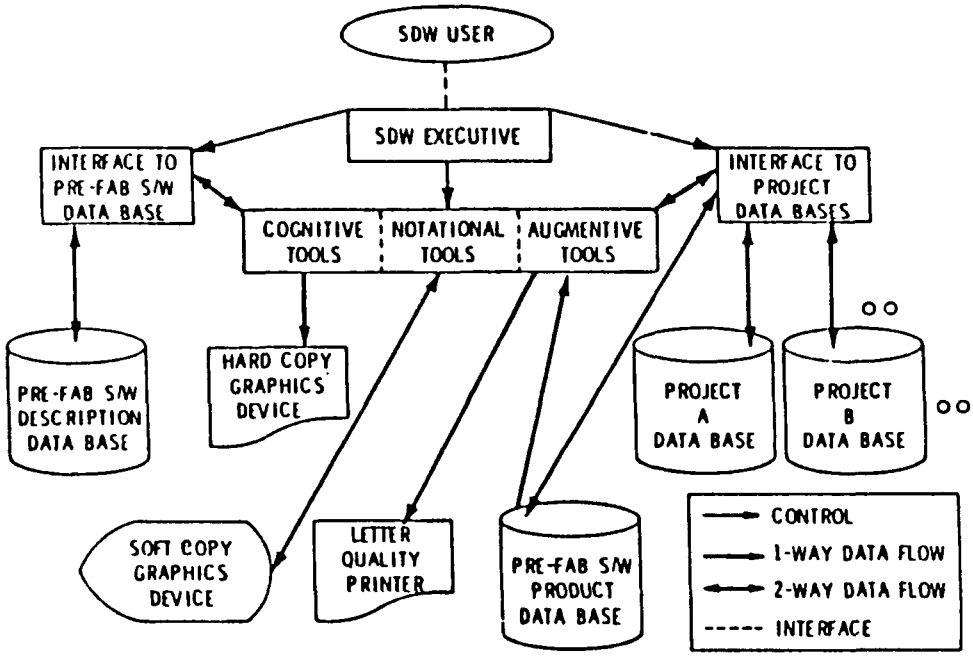


Figure 1: SDW Configuration Model.

The structure model of Figure 2 illustrates the generic tool types that are to be incorporated into the SDW. Those tool types annotated with a single asterisk are included in the initial implementation of the SDW, while those with two asterisks are scheduled for the second level of implementation. Those with three asterisks are to be included as they are developed or become available. Thus, the frameworks for the initial and eventual realizations of the SDW are established. With this background, a detailed design and implementation can be realized that includes the selection of existing tools as components and complete development of other components.

Current Implementation of the SDW

The detailed design, implementation, and integration stages of the SDW development effort focus on the accomplishment of an initial version of the environment. This initial version is composed of software development tools that support the pre-implementation activities of software development as well as provide the common capabilities found in most implementation oriented development environments such as editing, linking, and debugging.

The discussion of this initial version of the SDW is limited to two topics: the selection of an initial tool set and the complete development of the SDW Executive (SDWE) component that provides the common access and control mechanisms required to satisfy the first level of the integration criteria.

The tools selected for inclusion into the initial implementation of the SDW are taken, for the most part, from one of two sources. Those tools that specifically support the requirements specification and design activities were given by the Integrated Computer-Aided Manufacturing Division of the Air Force's Material Laboratory, Wright-Patterson Air Force Base. The tools used to provide the rest of the development support are the standard vendor supplied tools normally found on the target computer (the Digital Equipment corporation's VAX-11/780 under the VMS operating system).

Four distinct tools are selected to support the first two phases of software development. They are the AUTOIDEF [7], that supports the Integrated Computer-Aided Manufacturing (ICAM) Definition Techniques (IDEF) [8], the SYSFLOW graphics editor [9], and the Extended Requirements Engineering and Validation System (EREVS) [10].

The AUTOIDEF tools support and aid in the production and maintenance of three types of IDEF models. IDEF-0 models are used to provide a functional modelling capability which describes the flow of data through functional processes. IDEF-1 models provide an informational modelling technique that describes both the components of a data entity and the relationships between data entities. Finally, IDEF-2 models are used for dynamic modelling to simulate transaction flows through network-like systems. The AUTOIDEF tool greatly simplifies the production and maintenance of all three types of models because of its flexible graphics drawing and modification capabilities.

The SYSFLOW graphics editor is an easy to use and flexible tool. The tools provide a basic set of graphical constructs and character fonts, together with the capability for the user to define his own constructs, to provide a very flexible capability to produce and maintain a great variety of graphical/textual documentation. This system can be employed in generating data flow diagrams defining detailed requirements or it can be used to define structure diagrams as generated by transform analysis or transaction analysis [11] of the requirements in a data flow or SADT format.

The ICAM Decision Support System (IDSS) provides for the graphical and textual input of IDEF-2 dynamics models. The results of executing these simulation runs are analytical reports on the simulated system's performance. The provisions for graphical input of models and automatic translation into an executable format make the tool a truly state-of-the-art facility.

The Extended Requirements Engineering and Validation System (EREVS) was originally developed by TRW, Inc. for the Army's Ballistic Missile Advanced Technology Center. EREVS provides sophisticated facilities for specifying system requirements for concurrent and real time systems, checking those requirements for consistency and completeness, illustrating the requirements with a graphical technique

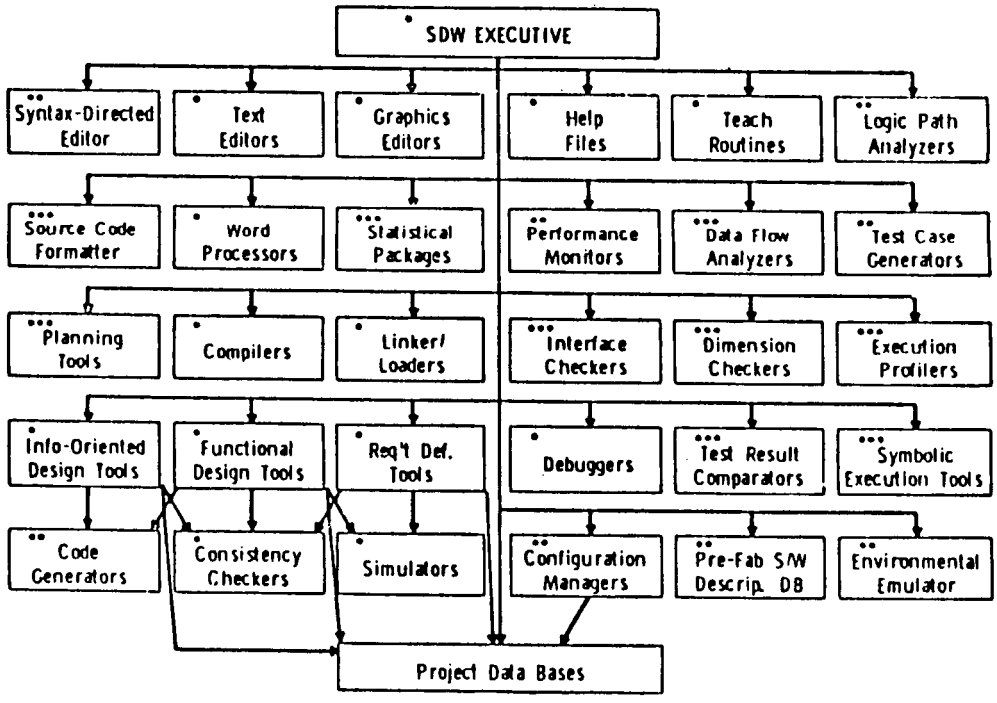


Figure 2: SDW Structure Model.

called R-nets, and then simulating the timing feasibility of the stated requirements. Although specifically designed for concurrent and real time software systems, EREVS is an effective aid in developing well stated and feasible requirements for all types of software systems.

In addition to these tools, the SDW uses the standard VMS available tools to perform the required compiler, linker, editor, debugger, comparator, and text processing functions. Moreover, DEC's Program Development Tools consisting of programming tools and project management tools can easily be integrated into the SDW. The programming tools include the language-sensitive editor (syntax-directed), the source code analyzer, the symbolic debugger, and the performance and coverage analyzer (PCA). The project management tools include the code management system(CMS), the module management system(MMS), the test manager and the common data dictionary(CDD). Other DEC software products supporting software development include their data-base management systems (DBMS, Rdb/VMS, Datatrive), forms management system(FMS), and the application control and management system(ACMS).

With this set of components established, the SDW Executive is developed with common interfaces to specific tools. The SDWE is a menu and command driven interface to all of the SDW component tools but also provides access to all of the facilities provided by the VMS operating system.

In order to structure the accessing of the SDW component tools, these tools are assembled into groups by related functions. The top level menu of the SDW allows for the selection of any of these functional groups. Once a functional group is specified, all of the member tools of that group can be accessed through a new menu.

At any level in the hierarchy formed by these menus, any of the standard VAX DCL commands may be executed. Furthermore, multiple levels of the menu hierarchy may be traversed at any time by simply entering the appropriate command string on a single command line.

On-line help facilities are provided for all levels of the hierarchy. These help facilities provide either general help on the environment, specific help information on any of the currently accessible commands, and access to the VMS standard help facility. Additionally, utility functions are provided to enable/disable the automatic displaying of current menus, to change the type of terminal in use, and to alter the manner in which the development data is stored.

Specific SDW component tools may either by accessed through the menu structure and command or by using special commands to execute the tool directly, thus increasing the flexibility of the environment.

Development data storage is provided by this implementation by establishing isolated data storage areas for each supported development effort. However, these data storage areas do not at present provide for the full integration of the data that is defined by the preservation of the relationships between the different development products.

The SDW Executive was also designed to be easily modifiable. Thus, new tools may be easily incorporated into the environment. Furthermore, a full set of documentation is provided on the SDWE. This documentation includes a user manual, an installation guide, and a maintenance guide that is to be used to modify and tune the environment for specific applications. The SDW is currently installed on the AFIT research VAX-11/780. Users have found the environment to be very easy to learn and use.

Expert Systems and Software Engineering

The integration of artificial intelligence concepts into software engineering environments currently focuses on expert systems. Specific expert subsystems must be developed for each phase of the software lifecycle to assist in design development and selection, structural formulation, algorithm determination, structured programming implementation (object-oriented, abstract data types, control structure), module and system testing and maintenance. An initial effort towards defining an associated environment resulted in a modification of the SDW executive using OPS5 for expert system inclusion [12]. Also, this initial design focused on the analysis and diagnosis of modules in terms of coupling and cohesion standards.

Another aspect of AI integration into software engineering environments was the development of a natural language interface [13]. This activity generated a natural language interface called "COIN" which uses Lisp and the Flavors package. This initial effort emphasized the interface to the data dictionary (DD) package mentioned previously since the perspective user would have a considerable dialog with the DD in defining detailed entries and preforming queries.

Additional efforts involve the use and analysis of transformational systems that can encompass knowledge-based capabilities for software production. Example efforts include the Knowledge-based Software Assistant (KBSA) and REFINE, a wide spectrum language for the development phases of the software lifecycle. Incorporation of wide-spectrum languages into a software environment may be feasible and economical which could be part of environment enhancements.

SDW Enhancements

The first enhancement is to extend and refine the SDW tool set to provide a full array of capabilities to support the entire software life cycle. This tool set must also be refined so that only those tools that are truly effective and useful remain part of the environment. Also, a user should be able to specify that only a certain sequence of tools be used in a given project and the SDW would provide only that environment, such as for Ada real-time applications.

The Pre-Fab Software Description and Product Data Bases must be completely developed and populated to support the prefabricated programming concept. After the establishment of a fairly static tool set, a schema for the Project Data Bases can be developed. These data bases will hold all of the development data for the products of each software development effort as well as the relationships between the different products of each effort. An initial project in this regard generated a prototype data dictionary [14] for the SDW using the DBMS Ingress. This effort was further enhanced with the System 690 project under the UNIX operating system.

The scope of the support provided by the SDW is also to be expanded to aid the software development manager in planning and tracking the development effort. Responses to queries on the Project Data Bases will provide the software development manager with near real time feedback on the status of the development effort.

The current implementation of the SDW is quite flexible and an easy to use aid for the development of quality software products. This initial implementation provides extensive support for the pre-implementation stages of software development. The environment effectively increases the cognitive and notational powers of the software developer.

The ultimate implementation of the environment will support the entire software development life cycle. Much of the tedious consistency and completeness testing of software will be automated in this environment. Furthermore, provisions will be included to store and maintain all development data in a fashion that preserves traceability between the products of the different life cycle stages. Such an environment would be a significant breakthrough in the production and maintenance of quality software systems.

Distributed SE Environment

Using some of the SDW concepts, a distributed software development environment called SYSTEM 690 is being developed to support classroom and research programming projects as well as research into environment issues. SYSTEM 690 addresses the same objectives of the SDW but in a distributed environment. The computer environment used by SYSTEM 690 is both heterogeneous and quite extensive. Most of the software development is done on a network of VAXes and Sun workstations running Unix and interconnected by a TCP/IP Ethernet, and on a series of DEC VAXes and MicroVAXes interconnected by DECNET. The two networks are interconnected by a gateway. All

of these systems are also accessible via a Gandalf RS-232 switch that is connected to a variety of terminals and PCs in offices and labs, and through dial-up lines to any number of home computers. Software development is performed on these computers under a variety of operating systems using Ada, C, Pascal, Lisp, Prolog, Fortran, and assembly language.

In SYSTEM 690, specific emphasis is placed on performance monitoring and analysis to provide needed data in such areas as tool performance, tool usage, user acceptance, and the nature of the workload, both in terms of the size of data and frequency of use of the tools.

Methodology

In order to put a production software system in operation and to develop tools to support that system, the methodology selected was that mentioned in the SDW discussion, namely the use of DOD Standard 2167 and the ICAM program structure. Again, this methodology was selected with the goal of supporting the automation of the software development process, and is centered around the comprehensive data dictionary system that documents all aspects of the lifecycle as discussed previously. Each phase of the lifecycle requires its own data dictionary entities, an action entity and a data entity. Figure 3 shows an example of the information contained in these entries for the design phase. A central concept is that the data dictionary provides the complete definition of the entire development. In each of the three major phases, however, some form of graphical representation provides a more human understandable means of generating and viewing the data dictionary information. Thus the IDEF model was chosen as developed under the ICAM program. Figure 4 shows a typical analysis diagram. The underlying abstract data is stored as two types of data dictionary entries: one for each *activity* (each box on the diagrams) and one for each *data element* (each arrow on the diagrams). Information relating to the graphical layout of the structured analysis diagram is not considered part of the requirements analysis information, and is not included in the data dictionary.

For the design phase, the primary graphical representation is a structure chart. This is also documented by two types of data dictionary entries: one for each *process* (each box on the diagrams) and one for each *parameter*, as shown in Figure 3.

The design process used with SYSTEM 690 uses transform analysis and transaction analysis to evolve the requirements specification into a modular design. Detailed design is accomplished by using PDL in the algorithm section of the process data dictionary entry. Currently this is a free form psuedo-code, but in the future will be an Ada based PDL. Note that an Ada software engineering environment called ARCADE is being developed with the SDW and SYSTEM 690 efforts.

Similar to the SDW, the primary graphical representation in the implementation phase is the structure chart, representing the structural relationship between the actual code modules and showing the actual passed variables. This is also documented by two types of data dictionary entries: one for each *module* (each actual code module, subroutine, or function) and one for each passed *variable*. Of course, in this phase there is another representation of the effort, that of the code itself. The implementation process used is top-down coding, with integrated testing.

Example Data Dictionary Entry for Process

```
NAME: Process Message
PROJECT: NETOS-ISO
TYPE: PROCESS
NUMBER: 4.0.1
DESCRIPTION: Processes a NETOS message.
INPUT DATA: msgptr
INPUT FLAGS: None
OUTPUT DATA: None
OUTPUT FLAGS: error2
ALIAS: PROC-MSG
  COMMENT: Used in earlier design.
CALLING PROCESSES: Process Messages and Data
PROCESSES CALLED: Decompose Message
   Process Network 4 Messages
   Determine Channel Number
   Build Queue Buffer for Qty  = 1
   Put Buffer in Queue
   Level 4 Cleanup
ALGORITHM:
  Decompose message.
  If network message
     Process Network 4 Messages
  else
     Determine channel number
     Build queue buffer
     Put buffer in Queue
  Cleanup Level 4.
REFERENCE: PROCESS SPOOLER MESSAGE
  REFERENCE TYPE: SADT
REFERENCE: Smith's Algorithm's, pp. 23-24.
  REFERENCE TYPE: Text.
VERSION: 1.1
VERSION CHANGES: Added "Level 4 Cleanup"
DATE: 11/25/85
AUTHOR: T. C. Hartrum
```

Example Data Dictionary Entry for Parameter

```
NAME: mess-parts
PROJECT: NETOS-ISO
TYPE: PARAMETER
DESCRIPTION: Decomposed message parameters.
DATA TYPE: Composite, C structure .
MIN VALUE: None
MAX VALUE: None
RANGE OF VALUES: None
VALUES: None
PART OF: None
COMPOSITION: SRC
     DST
     SPN
     DPN
     USE
     QTY
     Buffer
ALIAS: Message Parts
  WHERE USED: Decompose Message to Validate Parts.
  COMMENT: Part of earlier design
ALIAS: messy-parts
  WHERE USED: Passed from Dump Data to Flush Buffer.
  COMMENT: Part of existing library.
REFERENCE: MSG-PARTS
  REFERENCE TYPE: SADT
VERSION: 1.2
VERSION CHANGES: Component USE added
DATE: 11/05/85
AUTHOR: T. C. Hartrum
CALLING PROCESS: Process Message
  PROCESS CALLED: Decompose Message(parts-list)
  DIRECTION: up
  I/O PARAMETER NAME: parts-list
CALLING PROCESS: Process Message
  PROCESS CALLED: Process Network 4 Messages
  DIRECTION: down
  I/O PARAMETER NAME: parts
CALLING PROCESS: Process Message
  PROCESS CALLED: Build Queue Buffer for QTY  = 1
  DIRECTION: down
  I/O PARAMETER NAME: params
```

Figure 3: Design Phase Data Dictionary Example.

A1 Execute Remote Function

Abstract: This diagram decomposes the activity of the Remote Systems into its major functions.

NOTE: This analysis assumes that a user wishing to execute a function dealing with either the Spooler System or the MSS must first establish comms with that system with a specific comms request command. An alternate approach that might be a better user interface would be for each remote function (e.g. Get File) to establish that comms link transparent to the user.

A11 Select Remote Function is visualized as being implemented by the operating system's capability of loading and executing individual programs from disk. Thus the remaining activities are seen as seperatly executable programs (although a menu-called function structure could be used).

A12 Manipulate Files handles all remote requests to manipulate actual files. This includes printing local files, storing local files on the MSS, and requesting files from the MSS.

A13 Manipulate File Information handles remote activities involving information about files, but not the files themselves. This includes requests for MSS directory information, and updating and querying the catalog system.

A14 Manipulate Comms Links handles all remote requests to establish and terminate communication links to other remotes.

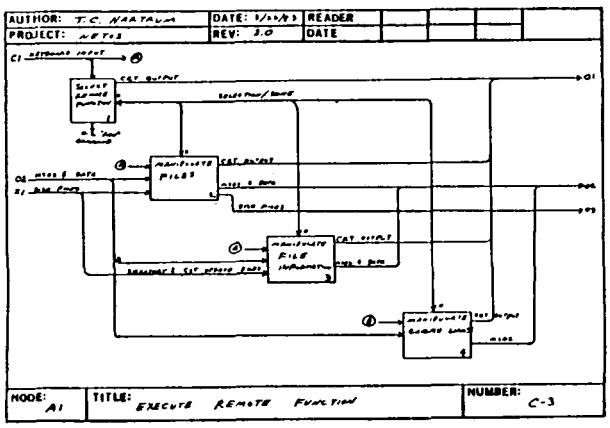


Figure 4: Example Structured Analysis Drawing.

SYSTEM 690 Approach

When considering an integrated environment, one can view integration from several perspectives. As shown in Figure 5, this includes integration of tools at the user-tool interface, integration between tools within any lifecycle phase, and integration across the entire lifecycle.

User level integration includes both consistency in interacting with the operating system (e.g. invoking tools via a menu) and consistency with interactive tool interfacing. This involves operating system specific issues as well as keyboard and display compatibility problems. It is planned to integrate the SDW environment with SYSTEM 690 to provide this level of integration, but this has not been done yet since in a heterogeneous environment a different version is required for each system.

Integration between tools is basically a question of compatibility of data, the ability of one tool to use the data generated by another tool. Compatibility itself can be viewed at a number of levels. In its most abstract form, compatibility of information is of concern. This is the biggest bar to integrating commercial tools from different vendors. Tools that use the same logical information may not have file format compatibility (a problem which frequently occurs when trying to integrate documentation from different word processors). Finally, two tools may not even have physical data format compatibility. Floppy disks written on one workstation may not be readable on a different one.

The tool-to-tool interface problem is being attacked at several levels. Compatibility of information is being controlled through the use of the data dictionary. File format problems are being handled through the use of a centralized database. The data dictionary definitions described earlier are decomposed into a set of third normal form relations which are maintained using the Ingres database management system. A data manager translates between the database and the file formats of specific tools. In order to minimize the amount of translation needed, a standard file format is used for all tools developed in-house.

Physical file format difficulties are avoided by using networking for all file transfers. This is available to any PC or workstation with a serial port using standard communication protocols, as well as between workstations and minicomputers via the Ethernet.

Integration over the lifecycle requires appropriate tools that are compatible with data used in two or more phases, and additional *mapping data* which relates items in the two phases. Currently there are no multi-phase tools in SYSTEM 690. It is anticipated that any such tools will be developed in-house, so that the mapping problem can be handled locally.

SYSTEM 690 Tools

Several classes of tools are in use or being developed for SYSTEM 690 which evolved in part from the SDW and other commercial tools. They include generic tools applied to the software engineering area, specialized graphical editors that allow creation or modification of data in a more graphical problem-oriented format, static analyzers that check various aspects of an existing design, and computer aided design (CAD) tools, including expert system and other AI techniques, that greatly assist the analyst or designer.

A number of old and new tools available for computers can be used to support the software development cycle. Some of the most useful are conventional text editors. Classically used for writing code and documentation, their big advantage is the universal availability and compatibility of text editors on all systems from micros to mainframes. By defining all standard file formats to contain only ASCII characters, a great amount of compatibility can be achieved in a heterogeneous environment. More sophisticated word processors are sometimes used to develop user's manuals, reports, and other such documentation. Here compatibility is maintained by defining a standard format (e.g. troff or TEX).

In terms of direct support for the SYSTEM 690 methodology, several tools have been developed to support data dictionary maintenance across all phases of the lifecycle. Although graphical tools to manipulate the data dictionary are being developed, they will be restricted to the more powerful graphics workstations. Therefore, we have developed a fill-in-the-blank forms editor for data dictionary entries that runs on a full range of computers [15]. The tool uses its own abbreviated ASCII files to store the data. Other data dictionary support tools include translators to convert between different file formats and the relational DBMS, and utilities for printing or viewing entries in the standard human-readable format [16].

To support the requirements analysis phase, an interactive structured analysis diagram editor is being developed on a SUN 3 workstation [17]. This tool makes it easy for an analyst to create and maintain such a diagram, while simultaneously updating the corresponding data dictionary entries. Similarly, an interactive structure chart editor is being developed to support the design and implementation phases.

Having all of the lifecycle data stored using a standard database manager makes it easy to develop static analyzers, tools that can examine the existing data dictionary information for consistency within a lifecycle phase and between phases. For the implementation phase there is also a style checker to analyze source code for adherence to local standards.

True computer-aided design tools are under development that will provide more than the ability to easily enter or examine design data. Several extensions are planned to the structured analysis diagram editor. These include the automatic placement of symbols and routing of lines, and software to help the analyst do the functional decomposition. An initial effort along these lines is a program that examines the data dictionary, and with interactive input from the designer analyzes the coupling and cohesion in a decomposition [12]. Similar extensions are planned to the structure chart editor. A planned extension to bridge the gap between the requirements analysis stage and the design stage will display a structured analysis diagram in

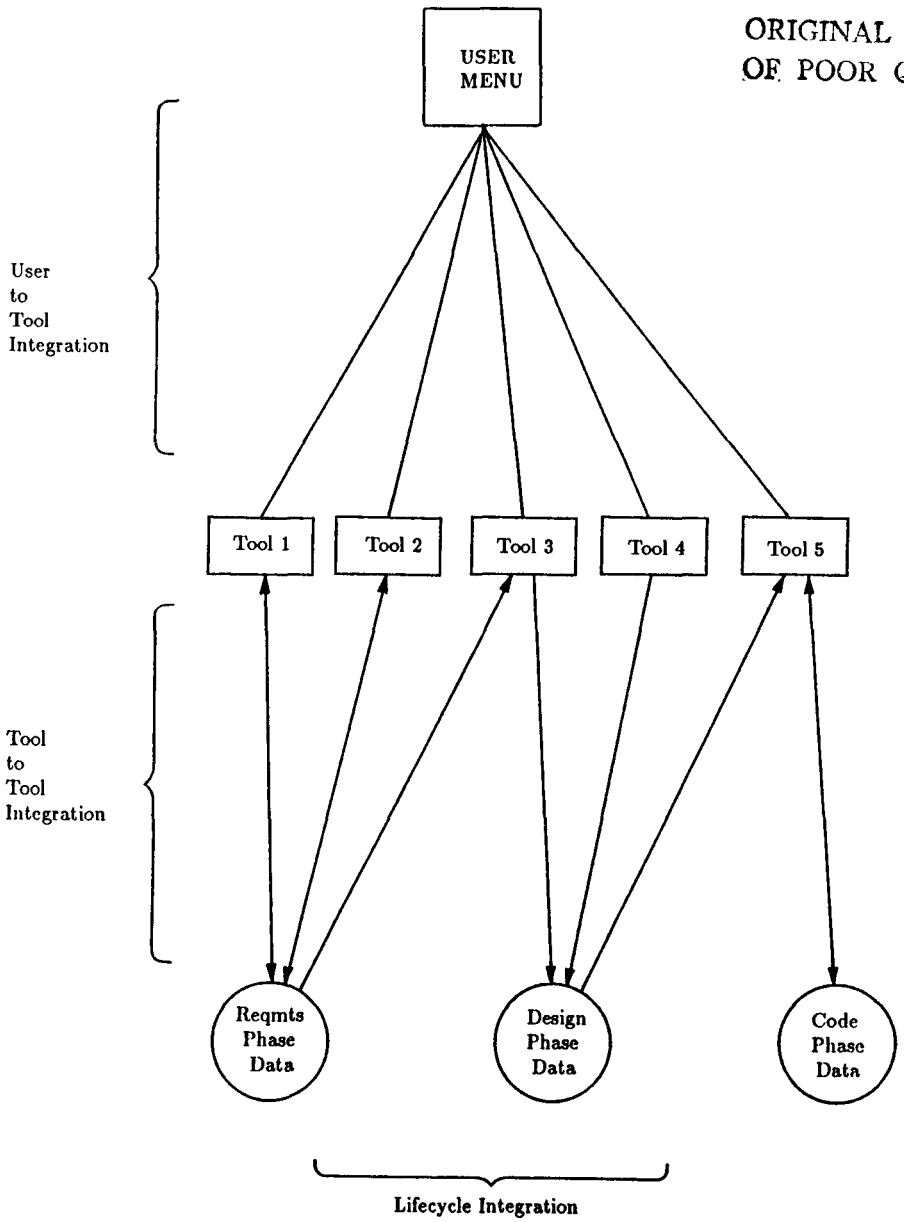


Figure 5: Types of Integration.

one window, and use transform and transaction analysis, along with expert systems techniques, to help the designer map directly into a structured design.

Testbed Considerations

A second objective of SYSTEM 690 is to provide a software engineering testbed to allow research into software engineering methodologies. The primary emphasis to date has been to develop ways of collecting data on the software engineering process. Tools are instrumented to collect usage and performance data to allow analysis of usage patterns [18] [15] [16]. The start and stop times of not only each tool, but of specific tool subfunctions, are recorded and stored in the Ingres database. A standard form for measuring user satisfaction with a given tool has been developed [19]. Standard statistical analysis packages are then used to analyze the data. A study is underway to determine what metrics are most needed to support the analysis of relative productivity for different software development methodologies.

Experiences and Plans

The first attempt at providing computer support was to create data dictionary entries in human-readable format using standard text editors and to store them in a common directory on a central VAX. Although this facilitated compatibility among the systems, the form of the files made it difficult to analyze or control data content with software tools.

The next version was a complete move to a centralized system. The data dictionary database was implemented under Ingres and an interactive editor was developed to run on the VAX that directly interfaced with the database. Although the database greatly simplified and encouraged the development of static analyzers and other tools, the load on the VAX from other applications slowed the editor response time to the point that users became frustrated. This experience with user dissatisfaction with response times made it clear that even PC level workstations are preferable for interactive tools. The development of the forms-based editor for the PC has been well accepted.

Most of our user experience has been in the design phase. The corresponding data dictionary has evolved with use and experience. We found several cases, mostly in the area of passed parameters, where what had been adequate for human understanding lacked the needed precision for machine readability. This required some augmentation of normal design techniques with rules of constraint to force a consistent and non-ambiguous design.

Finally, system reliability has turned out to be a critical issue. Although work can still be done when some components of the system are down, it is also true that there are more things that can go wrong. The communications network has been our biggest problem. Although the primary network is the 10 megabyte/sec Ethernet, the ability to use "uucp" or "kermit" over RS-232 backup links has proved essential. Critical items, such as laser printers and formatting software should be available on more than one machine.

The real future of the software engineering environment is in the use of graphic workstations coupled with AI techniques to create tools that truly aid the designer across all stages of the lifecycle. This requires a combination of interactive tools on heterogeneous workstations to provide a responsive user interface coupled with larger machines for more computationally intensive AI routines. Research issues include the determination of where AI can be applied in the design process and development of the corresponding expert knowledge, along with the development of techniques for integrating the heterogeneous environment in a manner transparent to the user.

The other primary research thrust planned is to utilize the instrumentation of the software engineering testbed to evaluate different software development methodologies, including rapid prototyping and object oriented design. In addition, investigation of tools and methodologies are needed for several specialized software development environments. These include VHDL, database design, AI systems, and parallel processing. Also being considered is the possible inclusion of other Computer-Aided Software Engineering (CASE) tools into the environments where source code is available. Examples include CASE packages from Textronix and McDonnell Douglas, DEASEL: an Expert System for Software Engineering (NASA) and the Software Engineering Testbed (Boeing/ Carnegie Group).

References

[1] Department of Defense. *Defense System Software Development.* Military Standard DOD-STD-2167, Department of Defense, Washington, D.C. 20301, June 1985.

[2] Steven M. Hadfield. *An Interactive and Automated Software Development Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1982. AD-A210 920.

[3] L. Osterwell. Software environment research: directions for the next five years. *COMPUTER*, 14(4):35–43, April 1981.

[4] Robert L. Glass. Persistent software errors. *IEEE Trans. on Software Engineering*, SE-7(2), March 1981.

[5] A. Wasserman. Automated development environments. *COMPUTER*, 14(4):7–10, April 1981.

[6] Douglas T. Ross. Structured analysis (sa): a language for communicating ideas. *IEEE Transactions on Software Engineering*, SE-3(1):16–34, January 1977.

[7] *An Introduction to SADT: Structured Analysis and Design Technique.* Softech, Inc., Waltham, Mass., 1976.

[8] SofTech, Inc. *Integrated Computer-Aided Manufacturing (ICAM) Function Modeling Manual (IDEF0).* User's Manual UM 110231100, Materials Laboratory, Air Force Wright Aeronautical Laboratories, Air Force Systems Command, Wright-Patterson AFB, OH, June 1981.

[9] Kevin Rose. *Development of Interactive Computer Graphics Software System and Graphics Tools.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1982.

[10] R. Balzer, N. Goldman, and D. Wile. On the transactional implementation approach to programming. In *Proceedings, 2nd International Conf. on Software Engr.*, Long Beach, CA., 1976.

[11] Meilir Page-Jones. *The Practical Guide to Structured Systems Design.* Yourdan Press, New York, 1980.

[12] David W. Fautheree. *An Analysis Tool in a Knowledge Based Software Engineering Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, March 1986. AD-A172 407.

[13] Stephen A. Wolfe. *A Natural Language Processor and Its Application to a Data Dictionary System.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1985. AD-A164 026.

[14] Charles W. Thomas. *An Automated/Interactive Software Engineering Tool to Generate Data Dictionaries.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1984. AD-A152 215.

[15] Jeffrey W. Foley. *Design of a Data Dictionary Editor in a Distributed Software Development Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, June 1986. AD-Ax172 406.

[16] Charles W. Hamberger, Jr. *Analysis, Definition, and Implementation of a Network-based Microcomputer Software Development Environment for the AFIT Digital Engineering Laboratory.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, March 1986. AD-A172 781.

[17] James W.Urscheler. *Design of a Requirement Analysis Design Tool Integrated with a Data Dictionary in a Distributed Software Development Environment.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1986. AD-A177 663.

[18] Thomas C. Hartrum and Charles W. Hamberger, Jr. Development of a distributed data dictionary system for software development. In *Proc. of IEEE 1986 Nat'l Aerospace and Elec. Conf., Vol. 3*, pages 648–655, Dayton, OH, May 1986.

[19] Thomas C. Mallary. *Design of the Human-Computer Interface for a Computer Aided Design Tool for the Normalization of Relations.* Master's thesis, Air Force Institute of Technology, Wright-Patterson AFB, OH, December 1985. AD-A164 100.

Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project

Bradley P. Allen
Peter L. Holtzman

Inference Corporation
5300 W. Century Blvd.
Los Angeles, CA 90045

Abstract

We provide an overview of the Automated Software Development Workstation Project, an effort to explore knowledge-based approaches to increasing software productivity. The project focuses on applying the concept of domain-specific automatic programming systems (D-SAPSs) to application domains at NASA's Johnson Space Center. We describe a version of a D-SAPS developed in the Phase I of the project for the domain of Space Station momentum management, and discuss how problems encountered during its implementation have led us to concentrate our efforts on simplifying the process of building and extending such systems. We propose to do this by attacking three observed bottlenecks in the D-SAPS development process through the increased automation of the acquisition of programming knowledge and the use of an object-oriented development methodology at all stages of program design. We discuss how these ideas are being implemented in the Bauhaus, a prototype CASE workstation for D-SAPS development.

1. Increasing software productivity through domain-specific automatic programming

Software development has come under criticism as an increasingly serious bottleneck in the construction of complex automated systems. Increasing the reuse of software designs and components has been viewed as an important way to address this problem, possibly increasing productivity by an order of magnitude or more [9]. A promising approach to achieving software reusability is through *domain-specific automatic programming*.

Domain-specific automatic programming systems (D-SAPS) use application domain knowledge to automate the refinement of a program description (written in a high-level domain language) into compilable code (written in a procedural target language) [1]. D-SAPSs can be distinguished from the more traditional domain-independent automatic programming systems in that the specification of the program is in a domain-specific language accessible to an end user, rather than a formal specification language (e.g. the predicate calculus with equality). Application generators of the type used in business report generation (e.g. Focus and DBASE-II) are examples of D-SAPSs in which the refinement process is completely automatic and implemented procedurally [10]. More complex domains can be handled if the user is allowed to interact with and guide the refinement process. Prototype knowledge-based systems that support user interaction and which work for practical application domains have been successfully developed (e.g. Draco [16], ΦNIX [3], and KBEmacs [25]).

2. The Automated Software Development Workstation Project

Since the fall of 1985, Inference Corporation has been involved in an effort, sponsored by NASA's Lyndon B. Johnson Space Center, to explore the applicability of domain-specific automatic programming to NASA software development efforts. Phase I of the project focused on the development of a D-SAPS for the domain

of Space Station momentum management [19]. During Phase I, A prototype D-SAPS was constructed, comprised of:

- a components catalog of FORTRAN subroutines used in the construction of Space Station orbital simulations;
- a design catalog of programs implemented using the system;
- a interactive graphical design system using a dataflow specification language for design editing and components composition;
- code generators for component interfaces, numerical subroutines and main programs; and
- a rule-based expert that:
  - proposed refinements for unimplemented modules in the dataflow specification;
  - flagged inconsistencies at manually-specified component interfaces; and
  - suggested possible workarounds to "patch" inconsistencies (e.g. coordinate system conversion routines).

The system was implemented by hand, to serve as a model for the implementation of similar D-SAPS for other NASA domains. The functionality and performance of the prototype was adequate to demonstrate the applicability of the D-SAPS approach to software development at NASA JSC. However, reflecting on our experience in building this system led us to be in accord with other D-SAPS developers in noting that:

- "domain analysis and design is *very hard*" [16]; and
- "domain-specific systems can be quite useful within their range of application, but the range is often quite narrow" [2].

We feel that these two issues must be addressed if D-SAPSs are to play a significant role in future software development environments. Therefore, in Phase II of the project, we are attempting to address the bottlenecks in the D-SAPS development process that lead to these observations.

3. Addressing the bottlenecks in D-SAPS development

We have focused on three bottlenecks in the D-SAPS development process that were observed during the development of the Phase I system:

- developing a domain language;
- describing design refinements and constraints; and
- describing the generation of target language code from a sufficiently detailed program description.

We plan to reduce the effort spent on each of these tasks by:

- automating the programming knowledge acquisition process; and
- using an object-oriented development methodology at all stages of the program design process.

We are currently implementing a knowledge-based D-SAPS development workstation, called the Bauhaus, that will embody these two approaches. We now describe how the design of the Bauhaus addresses the perceived bottlenecks.

3.1. Automating the programming knowledge acquisition process

By structuring the design process so that the types of knowledge required are made explicit, the knowledge acquisition process can be made simpler [14], and the resulting knowledge base easier to maintain [20]. To this end we are using a problem solving architecture based that of the RIME [24] and SOAR [12] systems, implemented using the ART expert system building tool [11]. This architecture allows us to organize design knowledge into a hierarchy of *problem spaces*, representing program design tasks. Each problem space consists of a set of operators for performing the task

represented by the space. In the Bauhaus, a problem space is associated with each program description that the system has in its knowledge base; the task represented by the problem space is that of refining the program description until it is sufficiently detailed to allow a code generator to translate it into code in the target language. The design process in the Bauhaus occurs in the following way:

1. **Select the initial design problem**: the user copies and edits an initial program description from the set of program descriptions in the knowledge base using a structure editor, making it the *current description*. The initial problem space is that associated with refining this description.

2. **Propose operators**: the Bauhaus determines what operators are applicable to the current description.

3. **Choose an operator**: the Bauhaus chooses an operator from the proposed set using operator preferences and constraints associated with the problem space, and implemented as ART production rules. User interaction is requested when the system reaches an *impasse*, where either no operator is known to apply, the system is unable to derive a preference for a specific operator, or the system's preferences are inconsistent [12]. This interaction takes one of two forms:

   - the user chooses a proposed operator for the system to apply; or
   - the user edits the current program description, in which case we return to step 2.

4. **Apply the chosen operator**: the Bauhaus applies the chosen operator to the current description. The operator may:

   - select a new problem space,
   - recurse into a problem subspace,
   - refine the current description,
   - signal that the task for the problem space is complete, or
   - signal that the task cannot be successful completed.

   We then return to step 2.

The design process terminates when the top-level task of refining the initial program description is successfully completed. This occurs when the description is detailed enough to allow the generation of target language code to occur. Given this problem solving architecture, we now discuss the knowledge acquisition mechanisms used to obtain the descriptions, operators, operators preferences and constraints used in the design process.

3.1.1. Acquiring descriptions

Our representation of domain objects and operations uses a description language, implemented in ART schemata, that is similar to KRYPTON [17]. New descriptions of domain objects and operations are created from existing descriptions using the copy&edit technique espoused by Lenat in the CYC system, [13] and are placed in the appropriate location in a subsumption hierarchy through an automatic classifier [8]. This use of description copy&edit together with automatic classification reduces the effort required to extend the domain language used to describe systems, by fostering reuse of existing domain languages in the creation of new domain languages. Using a subsumption hierarchy of descriptions as the organizing framework for the representation of objects and operations supports user access for copy&edit actions through a retrieval-by-reformulation browser similar in design to ARGON [18]. Retrieval-by-reformulation will permit a naive user of the Bauhaus to find a description needed for a copy&edit action more easily than using a tradition query mechanism [23].

3.1.2. Acquiring operators, operator preferences, and constraints

When the user performs a manual edit of a description in response to an impasse, the Bauhaus will create an operator whose condition is the current description and whose action is the manual editing action. Operator preferences are acquired by recording the conditions under which a user makes a selection from a set of

operators during an impasse where no operator is preferred. Constraints are acquired when a user manually rejects the application of an operator, causing the Bauhaus to backtrack to the previous problem solving state. This type of knowledge acquisition through the observation of manual programming steps taken by the user can be characterized as a *learning apprentice* approach [15]. In this respect, the Bauhaus is similar to the VEXED VLSI design system [22].

3.2. Using an object-oriented development methodology

By using object-oriented design (OOD) [5], we can decrease the level of effort required to implement a code generator that takes a sufficiently detailed program description and produces compilable target language code. This is due to the natural correspondence between the world and its model in an object-oriented framework [7]. In the Bauhaus, the world is the set of target language software components and code templates and the model is the set of descriptions of objects and sequences of operations in an application program. Ada's language level support of abstract data types and the existence of commercially supported reusable software component libraries constructed using OOD principles [6] make it our first choice as a target language in the Bauhaus system. The mapping between the description of an program and its realization in Ada code and the generation of the main subprogram in which the program objects are scoped is straightforward. We believe that the Bauhaus could easily be extended to support other languages with similar OOD features as target languages (e.g. Smalltalk, Objective-C, or ART).

4. System status and limitations

Implementation of the Bauhaus is currently underway using ART running on a Symbolics Lisp machine under the Genera 7.1 environment. Support for Ada compilation and library management is provided by the Symbolics Ada programming environment. As of July 1987, ART-based representations for descriptions, operators, operator preferences and constraints have been designed, the problem solving architecture and basic knowledge acquisition algorithms have been designed and implemented, and the target language reusable components library has been selected. The user interface is currently under construction, and the domain analysis for the demonstration domain, orbital flight simulation, is underway. We plan to demonstrate the use of the Bauhaus in the construction of a D-SAPS for this domain in the first quarter of 1988.

In the current design of the Bauhaus, there are a number of issues relevant to D-SAPSs that we do not address:

- **Lifecycle issues**: the Bauhaus is only useful as a programming-in-the-small environment, and ignores programming-in-the-large issues (e.g. version control). These would have to be addressed in a production-quality system.

- **Persistent object bases**: the Bauhaus has no provision for saving session state in a more sophisticated manner than simply saving changes out to a text file. We are looking to work on object-oriented databases to provide an answer here [4].

- **Automated algorithm synthesis**: the Bauhaus will always reach an impasse if a programming task requires algorithm design. However, the architecture should be extensible to encompass this kind of problem solving (e.g., see the work by Steier on the Cypress-Soar and Designer-Soar algorithm design systems [21]).

5. Conclusion

There is evidence that domain-specific automatic programming is a viable approach to increasing software productivity. To make this approach a practical one, the task of building and extending D-SAPS must be made simpler. As described above, we plan to accomplish this by improving the knowledge acquisition and software engineering methodologies used in constructing D-SAPS. Our ultimate goal is a production-

quality system that could be described as an "application generator generator"; i.e., a knowledge-based environment for the construction of special-purpose systems for the generation of applications software by end-users. Such a system could be available to systems analysts and designers in a DP/MIS organization for use when an applications programming task occurs frequently enough to merit the creation of a D-SAPS.

References

1. Barstow, D. "Domain-Specific Automatic Programming". *IEEE Transactions on Software Engineering 11*, 11 (November 1985).

2. Barstow, D. Artificial Intelligence and Software Engineering. Proceedings of the 9th International Conference on Software Engineering, IEEE, March-April, 1987.

3. Barstow, D., Duffey, R., Smoliar, S., and Vestal, S. An overview of ΦNIX. Proceedings of the Second National Conference on Artificial Intelligence, AAAI, August, 1982.

4. Bernstein, P.A. Database System Support for Software Engineering. Proceedings of the 9th International Conference on Software Engineering, IEEE, March-April, 1987.

5. Booch, G. "Object-Oriented Development". *IEEE Transactions on Software Engineering 12*, 2 (February 1986).

6. Booch, G. *Software Components With Ada.* Benjamin/Cummings Publishing, 1987.

7. Bordiga, A., Greenspan, S., and Mylopoulos, J. "Knowledge Representation as the Basis for Requirements Specification". *Computer 18*, 4 (April 1985).

8. Brachman, R.J. and Levesque, H.J. The Tractability of Subsumption in Frame-Based Description Languages. Proceedings of the National Conference on Artificial Intelligence, AAAI, August, 1984.

9. Horowitz, E. and Munson, J.B. "An Expansive View of Reusable Software". *IEEE Transactions on Software Engineering 10*, 5 (September 1984).

10. Horowitz, E., Kemper, A., and Narasimhan, B. Application Generators: Ideas for Programming Language Extensions. Proceedings of ACM'84 Annual Conference: The Fifth Generation Challenge, ACM, October, 1984.

11. Inference Corporation. *ART 3.0 Reference Manual.* Inference Corporation, 1987.

12. Laird, J.E., Newell, A. and Rosenbloom, P.S. "SOAR: An Architecture for General Intelligence". *Artificial Intelligence 33*, 1 (1987).

13. Lenat, D.B., Prakash, M., and Shepherd, M. "CYC: Using Common Sense Knowledge To Overcome Brittleness and Knowledge Acquisition Bottlenecks". *AI Magazine 6*, 4 (Winter 1986).

14. Marcus, S., McDermott, J., and Wang, T. Knowledge Acquisition for Constructive Systems. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, August, 1985.

15. Mitchell, T.M., Mahadevan, S., and Steinberg, L.I. LEAP: A Learning Apprentice for VLSI Design. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, August, 1985.

16. Neighbors, J.M. "The Draco Approach to Constructing Software from Reusable Components". *IEEE Transactions on Software Engineering 10*, 5 (September 1984).

17. Patel-Schneider, P.F. Small can be Beautiful in Knowledge Representation. Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems, December, 1984.

18. Patel-Schneider, P.F., Brachman, R.J., and Levesque, H.J. ARGON: Knowledge Representation meets Information Retrieval. Proceedings of the First Conference on Applications of Artificial Intelligence, IEEE, December, 1984.

19. Prouty, D.A. and Klahr, P. Automated Software Development Workstation. Proceedings of the Conference on AI for Space Applications, NASA, November, 1986.

20. Soloway, E., Bachant, J. and Jensen, K. Assessing the Maintainability of XCON-in-RIME: Coping with the Problems of a VERY Large Rule Base. Proceedings of the National Conference on Artificial Intelligence, AAAI, July, 1987.

21. Steier, D.M., Laird, J.E., Newell, A., Rosenbloom, P.S., Flynn, R.A., Golding, A., Polk, T.A., Shivers, O.G., Unruh, A. and Yost, G.R. Varieties of Learning in Soar: 1987. Proceedings of the Fourth International Workshop on Machine Learning, June, 1987.

22. Steinberg, L.I. Design as Refinement Plus Constraint Propagation: The VEXED Experience. Proceedings of the National Conference on Artificial Intelligence, AAAI, July, 1987.

23. Tou, F.N., Williams, M.D., Fikes, R., Henderson, A., and Malone, T. RABBIT: An Intelligent Database Assistant. Proceedings of the Second National Conference on Artificial Intelligence, AAAI, August, 1982.

24. van de Brug, A., Bachant, J. and McDermott, J. "The Taming of R1". *IEEE Expert 1*, 3 (Fall 1986).

25. Waters, R.C. "The Programmer's Apprentice: A Session with KBEmacs". *IEEE Transactions on Software Engineering 11*, 11 (November 1985).

THE KNOWLEDGE-BASED SOFTWARE ASSISTANT

Lt Kevin M. Benner, USAF and Douglas A. White
Command and Control Technology Division
Rome Air Development Center
Griffiss AFB, NY 13441-5700

ABSTRACT

In 1983, Rome Air Development Center (RADC) published "Report on a Knowledge-Based Software Assistant" [Green, et al 83]. This document brought together key ideas on how artificial intelligence (AI) could be used in the software development process. Since then RADC has embarked on the first of three contract iterations to develop both a Knowledge-Based Software Assistant (KBSA) and the enabling supporting technologies which are necessary. KBSA is a formalized computer-assisted paradigm for the development, evolution, and long-term maintenance of computer software. KBSA captures the history of system evolution. It provides a corporate memory of: how parts interact, what assumptions were made and why, the rationale behind choices, how requirements are satisfied, and explanation of the development process. KBSA accomplishes this through a collection of integrated dedicated facets. Their areas of expertise are: project management, requirements, specifications, implementation, performance, testing, and documentation.

RADC is currently in the midst of the first iteration. Facets which are now under contract include: Requirements Assistant with Sanders Associates, Specifications Assistant with Information Sciences Institute (ISI) at USC, Performance Assistant and Project Management Assistant with Kestrel Institute, and the KBSA Framework with Honeywell. This paper will first describe where the KBSA program is now, four years after the initial report; secondly, describe what RADC expects at the end of the first contract iteration; and finally, characterize what the second and third contract iterations will look like.

INTRODUCTION

In 1983 Rome Air Development Center (RADC) published "Report on a Knowledge-Based Software Assistant" [3]. This document brought together key ideas on how artificial intelligence (AI) could be used in the software development process. Since then RADC has embarked on the first of three contract iterations to develop both a Knowledge-Based Software Assistant (KBSA) and the enabling supporting technologies. This paper will describe: 1) History leading up to KBSA, 2) What is KBSA, 3) Development strategy and current status of KBSA, and 4) Concluding remarks.

HISTORY

The KBSA research program is a natural progression of research and development undertaken by RADC in its continuing pursuit of a solution to the well known "software life cycle problem". This application of AI technology to the problem of software development was a predictable outgrowth of RADC's longstanding commitment to research and development directed at enhancing productivity. From the early 1970's when RADC was championing the cause of "modern" high level languages and "structured" implementation methodologies, a less publicized but important track of research was being addressed on a smaller scale for the

development of greater formalism and abstraction for the objects and activities belonging to the technology of software development. The publicity given to the AI community in the late 1970's and early 1980's due to successes in building workable expert systems and the announcement of the Fifth Generation Computer Program of Japan, resulted in the emphasis of AI technology within RADC's research program. This in turn led to the examination of the possibility of applying the technology that worked so well in areas such as geological analysis and locomotive maintenance to the problems of software development and maintenance. This atmosphere, when coupled with the growing compilation of results from earlier research, encouraged the selection of software development as an application to drive and demonstrate AI technology research developments. The particular paradigm selected and eventually identified as the KBSA was the result of careful consideration of the state of technology as demonstrated by prior work, and the goals and practical requirements of a system to support software development.

Throughout the 1970's, concurrent with the major RADC projects in language/compiler systems and programming methodologies, efforts were undertaken which explored formalisms with which to better describe the objects and algorithms comprising software. It was recognized that there were many flaws with the existing manner in which programs were created and the languages in which they were expressed. A few of the outstanding problems that were addressed during this time included: programs could be syntactically correct without providing the desired solution or being logically correct; and, even with "better" high level languages, programs were incomprehensible, even to the author with the passing of time. Each of these problems and the research efforts addressing them had a part in exposing the members of the Command and Control Software Technology Division to the work being performed in the world of AI, and while simultaneously providing necessary support for some of the important AI research ideas of the time.

The problem of logical correctness of programs was attacked in many ways. However, the two that are important to the KBSA (even though not receiving widespread adoption in the world of software engineering) are formal proofs of correctness and formal specification languages. In late 1974 an initial effort was undertaken to explore the applicability of formal verification methods to existing programming languages. This process of "verifying" a program's correctness consists of establishing by mathematical proof that whenever a program is executed with specified input data and execution environment the execution will terminate and upon termination the values of program variables will meet output specifications. The foundations of formal program verification are identical with those of a significant body of the work in automatic programming. As might be expected, many of the same individuals are involved in both areas of research. The need for formal specification languages was emphasized by the difficulty of verification of programs in existing computer languages. In 1976, research was initiated to develop a "language" that could be used to provide formal descriptions of programs. The specification languages resulting from this research were found to be unwieldy for extensive use by humans and as is the case with formal program verification technology are not known to have achieved widespread use. However, formal specifications are particularly important to the KBSA because they provide the formalism needed to enable reasoning about programs. Additionally, these particular research efforts caused RADC to become involved in a progression of efforts addressing automatic programming which continue today.

The need for a more natural and abstract method of expressing a problem solution to a computer led to the exploration of the concept of a Very High Level Language (VHLL). The goal of this research was to provide a language system combining the capabilities of conventional languages with those of logic programming systems enabling the user to program not only computational processes in the conventional sense, but also "reasoning" processes. From this research at Syracuse has emerged an evolving family of languages which exhibit many of the characteristics that will be needed in the Wide Spectrum Language (WSL) of the KBSA. This research was facilitated by the early theoretical work of Robinson [14] which has provided much of the foundation for logic programming. Through this work at Syracuse University, which began in the mid 1970's, RADC has been cognizant of the potential for a software development paradigm unlike that existing for conventional programming languages.

The arrival of practical diagnostic systems based on AI technology in the late 1970's led to a project to investigate the possibility of creating a knowledge-based system that would diagnose software systems and assist in their maintenance. The conclusion of this investigation was that this type of diagnostic expert system would be impossible for software because of the inadequacy of the knowledge about software in general and any software system in particular. This initial negative result, along with the dim prospect for immediate relief from automatic programming caused a serious consideration of the alternatives. The alternative perceived to have the greatest promise was that of a knowledge-based system that did not provide total automation of the software synthesis process, but did maintain a total record of all decisions and activities which occurred in the creation of a software system. This system would possess the expertise to automatically perform many of the tedious tasks of program development, but would be guided in the application of transformations by the human user. Communications among members of a development organization would also be enhanced by the monitoring and reporting capabilities provided by the knowledge-based system. These are the concepts that were further developed and described in the 1983 report considered to be the "defining document" of the KBSA.

WHAT IS KBSA?

The KBSA approach is a departure from the existing software engineering paradigm in that it attempts to formalize all activities as well as products of the software life cycle. It is a formalized computer-assisted paradigm for the development, evolution, and long-term maintenance of computer software. KBSA captures the history of system evolution. It provides a corporate memory of: how parts interact, what assumptions were made and why, the rationale behind choices, how requirements are satisfied, and explanation of the development process. KBSA accomplishes this through a collection of integrated dedicated facets and an underlying common framework.

KBSA has four main distinguishing features. First, the specification is incremental, executable, and formal. Incremental means that the specifier may gradually add more detail to the specification and is not forced to initially describe the system in complete detail. Executable means that the specification is "runnable" like a prototype. This allows the specifier to validate the specification against user intent by actually showing him/her the "running" specification. Finally, formal means that the specification is expressed in a language with precise semantics, avoiding the ambiguity of natural language.

Second, the implementation is formal, that is, all decisions made during the implementation are captured and justified. Typically, implementation will be done via correctness preserving transformations, thus guaranteeing by default a verified implementation.

Third, project management policies will be formally stated and enforced by KBSA. That is, project policy will define the relationship between various software development objects (eg. requirements, specifications, code, test cases, bug reports, etc.) and then be enforced by KBSA throughout the software development process.

Fourth, and finally, maintenance will be done at the requirements and specification level, rather than via patches to the code. That is, since maintenance activities are normally a result of new or better defined user requirements, it makes sense to reflect this in the requirement/specification.

In order to build a KBSA, the authors of the initial report point to the need for specific supporting technologies. These supporting technologies fall into four main categories: a wide spectrum language, general inferential systems, domain specific inferential systems, and system integration.

A wide spectrum language (WSL) is a single language which provides the user with the ability to capture the formal semantics of the system under development regardless of the level of detail (or the step in the development cycle). A wide spectrum language is both a language and an environment. It must provide uniform expressibility, regardless of what is being described (ie.

requirements, specifications, code, test cases, project management policy, etc). Not only must a WSL be able to express these objects, it must do so in a way which is consistent at all levels, both syntactically and semantically.

A general inferential system is a system which supports reasoning. In particular, we are concerned with the overall efficiency of this reasoning, how to capture such things as logic in inference rules and data structures, the quality of explanation generated by the system, and the ability to apply this inferencing power to specific domains.

Domain specific inferential systems extend general inferential systems to include aspects unique to software development. This topic focuses on the knowledge representation of software development objects and inference rules and, in particular, how they can be formally represented and used for further reasoning.

System integration deals with the inherent competition between facets and how a technology base can be put together such that all phases in the software development process are supported sufficiently.

DEVELOPMENT STRATEGY AND CURRENT STATUS OF KBSA

When the KBSA report first came out, it was clear that the supporting technologies were not adequately developed. To address this shortfall a KBSA was to be developed in three iterations. The first iteration was aimed at designing the individual facets and seeing where the supporting technologies could be pulled along. Additionally, there was a desire for an advancement of the understanding of the software development process, particularly within this new KBSA development paradigm. In line with this concept, work began on a Framework (FW) and five (5) facets: Project Management Assistant (PMA), Requirements Assistant (RA), Specification Assistant (SA), Performance Assistant (PA), and Development Assistant (DA) [DA will not begin until FY 88]. Though boundaries between facets may appear in the first iteration, they will blur in the second iteration and disappear completely in the third iteration.

First Iteration

The results of this have been twofold. First, each facet has pulled at the supporting technologies such that they have advanced the state of these technologies. Universal solutions were not sought, rather solutions unique for each facet have been found. Secondly, each facet developer has made progress in formalizing their particular life cycle phase. This formalization has focused both on the products of individual phases (eg. requirements, specification, and code), but more importantly on the process of how these products came about. In this section the basic approach and milestones of each contractor will be described. Included in this description will be the formalization of the particular life cycle phase and the impact on supporting technologies.

Work on the definition of a PMA formalism and construction of a prototype began in 1984 [9]. Kestrel Institute was the developer. The life cycle goals of PMA were to provide knowledge-based help to users and managers in project communication, coordination, and task management.

The capabilities of PMA fall into three categories: project definition, project monitoring, and user interface. Project definition consists of structuring the project into individual tasks and then scheduling and assigning these tasks. Once the project has been decomposed into manageable tasks, it must be monitored. This monitoring is in the form of cost and schedule constraints. Also included in monitoring is the enforcement of specific management policies (eg.DoD-Std-2167, rapid prototyping, KBSA, etc.). In addition, PMA provides a good user interface for project monitoring and project definition. This interaction is in the form of direct queries/updates, Pert Charts, and Gantt Charts.

The above capabilities were important, but would be expected of any project management tool. What sets PMA apart from its predecessors is the

expressibility and flexibility of the PMA architecture. Not only does PMA handle user defined tasks, but it also understands their products and the implicit relationships between them (eg. components, tasks, requirements, specification, source code, test cases, test results, and milestones). Also present in PMA are objects unique to programming-in-the-large (ie. versions, configurations, derivations, releases, and people).

From a technical perspective, the advances made in PMA include: the formalization of the software development objects enumerated above, the development of a powerful time calculus for representing temporal relationships between software development objects [10], and a mechanism for directly expressing and enforcing project policies.

The work on the Requirements Assistant [2, 15] began in 1985 by Sanders Associates. The main task of RA was to deal with the informal nature of the requirements process. Sanders' intention was to allow the user to enter requirements in any desired order or desired level of detail. It would be the responsibility of RA to: 1) do the necessary bookkeeping to allow user manipulation of requirements and 2) maintain consistency among requirements as they become known.

RA capabilities include: support for multiple viewpoints (eg. data flow, control flow, state transition, and functional flow diagrams), management and smart editing tools to organize the requirements, and the ability to support free form annotations to requirements. In addition to this, RA's underlying knowledge representation, Structured Object and Constraint Language Environment (SOCLE), enables RA to identify contradictions and generate explanations.

The main technical thrust has been on how to handle informality when trying to build an underlying representation of the requirements. This is done by supporting incomplete graphical descriptions at the user interface level, but maintaining a consistent, though not necessarily complete, internal representation. This is done via SOCLE that provides a truth maintenance system which supports default reasoning, dependency tracing, and local propagation of constraints. RA provides application specific automatic classification which is used to identify missing requirements by comparing the current requirements against "typical requirements" of a generic system (already represented within RA's knowledge base). This comparison is then used to generate questions of the specifier to either be sure something vital has not been left out or to gather a justification for the difference.

The main goal of the Specification Assistant [1, 8] is to develop a formal specification of the system under development and then to validate it against user intent. The development of the formal specification must be supported in an incremental fashion, modeling the way developers typically construct specifications. The validation must be done by exposing the specification to the user at the earliest opportunity and continued throughout the construction process. The effort to develop a KBSA Specification Assistant began in 1985. This work is being done at the University of Southern California- Information Sciences Institute (ISI).

SA capabilities include: an incremental specification language which is executable and a natural language paraphraser which will translate a given specification into English. These capabilities have been built on top of ISI's Wide Spectrum Language AP5, and the development environment CLF. SA can currently handle specifications of a couple pages.

The main technical issues concern: 1) identification of specification statements as requirements or goals and the transformation of these from a high level specification into a low level specification and 2) extracting and assembling system views (ie. reusing specifications and parts of specifications).

The distinction that ISI makes between requirements and goals is that requirements are inviolable constraints, while goals describe general behavior which may have exceptional cases not currently covered by the goal. With this distinction in mind SA provides high level editing commands to further transform the specification into a low level specification. Requirements are transformed in a correctness preserving manner to maintain satisfaction of the requirements.

Goals, on the other hand, may be "compromised" in order to handle exceptional cases. That is, after a transformation, the meaning of a goal specification may change.

Extracting and assembling system views deals with building up a specification from smaller specifications (ie. reuse of other previously defined specifications) as opposed to the top down refinement presented in the previous paragraph. SA can combine disjoint specifications, but tools are needed to aid in combining specifications which share common terms.

The Performance Assistant [4, 5, 6] work began at Kestrel Institute in 1985 and is expected to run until 1990. Long term goals for a performance facet are to guide software performance decisions at many levels in the software development cycle, from requirements specifications in very high level programs to low level code. The approach is to combine heuristic, symbolic, and statistical approaches which will provide capabilities for: symbolic evaluation, data structure analysis and advice, and algorithm design analysis and advice. This effort is focusing on data structure selection, performance annotations of a specification, analysis and propagation of performance information, and control structure performance analysis.

Technical issues that have been addressed thus far are data structure selection (DSS) using symbolic and heuristic techniques and the development of PERFORMO, a functional specification language with set theoretic data types. PERFORMO is similar to VAL [12], developed at MIT, and SISAL [13], developed at Lawrence Livermore Laboratory. PERFORMO is intended primarily for DSS work, but is sufficiently expressive to be a good initial specification language for the next two research issues: subroutine decomposition and control flow optimization.

The basic strategy employed in DSS is to supply refinement decisions when they are needed by the implementation generator (ideally this would be the DA). When a refinement decision is needed, PA determines the relevant program properties necessary to make a satisfactory selection. The relevant properties would vary on where in the implementation the generator is. Properties refer to how a specific variable will be used and some characteristics of it. These properties could include: whether the variable is random access, ordered, enumerated, dynamic, and/or possibly empty. Based on these properties, specific implementation decisions can be made.

Development Assistant is the most recent facet undertaken, although contract work has not yet begun. RADC will award the DA contract in early FY 88. The basic thrust of this effort will be to derive an implementation from a completed specification, automating (via automatic transformation) where possible and capturing user supplied design decisions when needed.

The Framework [7, 11] was considered to be necessary to bring a global perspective to KBSA. Initial work on the FW began at Honeywell Systems and Research Center in early 1986. The goal of the Framework is twofold: 1) to develop an integrated KBSA demonstration and 2) to propose the specification of a KBSA framework through which all facets must interact and communicate. The purpose of the former is to provide a concept definition that would be intuitively obvious to the most casual observer. The purpose of the latter is to facilitate a tightly coupled interaction between facets. The framework will provide a common reference for each facet developer allowing the sharing of information. Interacting with the framework will be a requirement for all second iteration contracts. The result in the future will be a more tightly integrated KBSA.

The main technical issues are 1) define minimum functionality which the framework must provide to all facets, 2) define a common interface to the framework, 3) extend the framework to a distributed environment, 4) support programming-in-the-large concepts like configuration control, and 5) provide a consistent user interface.

The overall results of the first iteration will be a KBSA concept demonstration consisting of mostly loosely coupled facets with the exception of PMA and the framework which will be tightly coupled. Each facet will exist on separate machines and communicate via the framework. The framework will be responsible

for maintaining traceability between software development objects and keeping facets updated. Establishing the initial traceability (eg. the relationship between requirements objects and specification objects) is the responsibility of the involved facets.

In the past, each facet developer has had their own problem domain in which to work. In general these domains have been small or toy-like. For the KBSA demonstration the problem domain will be the air traffic control problem (ATC). This domain has the advantage of being a substantial problem with a variety of real world issues (ie. real time requirements, data base management, user interaction, interaction with the outside world, and changing or not well defined requirements). The intention of the demonstration is not to solve the ATC problem, but rather to have the ATC requirements be a driver for KBSA. The demonstration will most likely focus on some portion of the overall ATC problem.

Second And Third Iterations

The second iteration of KBSA will begin at the completion of the framework effort. All facets in the second iteration will interact with the framework and thus each other. This will be done by either building individual facets in Honeywell's framework, or more likely, individual facet developers will extend their own frameworks (eg. REFINE, AP5, SOCLE, etc) to adhere to the framework specification. Both are acceptable from a RADC perspective. Prospective developers must convince an RADC that either 1) they will use the Honeywell framework or 2) that their framework does or will soon adhere to the standard. There will be a mechanism to allow for some deviations from the framework specification. This is important since at the beginning of the second iteration the framework described in the specification may not be sufficiently powerful to implement all facets or some specific feature of the framework may preclude functionality necessary for a particular facet. For the third iteration there will be no exceptions.

During the second iteration, work will continue on individual life cycle phases, while also focusing on the interaction between facets and the framework.

For the framework contract, work will address raising the functional level of the framework. The goal is for individual facets to be concerned only with activities unique to their respective facets, while the framework will be responsible for all generic tasks (eg. knowledge representation, knowledge base maintenance, communication between facets, policy enforcement, and general inferencing capabilities).

CONCLUSION

In conclusion, there has been progress over the last four years. The question now is, how close are we to a workable KBSA? The answer is greatly dependent upon the framework specification which will come out of this first iteration. If it is sufficiently powerful, we could have a workable KBSA at the end of the second iteration. On the other hand, if most second iteration contractors have to make generic extensions to the framework, we can not expect a workable KBSA until the third iteration.

REFERENCES

[1] Balzer, R. et al., "Knowledge-Based Specification Assistant", Interim Technical Report, RADC, Griffiss AFB, NY, Dec., 1986.

[2] Czuchry, A. J. Jr., "Where's the Intelligence in the Intelligent Assistant for Requirements Analysis?", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[3] Green, C. et al., " Report on a Knowledge-Based Software Assistant," RADC Tech. Report TR-83-195, RADC, Griffiss AFB, NY, Aug, 1983.

[4] Goldberg, A. and Smith, D., "Towards a Performance Assistant", Interim Technical Report, RADC, Griffiss AFB, NY, Nov., 1986.

[5] Goldberg, A. and Smith, D., "Performance Estimation for a Knowledge-Based Software Assistant", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[6] Goldberg, A., "Technical Issues for Performance Estimation", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[7] Huseth, S. and King, T., "A Common Framework for Knowledge-Based Programming", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[8] Johnson, W. J., "Turning Ideas into Specifications", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[9] Jullig, R., et al., "KBSA-PMA Technical Report", Final Technical Report, RADC, Griffiss, NY, Nov., 1986.

[10] Ladkin, P., "Primitives and Units for Time Specification", AAAI-86, Philadelphia, PA, Aug. 11- 15, 1986.

[11] Larson, A. and Huseth, S., "KBSA Common Framework Implementation", RADC, 2nd Annual KBSA Conference, Utica, NY, Aug 18-20, 1987.

[12] Robinson, J. A., "A Machine Oriented Logic Based on the Resolution Principle", Journal of the ACM, Jan., 1965.

[13] McGraw, J. R., "The VAL Language: Description and Analysis", ACM Transactions on Programming Languages and Systems, Jan., 1982.

[14] McGraw, J. R., et. al, "SISAL: Streams and Iteration in a Single Assignment Language", Technical Report M-146, Lawrence Livermore Laboratory, Mar., 1985.

[15] Sanders Associates, "Knowledge Based Requirements Assistant", Interim Technical Report, RADC, Griffiss AFB, NY, Mar., 1986.

AN INTELLIGENT TRAINING SYSTEM FOR PAYLOAD-ASSIST MODULE DEPLOYS

R. Bowen Loftin
University of Houston-Downtown
One Main Street
Houston, TX 77002

Lui Wang, Paul Baffes, and Monica Rua
Artificial Intelligence Section, FM72
NASA/Johnson Space Center
Houston, TX 77058

ABSTRACT

An autonomous intelligent training system which integrates expert system technology with training/teaching methodologies is described. The system was designed for use by Mission Control Center (MCC) Flight Dynamics Officers (FDOs) training to perform payload-assist module (PAM) deploys from the orbiter. The system (termed PD/ICAT for Payload-assist module Deploys/Intelligent Computer-Aided Training system) is composed of five distinct components: a user interface, a domain expert, a training session manager, a trainee model, and a training scenario generator. A user interface has been developed which permits the trainee to access data in the same format as it is presented on console displays in the MCC. The interface also permits the trainee to take actions in much the same manner as a FDO in the MCC and provides the trainee with information on the current training environment and with on-line help (if permitted by the training session manager). The domain expert (DeplEx for Deploy Expert) contains the rules and procedural knowledge needed by a FDO to carry out a PAM deploy. DeplEx also contains "mal-rules" which permit the identification and diagnosis of common errors made by the trainee. The training session manager (TSM) examines the actions of the trainee and compares them with the actions of DeplEx in order to determine appropriate responses. A unique feature of the TSM is its ability to grant the trainee the freedom to follow any valid path between two stages of the deploy process. A trainee model is developed for each individual using the system. The model includes a history of the trainee's interactions with the training system and provides evaluative data on the trainee's current skill level. Following each trainee action, evaluative assertions are made by the TSM and used to update the trainee model. A training scenario generator designs appropriate training exercises for each trainee based on the trainee model and the training goals. PD/ICAT is currently being tested by both experienced and novice FDOs in order to refine the system and determine its efficacy as a training tool. Ultimately, this project will provide a vehicle for developing a general architecture for intelligent training systems together with a software environment for creating such systems.

INTRODUCTION

The Mission Operations Directorate (MOD) at NASA/Johnson Space Center is responsible for the ground control of all space shuttle operations. Those operations which involve alterations in the shuttle's orbit characteristics are guided by a FDO who sits at a console in the "front room" of the MCC. Currently, the training of the FDOs (called "fidos") in flight operations is carried out principally through the study of flight rules, training manuals, and "on-the-job training" (OJT) in integrated simulations. From two to four years is normally required for a trainee FDO to be certified for many of the tasks for which he is responsible during shuttle missions. OJT is highly labor intensive and presupposes the availability of experienced personnel with both the time and ability to train novices. As the number of experienced FDOs has been reduced through retirement, transfer (especially of Air Force personnel), and promotion and as the preparation for and actual control of missions occupies most of the MCC's available schedule, OJT has

become increasingly difficult to deliver to novice FDOs. As a supplement to the existing modes of training, the Orbit Design Section (ODS) of the MOD has requested that the Artificial Intelligence Section (AIS) of the Mission Support Directorate assist in developing an autonomous intelligent computer-aided training system. After extensive consultation with ODS personnel, a particular task was chosen to serve as a proof of concept: the deployment of a PAM satellite from the shuttle. This task is complex, mission-critical and requires skills used by the experienced FDO in performing many of the other operations which are his responsibility.

The training system is designed to aid novice FDOs in acquiring the experience necessary to carry out a PAM deploy in an integrated simulation. It is intended to permit extensive practice with both nominal deploy exercises and others containing typical problems. After successfully completing training exercises which contain the most difficult problems, together with realistic time constraints and distractions, the trainee should be able to successfully complete an integrated simulation of a PAM deploy without aid from an experienced FDO. The philosophy of the PD/ICAT system is to emulate, to the extent possible, the behavior of an experienced FDO devoting his full time and attention to the training of a novice--proposing challenging training scenarios, monitoring and evaluating the actions of the trainee, providing meaningful comments in response to trainee errors, responding to trainee requests for information and hints (if appropriate), and remembering the strengths and weaknesses displayed by the trainee so that appropriate future exercises can be designed.

BACKGROUND

Since the 1970's a number of academic and industrial researchers have explored the application of artificial intelligence concepts to the task of teaching a variety of subjects (e.g., geometry, computer programming, medical diagnosis, and electronic troubleshooting). A body of literature is now extant on student models and teaching/tutoring methodologies adapted to intelligent tutoring systems in the academic environment[1]. The earliest published reports which suggested the applications of artificial intelligence concepts to teaching tasks appeared in the early 1970's.[2,3] Hartley and Sleeman[3] actually proposed an architecture for an intelligent tutoring system. However, it is interesting to note that, in the fourteen years which have passed since the appearance of the Hartley and Sleeman proposal, no agreement has been reached among researchers on a general architecture for intelligent tutoring systems[4]. Nonetheless, a study of the literature on intelligent tutoring systems is an essential starting point for the development of the elements of an intelligent training system.

Among the more notable intelligent tutoring systems reported to date are SOPHIE[5], PROUST[6] and the LISP Tutor[7]. The first of these systems, SOPHIE, was developed in response to a U.S. Air Force interest in a computer-based training course in electronic troubleshooting. SOPHIE contains three major components: an electronics expert with a general knowledge of electronic circuits, together with detailed knowledge about a particular type of circuit (in SOPHIE this was an IP-28 regulated power supply); a coach which examines student inputs and decides if it is appropriate to stop the student and offer advice; and a troubleshooting expert that uses the electronics expert to determine which possible measurements are most useful in a particular context. Three versions of SOPHIE were produced and used for a time but none was ever viewed as a "finished" product. One of the major lacks of the SOPHIE systems was a user model. It is interesting to note that the development of a natural language interface for SOPHIE represented a large portion of the total task.

PROUST and the LISP Tutor are two well-known intelligent tutoring systems that have left the laboratory and found wider applications. PROUST (and its offspring, Micro-PROUST) serves as a "debugger" for finding nonsyntactical errors in Pascal programs written by student programmers. The developers of PROUST claim that it is capable of finding all of the bugs in at least seventy percent of the "moderately complex" programming assignments that its examines. PROUST contains an expert Pascal programmer that can write "good" programs for the assignments given to students. Bugs are found by matching

the expert's program with that of the student; mismatches are identified as "bugs" in the student program. This ability is contained in the PROUST "bug rule" component. After finding a bug, PROUST provides an English-language description of the bug to the student, enabling the student to correct his error. The system cannot handle student programs that depart radically from the programming "style" of the expert. The LISP Tutor is currently used to teach the introductory Lisp course offered at Carnegie-Mellon University. This system is based on the ACT (historically, Adaptive Control of Thought) theory and consists of four elements: a structured editor which serves as an interface to the system for students, an expert Lisp programmer that provides an "ideal" solution to a programming problem, a bug catalog that contains errors made by novice programmers, and a tutoring component that provides both immediate feedback and guidance to the student. Evaluations of the LISP Tutor show that it can achieve results similar to those obtained by human tutors. One of its primary features is its enforcement of what its authors regard as a "good" programming style.

TRAINING VERSUS TUTORING

The PD/ICAT system was developed with a clear understanding that training is not the same as teaching or tutoring[8]. The NASA training environment differs in many ways from an academic teaching environment. These differences are important in the design of an architecture for an intelligent training system:

a. Assigned tasks are often mission-critical, placing the responsibility for lives and property in the hands of those who have been trained.
b. Personnel already have significant academic and practical experience to bring to bear on their assigned task.
c. Trainees make use of a wide variety of training techniques, ranging from the study of comprehensive training manuals to simulations to actual on-the-job training under the supervision of more experienced personnel.
d. Many of the tasks offer considerable freedom in the exact manner in which they may be accomplished.

FDO trainees are well aware of the importance of their job and the probable consequences of failure. While students are often motivated by the fear of receiving a low grade, FDO trainees know that human lives, a billion dollar shuttle, and a $100+ million satellite depend on their skill in performing assigned tasks. This means that trainees are highly motivated, but it also imposes on the trainer the responsibility for the accuracy of the training content (i.e., verification of the domain expertise encoded in the system) and the ability of the trainer to correctly evaluate trainee actions. PD/ICAT is intended, not to impart basic knowledge of mathematics and physics, but to aid the trainee in developing skills for which he already has the basic or "theoretical" knowledge. In short, this training system is designed to help a trainee put into practice that which he already intellectually understands. The system must take into account the type of training that both precedes and follows--building on the knowledge gained from training manuals and rule books while preparing the trainee for and complementing the on-the-job training which will follow. Perhaps most critical of all, trainees must be allowed to carry out an assigned task by any valid means. Such flexibility is essential so that trainees are able to retain and even hone an independence of thought and develop confidence in their ability to respond to problems, even problems which they have never encountered and which their trainers never anticipated.

SYSTEM DESIGN

The PD/ICAT system is modular and consists of five basic components:

1. A user interface that permits the trainee to access the same information available to him in the MCC and serves as a means for the trainee to assert actions and communicate with the intelligent training system
2. A domain expert (DeplEx) which can carry out the deployment process using the same information that is available to the trainee and which also contains a list of "mal-rules" (explicitly identified errors that novice trainees commonly make).
3. A training session manager TSM) which examines the assertions made

by the DeplEx (of both correct and incorrect actions in a particular context) and by the trainee. Evaluative assertions are made following each trainee action. In addition, guidance can be provided to the trainee if appropriate for his skill level.

4. A trainee model which contains a history of the individual trainee's interactions with the system together with summary evaluative data.
5. A training scenario generator that designs increasingly-complex training exercises based on the current skill level contained in the trainee's model and on any weaknesses or deficiencies that the trainee has exhibited in previous interactions.

Figure 1 contains a schematic diagram of the

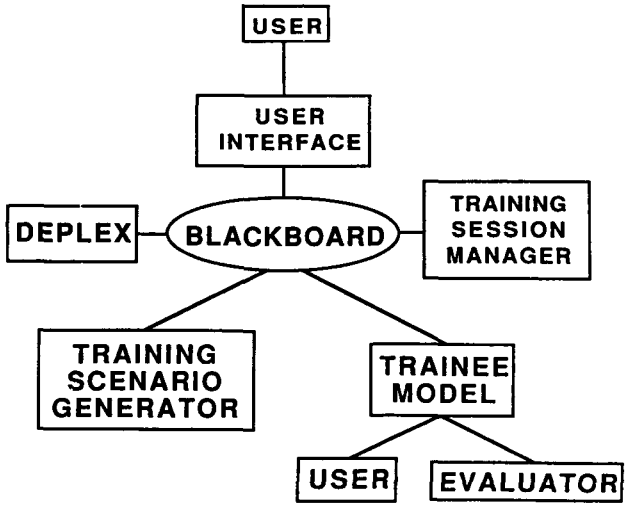


FIGURE 1 - PD/ICAT ARCHITECTURE

PD/ICAT system. Note that provision is made for the user to interact with the system in two distinct ways and that a supervisor may also query the system for evaluative data on each trainee. The blackboard serves as a common "factbase" for all five system components. With the exception of the trainee model, each component makes assertions to the blackboard, and the rule-based components look to the blackboard for facts against which to pattern match the left-hand sides of their rules.

User Interface

The primary factor influencing the interface design was fidelity to the task environment. To avoid negative training, it was deemed essential that the functionality and, to the extent possible, the actual appearance of the training environment duplicate that in which the task is performed. Figure 2 contains a view of the

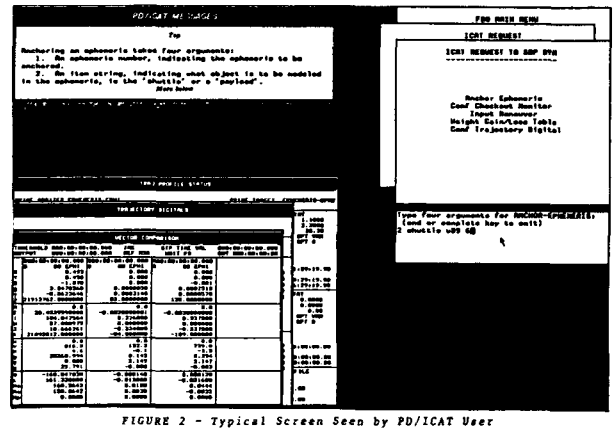


FIGURE 2 - Typical Screen Seen by PD/ICAT User

typical display seen by a trainee on a Symbolics 3600 series LISP machine. The upper right corner of the display contains menus that allow the trainee to make requests of other flight controllers, respond to requests from other controllers, call up displays, obtain information about the current or previous step in the deploy process, request help from the training system, and return to a previous step in the process. This menu has as many as three levels depending on the nature of the action taken by the trainee. Some actions are completely menu driven while others require the input of one or more "arguments". All actions taken by the trainee through these menus and the arguments that they may require become assertions to the blackboard. All requests directed to the trainee and all messages sent to the trainee in response to his requests or actions appear in a window in the upper left corner of the screen. These two portions of the screen serve to functionally represent the voice loop interactions that characterize the current FDO task environment. Any displays requested by the trainee appear in the lower portion of the screen,

overlapped, if more than one is requested. Clicking the mouse on any exposed portion of a background display will bring it to the foreground. The displays replicate those seen by a FDO on console in the MCC. During development nominal data was supplied to these displays (from a dedicated ephemeris-generating program or from "dummy" data sets) so that negative training would not occur. Experienced FDOs using PD/ICAT have expressed satisfaction with the user interface.

DeplEx

The Deploy Expert is a "traditional" expert system in that it contains if-then rules which access data describing the deploy environment and is capable of executing the PAM deploy process and arriving at the correct "answers". In addition to "knowing" the right way to conduct the PAM deploy, DeplEx also contains knowledge of the typical errors that are made by novice FDOs. In this way, PD/ICAT can not only detect an erroneous action made by a trainee, but also, through these so-called "malrules", it can diagnose the nature of the error and provide an error message to the trainee specifically designed to inform the trainee about the exact error made and correct the misconception or lack of knowledge which led to the commission of that error. Another of the interesting features of the PD/ICAT system is its continual awareness of the environment (the external constraints dictated by the training exercise) and the context of the exercise. Rather than having DeplEx generate a complete and correct solution to the deployment problem, only those actions which are germane to the current context are asserted. In this way the expert "adapts" to alternate, but correct, paths that the trainee might choose to follow. Figure 3 shows schematically how DeplEx operates. This strategy was adopted because the human experts that perform PAM deploys recognize that many steps in the deploy process may be accomplished by two or more equally valid sequences of actions. To grant freedom of choice to the FDO trainee and to encourage independence on his part, the experts felt that it was essential to build this type of flexibility into the PD/ICAT system.

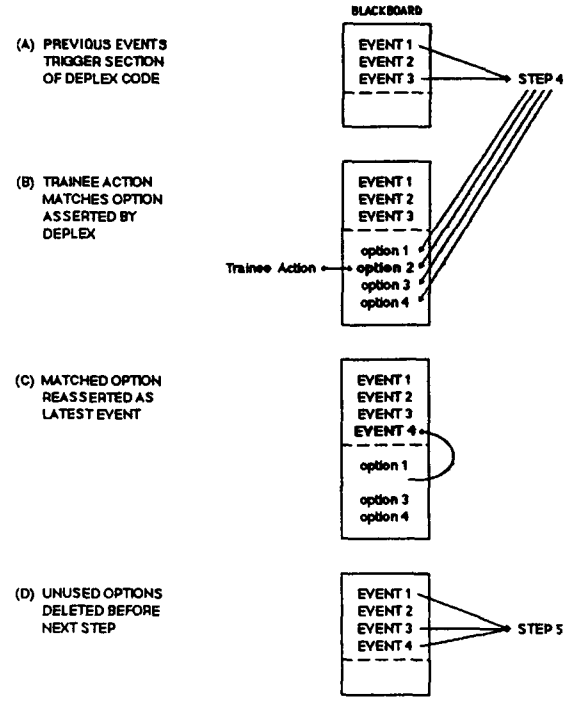


FIGURE 3 - DeplEx Operation, Illustrating Its Adaptability to Valid, User-Selected Alternatives

Training Session Manager

The training session manager is dedicated principally to error-handling. Its rules compare the assertions of DeplEx with those of the trainee to detect errors. Subsequently, DeplEx asserts facts that allow the TSM to write appropriate error messages to the trainee through the user interface. In addition, TSM is sensitive to the skill level of the trainee as represented by the trainee model. As a result, the detail and "tone" of error messages is chosen to match the current trainee. For example, an error made by a first time user of the training system may require a verbose explanation so that the system can be certain the trainee will have all of the knowledge and concepts needed to proceed. On the other hand, an experienced trainee may have momentarily forgotten a particular procedure or may have "lost his place". In this latter case a terse error message would be adequate to allow the trainee to resume the exercise. The TSM also encodes all trainee actions, both correct and incorrect, and passes them to the trainee model.

Trainee Model

Successful intelligent tutors incorporate student models to aid in error diagnosis and to guide the student's progress through the tutor's curriculum[9]. The trainee model in the PD/ICAT system stores assertions made by the TSM as a result of trainee actions. Thus, at its most fundamental level, the trainee model contains, for the current session, a complete record of the correct and incorrect actions taken by the trainee. At the conclusion of each training session, the model updates a training summary which contains information about the trainee's progress such as a skill level designator, number of sessions completed, number of errors made (by error type and session), and the time taken to complete sessions. After completing a session, the trainee can obtain a report of that session which contains a comprehensive list of correct and incorrect actions together with an evaluative commentary. A supervisor can access each trainee's model to obtain this same report or to obtain summary data, at a higher level, on the trainee's progress. Finally, the training scenario generator uses the trainee model to produce new training exercises.

Training Scenario Generator

The training scenario generator relies upon a database of task "problems" to structure unique exercises for a trainee each time he interacts with the system. The initial exercises provided to a new trainee are based on variants of a purely nominal PAM deploy with no time constraints, distractions or "problems". Once the trainee has demonstrated an acceptable level of competence with the nominal deploy, the generator draws upon its database to insert selected problems into the training environment (e.g., a propellant leak which renders the thrusters used for the nominal separation maneuver inoperable and requires the FDO to utilize a more complicated process for computing the maneuver). In addition, time constraints are "tightened" as the trainee gains more experience and distractions, in the form of requests for information from other MCC personnel, are presented at "inconvenient" points during the task. The generator also examines the trainee model for particular types of errors committed by the trainee in previous (and the current) sessions. The trainee is then given the opportunity to demonstrate that he will not make that error again. Ultimately, the trainee is presented with exercises which embody the most difficult problems together with time constraints and distractions comparable to those encountered during integrated simulations or actual missions.

SYSTEM INTEGRATION

The PD/ICAT system is currently operational on a Symbolics 3600 series Lisp machine. The user interface and trainee model are written in common Lisp while the rules of DeplEx, TSM, and the training scenario generator are written in ART 3.0. The system will ultimately be delivered to MOD in a Unix workstation environment. To accomplish this delivery, the ART rules were written to facilitate translation into CLIPS[10] and the Lisp code will be converted into C. It is uncertain, until the exact delivery environment is specified, how well the user interface can be ported.

CONCLUSIONS

The PD/ICAT system has, so far, proven to be a potentially valuable addition to the training tools available for training Flight Dynamics Officers in shuttle ground control. The authors are convinced that the basic structure of PD/ICAT can be extended to form a general architecture for intelligent training systems for training flight controllers and crew members in the performance of complex, mission-critical tasks. It may ultimately be effective in training personnel for a wide variety of tasks in governmental, academic, and industrial settings.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the invaluable contributions of expertise from three FDOs: Capt. Wes Jones, USAF; Major Doug Rask, USAF, and Kerry Soileau. Various students assisted with the knowledge engineering and coding of portions of the user interface and TSM: Tom Blinn, Joe Franz, Bebe Ly, Wayne Parrott, and Chou Pham. Finally, the encouragement and guidance of Chirold Epp (Head, ODS) and Bob Savely (Head, AIS) are gratefully acknowledged. Financial support for this endeavor has been provided by the Mission

Planning and Analysis Division, NASA/Johnson Space Center and (for RBL) by an American Society for Engineering Education/NASA Summer Faculty Fellowship.

REFERENCES

1. See, for example, Sleeman, D. and Brown, J.S. (eds.), Intelligent Tutoring Systems (London: Academic Press, 1982) and Yazdani, M. "Intelligent Tutoring Systems Survey," Artificial Intelligence Review **1**, 43 (1986).

2. Carbonell, J.R. "AI in CAI: An Artificial Intelligence Approach to CAI," IEEE Transactions on Man-Machine Systems **11**(4), 190 (1970).

3. Hartley, J.R. and Sleeman, D.H., "Towards Intelligent Teaching Systems," International Journal of Man-Machine Studies **5**, 215 (1973).

4. Yazdani, M. "Intelligent Tutoring Systems Survey," Artificial Intelligence Review **1**, 43 (1986).

5. Brown, J.S., Burton, R.R., and de Kleer, J., "Pedagogical, Natural Language and Knowledge Engineering Techniques in SOPHIE I, II, and III," in Sleeman, D. and Brown, J.S., (eds.), Intelligent Tutoring Systems (London: Academic Press, 1982), p. 227.

6. Johnson, W.L. and Soloway, E. "PROUST, " Byte **10** (4), 179 (April, 1985).

7. Anderson, J.R., Boyle, C.F., and Reiser, B.J., "Intelligent Tutoring Systems," Science **228**, 456 (1985) and Anderson, J.R. and Reiser, B.J., "The LISP Tutor," Byte **10**(4), 159 (April, 1985).

8. Harmon, P. "Intelligent Job Aids: How AI Will Chang Training in the Next Five Years," in Kearsley, G., ed., Artificial Intelligence and Instruction: Applications and Methods (Reading, MA: Addison Wesley Publishing Co., 1987).

9. See, for example, a number of papers on student models in Sleeman, D. and Brown, J.S., (eds.), Intelligent Tutoring Systems (London: Academic Press, 1982)

10. "CLIPS" is an acronym for "C-Language Integrated Production System" and was developed by the Artificial Intelligence Section, Mail Code FM72, NASA/Johnson Space Center, Houston, TX 77058. Its advantages as a delivery vehicle for expert systems are discussed in Giarratano, J., Culbert, C., Riley, G., and Savely, R.T., "A Solution of the Expert System Delivery Problem," submitted for publication in IEEE Expert. For additional information on CLIPS, write to the AI Section at NASA/JSC.

ACRONYMS

AIS	Artificial Intelligence Section
DeplEx	Deploy Expert
FDO(s)	Flight Dynamics Officer(s)
MCC	Mission Control Center
MOD	Mission Operations Directorate
ODS	Orbit Design Section
OJT	on-the-job training
PAM	Payload-Assist Module
PD/ICAT	Payload-Assist Module Deploy/Intelligent Computer-Aided Training
TSM	Training Session Manager

TUTORING ELECTRONIC TROUBLESHOOTING IN A SIMULATED MAINTENANCE WORK ENVIRONMENT

Sherrie P. Gott, PhD
Air Force Human Resources Laboratory
Manpower and Personnel Division
Brooks AFB TX 78235-5601

ABSTRACT

Expert performances on authentic technical problems such as electronic fault isolation are being captured in "real time" to provide the basis for a new generation of Air Force training systems. Experts (and novices) in dozens of maintenance jobs in electronic and electro-mechanical domains are being studied with a hybrid knowledge engineering-cognitive task analysis methodology. A primary goal is to establish what humans really need to know and how they use their knowledge when they problem solve in complex workcenters that are saturated with "smart" machines. The cornerstone of the method is an expert problem solving dyad. One expert poses a problem and simulates equipment responses to a second expert who attempts to isolate the fault conceived by the first expert. Engineering expert knowledge in this fashion situates skill in the actual problem context and thus highlights the conditionalized character of expert knowledge. This is in contrast to representation techniques that yield decontextualized (and perhaps nonessential) declarative knowledge through interrogation of a single expert. A series of intelligent tutoring systems--or intelligent maintenance simulators--is being developed based on expert and novice problem solving data of this type. The training systems rest on the same problem-based cornerstone. A graded series of authentic troubleshooting problems provides the curriculum, and adaptive instructional treatments foster active learning in trainees who engage in extensive fault isolation practice and thus in conditionalizing what they know. A proof of concept training study involving human tutoring was conducted as a precursor to the computer tutors to assess this integrated, problem-based approach to task analysis and instruction. Statistically significant improvements in apprentice technicians' troubleshooting efficiency were achieved after approximately six hours of training.

INTRODUCTION

Both military and industrial work environments have grown steadily in complexity in recent decades as technologies, particularly electronics related, have advanced at staggering rates. Today's workers find themselves in contexts where interacting with complex machines is the rule. And yet, the nature of intelligent performance in such machine interactions is not well understood. In addition, beliefs that cognitive demands on humans have diminished with the proliferation of so-called smart machines have diverted attention away from the human capabilities that are important for a high-tech workforce. Yet, it now seems clear that for the foreseeable future, machine diagnostic capabilities have definite limits. These limits in turn place a premium on the human expertise that is needed to pick up where the machines leave off. For example, the hit rate for some built-in diagnostics for the B1B is only 65 percent. Even with today's widely used maintenance aiding machines (many having expert system features), the ratio of maintenance hours to flying hours for the F-15 aircraft is 50:1 (Atkinson & Hiatt, 1985). In more general terms it has been estimated that as much as 90 percent of the life-cycle cost of a defense hardware system is the cost of maintaining it.

A large-scale research program is underway at the Air Force Human Resources Laboratory in direct response to this problem. The goals are to develop methods for representing human expertise on complex technical tasks so that training systems capable of meeting the demands of high-tech workcenters can be realized.

THE ENGINEERING OF CONDITIONALIZED KNOWLEDGE

The knowledge engineering approach in the Air Force Basic Job Skills (BJS) Research Program involves "real-time" problem solving, multiple stages and types of knowledge engineering inquiry, and a number of formats for knowledge representation. A framework has been adapted from knowledge engineering work in medical diagnosis to represent the mental events of troubleshooting as conditionalized knowledge (Clancey, 1985). In this framework, actions of the problem solver are recorded as discrete operations or procedures, e.g., tracing

schematics or measuring voltage. In addition, reasons or precursors for the actions are expressed as the goals or intents of the problem solver, and the interpretations of outcomes resulting from the actions are recorded as well. Finally, block diagram-like sketches of the equipment parts that are affected by the outcomes and actions are generated by the technician to illustrate the series of steps. Sequences of mental events such as these are called PARI structures (Precursor [to Action] - Action - Result - Interpretation). An example of PARI data for a single action node is shown in Table I.

Notice in this PARI example that the Action element is a familiar troubleshooting procedure, namely, taking a voltage measurement with a multimeter. The representational formalism of the PARI framework does more than reveal that a technician needs to know how to take a voltage measurement, however. What is also captured are the conditions that surround such a measurement operation, including the reasons behind the action (..."to see if the signal is good up to test package cable") and the interpretation of an expected voltage level (..."tells me...that part of stimulus path [upstream] is good"). In effect, the vital strategic processes of troubleshooting are made explicit with this representation scheme. The plan that produced the measurement operation becomes known. The technician's plan is to constrain the problem space by eliminating either the stimulus or measurement (return) portion of the signal path. It is precisely this kind of strategic skill that too often goes "untaught" in electronics training, in much the same way that strategic knowledge is frequently ignored in the teaching of mathematics (Greeno, 1978). When problem solving performances are captured in real time, it becomes possible to engineer strategic knowledge for input to instructional systems along with the more standard declarative knowledge. In this manner a skill such as taking a voltage reading is represented in terms of its ties to the conditions of use, just as it occurs in real world expert performances.

Representing skill components in this form offers considerable power to instruction, given that conditionalized knowledge is a recognized hallmark of expertise. Conversely, novices often display fragmented, unprincipled behavior that suggests weakness in the proceduralizing (or conditionalizing) of their skill components. In the present example, novices may know how to use a multimeter to take a voltage reading but often do not produce that action under the appropriate conditions. If produced, they often have difficulty interpreting the results of the action.

KNOWLEDGE ENGINEERING RESULTS

Approximately 15 technical experts and 200 less-than-expert technicians in four related AF electronics specialties have participated to date in knowledge engineering studies similar to those described above as part of the Basic Job Skills Research Program. On the basis of these studies, a meaningful superstructure for organizing troubleshooting performance data has been developed. It consists of three components, one of which is strategic knowledge as previewed above. The three interacting components are (1) system knowledge or the equipment device models experts use in problem solving (e.g., system knowledge regarding the stimulus or measurement functionalities of the equipment); (2) troubleshooting procedures or operations performed on the system; and (3) strategic knowledge, which includes (a) strategic decision factors that involve fault probabilities and efficiency estimates and (b) a top-level plan or strategy that is responsible for the orchestration of skill components in task execution. The orchestration occurs as the Strategy component, which sits on top of the Procedures and System Knowledge components, deploys pieces of knowledge and procedural subroutines as needed and as driven by the decision factors (Figure 1).

The System Knowledge component of the architecture deserves special attention for several reasons. First, it provides the dominant organizing principle for this cognitive skills architecture. It is the foundation to which the companion Procedures component in Figure 1 is attached. According to this view, a measurement or swapping operation is attached to a device model representation, since the purpose of the operation is viewed as adjusting the technician's present model of the device with new knowledge of faulty components. This "attachment" is part of the conditionalized character of expert knowledge. System Knowledge also feeds the strategic decision factors that underlie the Strategy component, since these factors involve system fault probabilities and efficiency estimates associated with operations on the system, e.g., it is judged time efficient by experts to run self diagnostics on some pieces of equipment but not others. Finally, System Knowledge influences the goal structure of the Strategy component in the sense that certain areas of the equipment are targeted before others (again due to fault probabilities and efficiency considerations).

The second reason why System Knowledge merits special attention here is because the curriculum content for the intelligent tutor described in the next section is directly influenced by the different system perspectives of expert troubleshooters. In the course of the knowledge engineering studies conducted to date in the BJS project, it has become clear that experts' decision making during troubleshooting is partially driven by system schemas. The schemas represent a set of system-related questions that experts entertain at various stages in the fault isolation process (Collins, 1987). They include the following:

-Is the system fail a glitch, an intermittent fail, or a hard fail?
-In which large functional area of the equipment--i.e., Line Replaceable Unit (LRU),

Test Package, or Test Station--is the fault located?
-Is the problem a power-related fail?
-Is the problem a stimulus or measurement problem?
-Is the problem a signal or data flow problem?
-Do the symptoms indicate the fault is in a device or in the connections between devices?

These questions can be viewed as the major parses the expert makes of the fault isolation space in which he/she works. Three of these parses have provided the framework for the troubleshooting problems that comprise the instructional content for the avionics intelligent tutor to be described next.

A SIMULATED MAINTENANCE WORK ENVIRONMENT

An intelligent maintenance practice environment for F15 integrated avionics technicians has been developed by researchers at the University of Pittsburgh's Learning R&D Center in collaboration with AF technical experts (Lesgold, 1987). The tutor is based on results from cognitive analyses of expert and novice AF technicians using the knowledge engineering methods referenced above. The analyses have provided three general types of input to the intelligent tutoring system: detailed characterizations of expert performance which are the targets for instruction (expressed in terms of the cognitive skills architecture of Figure 1); a framework for the design of the troubleshooting curriculum based on three parses experts make of the problem space in this domain; and guidelines for the instructional treatment based on expert-novice differences as well as on present impediments to apprenticeship learning in the workplace.

Expert Parses

Two central system schemas that experts activate as they navigate and parse problem spaces in this domain have provided the design framework for the maintenance tutor's problem set. These schemas represent two system perspectives experts' invoke, depending upon the conditions of the problem. The first concerns the major functionalities of the equipment, namely, stimulus and measurement functions. Recall that in the example reported in Table I the expert both explains his action and interprets the system's response to the action in terms of the stimulus portion of the equipment. More specifically, the procedure (action) used allows him/her to achieve the goal of verifying that a major functional area of the equipment is operating properly.

The stimulus-measurement functionalities of this equipment are illustrated in Figure 2. This is an abstracted characterization of the system's signal path. As shown, the signal originates in the stimulus drawer of an avionics test station, travels through the station's switching drawer (S/C) which performs signal switching functions, and through an interface test package to an aircraft line replaceable unit (LRU) which is being tested for a malfunction. It returns through the interface package to a measurement source in the test station.

Problems in the tutor curriculum represent faults at varying levels of difficulty in the stimulus and measurement routing of the equipment. Trainees will have modeled for them how an expert uses this perspective to isolate various faults. They will then have extensive opportunities to solve problems--with the assistance of a hint-giving coach--so that system functionality knowledge is tied to problem solving conditions. This kind of learning environment is in contrast to instruction where system knowledge would be taught as declarative facts detached from the conditions of use, or where measurement procedures would be taught in isolation from the system and the fault isolation context.

Results of our knowledge engineering work plus input from the dominant theory of skill acquisition in psychology today (Anderson, 1982) have shaped this instructional approach. First, our results have indicated that a principal form of the conditionalized knowledge of experts in this domain is the coupling of conceptual system knowledge (e.g., the stimulus-measurement functionality) with procedural and strategic components. This results in experts' investigating their equipment with specific intents and particularized procedures. In other words their system knowledge is not detached and inert, but rather is tightly interwoven with problem solving actions that are produced by strategic plans in response to certain malfunction conditions. Presently, in the Air Force this form of conditionalized knowledge results only after many years of experience, as would be predicted by Anderson's theory. A principal goal of the BJS maintenance tutor is to speed up that conditionalizing process.

The second system perspective or schema used to shape the tutor's problem set is signal flow vs data flow. Experts also view the equipment (and thus represent faults) in terms of these two interrelated system properties. In short, this schema involves knowledge that both an electronic signal and instructions (control data) to the equipment for handling the signal move through the system. Faults can occur with respect to either property. Accordingly, signal flow and control data flow problems are incorporated in the tutor at varying levels of difficulty.

Finally, a third schema, namely, the macro level functional representation of the equipment (LRU vs Test Package vs Test Station) has guided problem development. This schema is integrally tied to experts' strategic planning knowledge in the sense that they typically plan their moves through the problem space so that they systematically rule out the LRU before moving their focus to either the Test Package or Test Station. Trainees will make decisions within the tutor environment to pursue either an LRU Plan, a Test Package Plan, or a Test Station Plan.

In summary, the development of the Air Force avionics tutor illustrates that knowledge engineering can usefully feed instructional design as well as provide the more standard type of input, i.e., the expert knowledge base. Further, dynamic, problem-based knowledge engineering allows for the representation of conditionalized knowledge so that the most critical stage of skill acquisition can be targeted by instruction. That is the stage at which knowledge becomes tied to conditions of use. The avionics maintenance tutoring system based on this approach will be discussed in more detail in the next section.

An AI Instructional Application.

The BJS tutoring system that has resulted from the expert dyad approach to knowledge engineering is an interesting AI application in the sense that it embodies minimally deep intelligence. It avoids complete qualitative physics of the work environment as well as a complete computer representation of expertise (Lesgold, 1987). In short, there is neither a fully articulate expert nor a runnable equipment simulation. Later tutors in the BJS series will have these features; however, this initial system is of special interest in its own right. Its development is much less resource intensive than that of deep intelligence tutors, and it has received an enthusiastic reception from technical experts at the three operational sites where it will soon be tested. If the evaluation results reveal troubleshooting performance gains in accordance with the predictions of field personnel, this form of intelligent tutoring system represents a quite feasible prototype that can immediately generalize to other troubleshooting domains.

A rigorous evaluation study will accompany the intervention in order to formally assess its effectiveness. A controlled experiment will permit the determination of how much on-the-job experience is replaced by the 30 to 50 hours of tutor instruction. In addition, performance of individual technicians and the shop-level productivity of the three F15 workcenters will be tracked longitudinally to ascertain the long-term impact of the instruction.

As a precursor to this series of BJS intelligent tutoring systems, a training study involving a human tutor (versus a computer coach) was conducted in a related F15 integrated avionics domain. One goal was to test the concept of basing instruction on representations of conditionalized expert knowledge. The treatment involved the posing of authentic troubleshooting problems similar to those generated in a BJS knowledge engineering study as described above. The expert-like skills targeted for enhancement were particular instantiations of the cognitive skills architecture (Figure 1). The system knowledge of interest was the abstracted signal path shown in Figure 2, plus several layers of elaborated system knowledge. The procedures of interest were three methods for investigating the equipment that ranged from rudimentary to advanced:

(1) swapping equipment components
(2) using self-diagnostics to test system integrity
(3) measuring device and circuit functionality.

Increasingly complex system and strategic knowledge are associated with increasingly sophisticated methods.

During three to five hours of individual instruction over a period of three days, seven technicians were tutored. They were presented a troubleshooting scenario and then probed regarding what they would do to isolate the fault (Actions), why they would take the particular action (Precursors), and what the outcome (Result) of the action meant to them (Interpretation). In effect, technicians were instructed to generate PARI records (see Table 1) including the associated device model sketches. The human tutor gave feedback to their stated Precursors, Actions, and Interpretations in the form of hints intended to move them toward more expert-like performances.

To evaluate their learning, they were given both an end-of-training problem-based test as well as a delayed posttest after the weekend. The tests were authentic troubleshooting scenarios belonging to the same class and difficulty of problems on which they had been tutored. Their progress was scored both in terms of the sufficiency of their operations--that is, whether they sufficiently investigated all suspect pieces of the equipment--and the efficiency of their moves--that is, whether they efficiently conserved time and equipment resources.

Results showed statistically significant improvements in both areas, with particularly dramatic gains in efficiency. Mean scores are plotted in Figure 3. The group's Sufficiency in examining all suspect parts of the equipment improved from a pretest mean value of 84 (range = 60 to 95) to a posttest mean of 100. The dalayed posttest mean was also 100, indicating the improvement was retained over the weekend. The group's Efficiency in fault isolation improved over twofold. The mean pretest value was 37 (range = 24 to 52); the initial posttest mean was 92 (range = 81 to 100); and the delayed posttest mean was 93 (range = 81 to 97).

Pedagogically, this human tutor training study was based on the same instructional principles that underpin the computer-based avionics tutor. Technicians were afforded extensive practice in fault-isolation; they were required to articulate and focus on their reasons and their interpretations of various troubleshooting moves; they were aided by a human tutor who, principally through Socratic dialogue, challenged them to reflect on what they did in terms of expert standards of thoroughness and efficiency. The technicians later attributed the gains they made to the opportunities they

had to practice fault isolation procedures intensively and to solve problems independently. They reported that recording and reflecting on their actions and reasons was helpful and that they profitted from the hints and consistent feedback. This successful study is viewed as empirical support for the effectiveness of skill acquisition treatments that focus on the conditionalizing of knowledge in intelligent learning environments. External support in the form of the PARI records and the human tutor's feedback appeared to play a central role in learning. Finally, the instruction was realizable because of the knowledge engineering input that revealed the processes by which experts conditionalize what they know.

REFERENCES

1. Anderson, J.R., Acquisition of cognitive skill, PSYCHOLOGICAL REVIEW, Vol 89, 1982, pp 369-406.

2. Atkinson, R. & Hiatt, F., Nation's high-tech weaponry requires high-priced repairs, THE WASHINGTON POST, Aug 18, 1985, pp 1, 8.

3. Clancey, William, Acquiring, representing, and evaluating a competence model of diagnostic strategy, REPORT NO. STAN-85-1067, Department of Computer Science, Stanford University, Stanford, CA, August, 1985.

4. Collins, Allan, Users of task analysis, Air Force Basic Job Skills Progress Review, San Antonio, Tx, June 1987.

5. Greeno, James G., A study of problem solving, ADVANCES IN INSTRUCTIONAL PSYCHOLOGY, Vol 1, Lawrence Erlbaum Associates, Hillsdale, NJ, 1978, pp 13-75.

6. Lesgold, Alan, The manual avionics tutor, Air Force Basic Job Skills Progress Review, San Antonio, Tx, June 1987.

Table 1: PARI DATA

Precursor: Want to see if the stimulus signal is good up to test package cable

Action: Measure signal at J14-28 with multimeter

Result: 28 volts

Interpretation: This is expected reading; this tells me that the stimulus is getting from the test station through the cable, so that part of the stimulus path is good

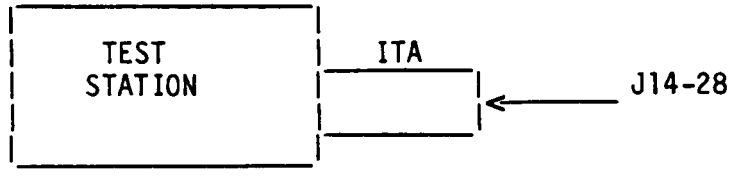


Figure 1.

Cognitive Skills Architecture

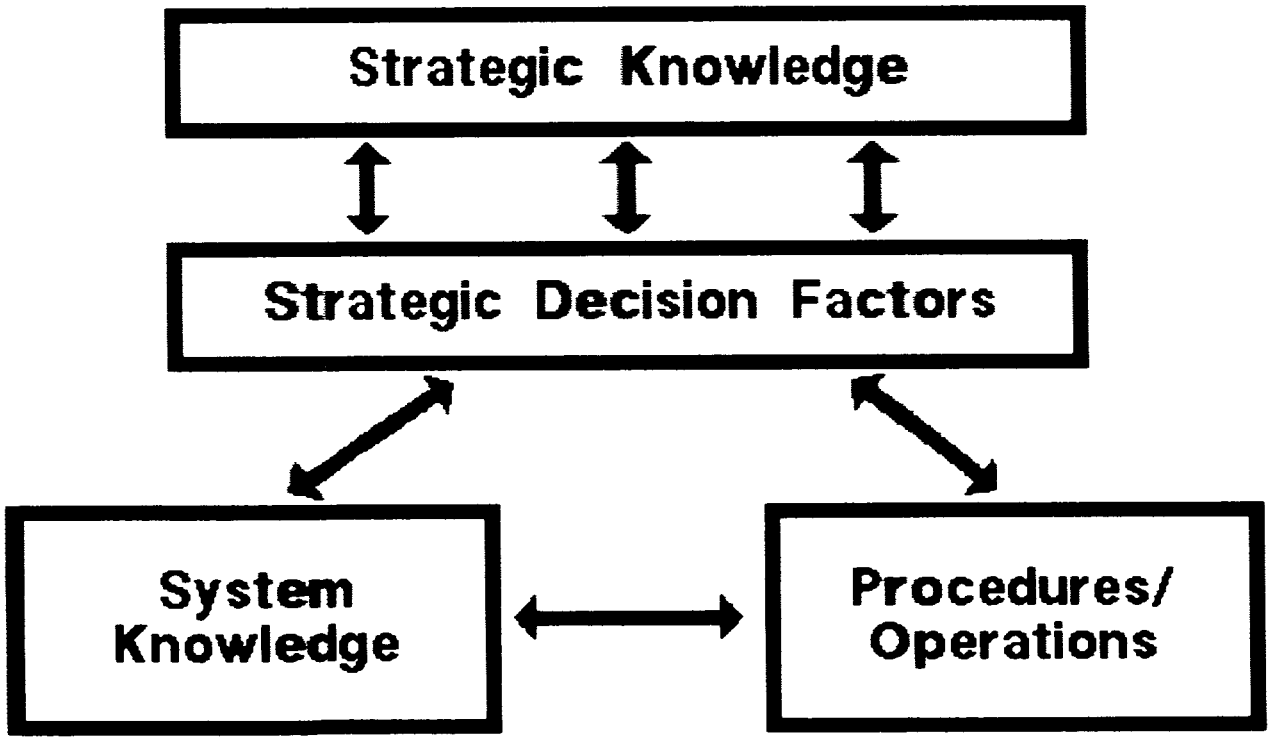


Figure 2.

Avionics Equipment Configuration (Signal Path)

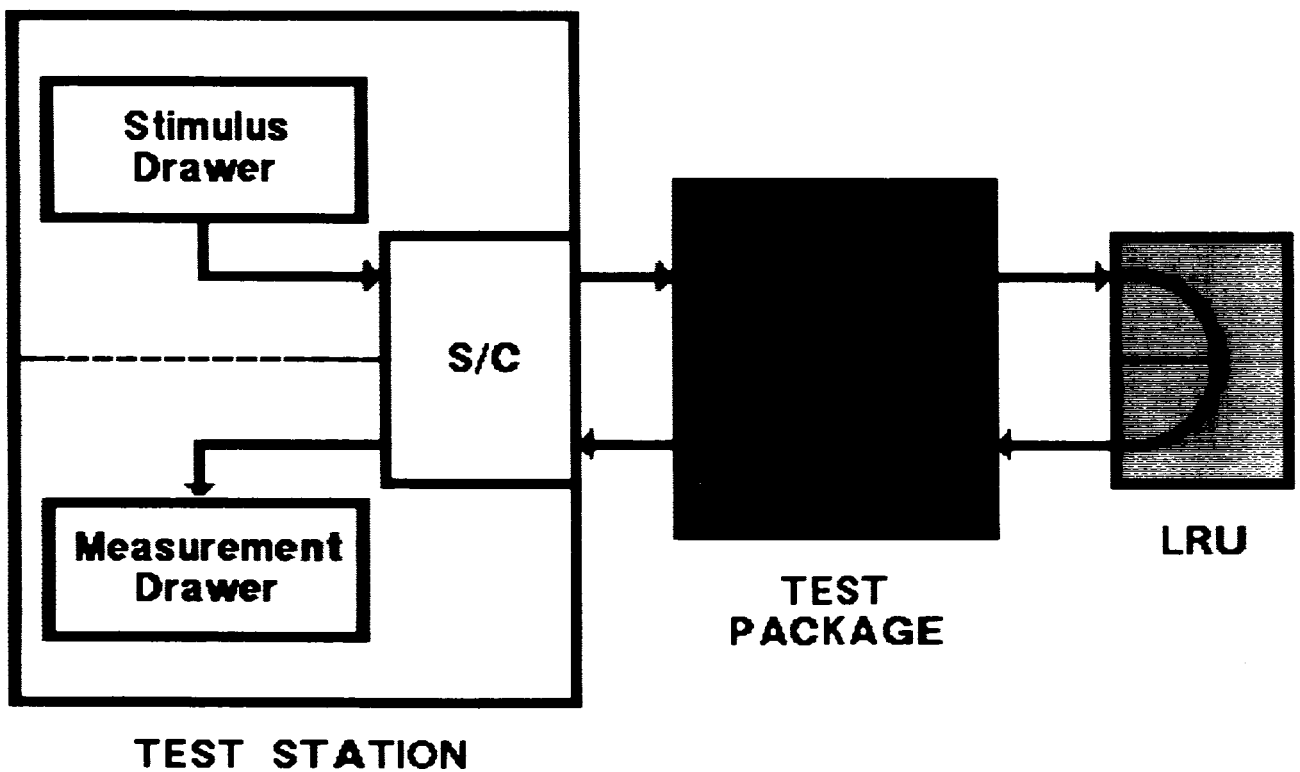
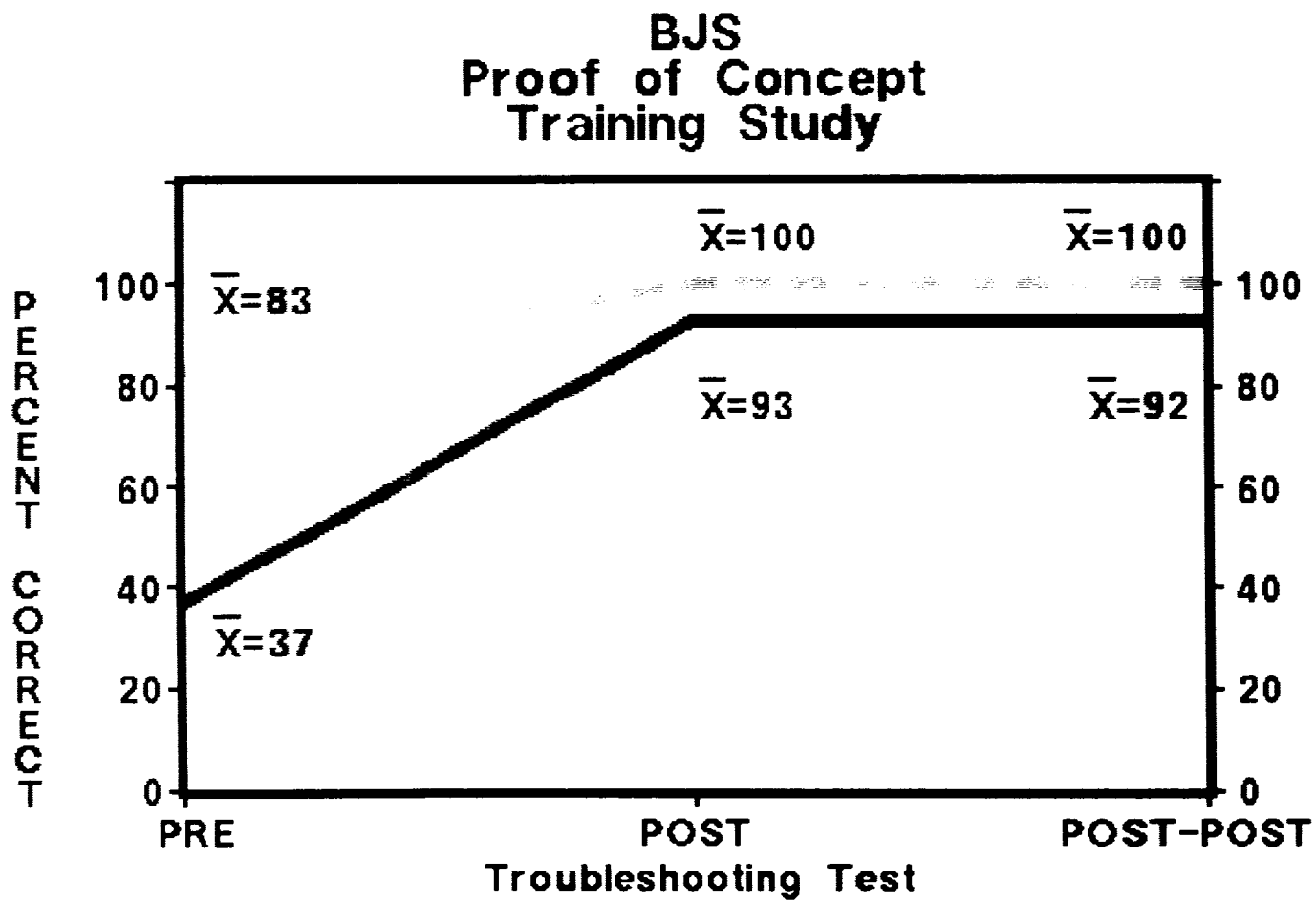


Figure 3.


BJS
Proof of Concept
Training Study
PERCENT CORRECT
100
80
60
40
20
0
X=83
X=37
X=100
X=93
X=100
X=92
PRE
POST
POST-POST
Troubleshooting Test
Troubleshooting Efficiency p ◀ .01

INTELLIGENT TUTORING SYSTEMS AS TOOLS FOR INVESTIGATING INDIVIDUAL DIFFERENCES IN LEARNING

Valerie J. Shute

AFHRL/MOE

Brooks AFB, TX

INTRODUCTION

The purpose of the Learning Abilities Measurement Program (LAMP) is to conduct basic research on the nature of human learning and performance. The ultimate goal of this research is to build an improved model-based selection and classification system for the United States Air Force. During the first few years of the program, and continuing through to the present, researchers are developing innovative approaches to ability testing (Kyllonen & Christal, in press). In conjunction with this framework, new kinds of computerized ability tests have been developed (Fairbank, Tirre & Anderson, 1987; Tirre & Rancourt, 1986; Woltz, 1986; Woltz, 1987). LAMP examines individual differences in learning abilities, seeking answers to the following questions:

1. Why do some people learn more and better than others?
2. Are there basic cognitive processes applicable across tasks and domains that are predictive of successful performance, or are the behaviors in question more involved (e.g., complex problem solving behaviors)?
3. Which of these processes or learning abilities are domain specific and which generalize across subject areas?

We have used some simple learning tasks to determine the elementary cognitive processes involved in learning abilities such as: Information processing speed, prior knowledge, and working memory capacity (size and activation level). To test the extent of differential learning abilities based on these rudimentary processes, we need to examine learning in progress in complex environments, like intelligent tutoring systems (ITS's), which reflect 'real world' performance rather than artificial laboratory tasks (like paired associate or rule learning) which often do not generalize to the real world. There are basically two categories of related activities in this research program. First, we are concerned with individual differences in learners' knowledge and skills. In this regard, our aim is to identify more efficient and precise methods of individual assessment. Second, we are interested in validating models of ability organization by (a) estimating individual skill and knowledge levels, (b) estimating individual proficiency levels on various learning tasks, and (c) relating the two sets of

estimates using exploratory and confirmatory mathematical modeling techniques such as regression analysis and factor analysis.

We have contracted to have three complex, long-term learning tasks (i.e., ITS's) developed. The three tutors teach electronics trouble-shooting, flight engineering, and Pascal programming. These ITS's, each requiring about seven days for completion of the curricula, are, for the most part, based on instruction and test modules from operational Air Force training courses. We are using another ITS for basic research that has a more discovery-oriented learning approach involving principles of microeconomics. In addition to encompassing economic concepts, "Smithtown" (Shute & Glaser, in press) assists the learner in becoming more methodical and 'scientific' in their pursuit of knowledge obtainable from the system. Learning parameters estimated from performance in all of the ITS courses serve as intermediate criteria against which measures of knowledge and skill acquisition will be evaluated. The success of LAMP will ultimately not depend on whether we can predict who is more adept at acquiring simple facts and rules from the short-term tasks, but on whether we can predict who will acquire more permanent and complex sets of skills characteristic of effective operational job performance. Thus, our main concern is with validating models of cognitive skills against performance in complex learning environments.

INTELLIGENT TUTORING SYSTEMS AS RESEARCH TOOLS

We are using intelligent tutoring systems as experimental vehicles to determine the set of predictor variables effective in predicting understanding and learning in complex environments. In any intelligent tutoring system, the learner interacts with a computer program to acquire new information and exercise newly acquired skills. The program presents problems to the student in an adaptive fashion by taking into account both the structure of the concepts from a subject domain (i.e., the curriculum) and the individual learner's current knowledge and understanding of that subject domain (i.e., the student model). Such programs can provide a rich trace of the individual's learning performance, states of knowledge, and rate of progress through the curriculum.

With each ITS, analyzed separately, we begin our research by delineating a large set of knowledge and performance indicators for a given tutor, and then relate these behaviors back to the individual cognitive processes as well as to objective measures of learning (see Shute, Glaser & Raghavan, 1987). To illustrate, the Pascal programming tutor has general purpose data analysis tools which let us specify exactly which performance or knowledge indicators we want output from the extensive student history list. Any action or sequence of actions can be specified as an 'event'. For example, we can set up any event where A, B, C, and D are particular actions:

E1: (The student does A and B then (C or D)), or

E2: (The student does A or B and (not C)).

The system computes how many times this sequence

occurred, the errors in performance on this event, the number of intervening events between subsequent occurrences of this event, and so on. We can specify very simple actions as events (e.g., The student does A) to more complex series of actions to see how the student progresses over time.

Thus, the ITS research can serve as an ideal source of intermediate learning criteria against which conventional and experimental aptitude tests can be validated. For instance, we can determine whether processing speed or working memory capacity is more important in ascertainining who will be successful in learning Pascal programming, or perhaps it is determined more from higher level "planning" types of behaviors (Anderson, 1987).

Intelligent tutoring systems provide us with controlled, rich environments to investigate individual differences in the acquisition of knowledge and skills. In addition, they provide us with comprehensive traces of all student actions involved in the learning of a given subject matter. The tutors consist of complex, real world type environments, allowing us to abstract so much more information about learning than is possible from static paper and pencil tests.

One important consideration in using ITS's is that some computer learning environments are clearly not suitable for all types of subject populations (e.g., discovery worlds). To illustrate, two groups of subjects have been run on Smithtown, the intelligent discovery world environment mentioned earlier that embraces the laws of supply and demand in a hypothetical marketplace (Shute, et al., 1987). Variables such as the population or weather can be manipulated, the results noted, and principles and laws induced from the findings. University students were, for the most part, very positive about it, and said things like, "What a fun game... I learned a lot about economics". On the other hand, basic Air Force recruits (N= 530) were mostly bewildered by the environments, typically complaining that, "I've been lost the whole time!" and constantly asking, "What should I be doing?" This is not surprising given the different structures and emphases of the two settings (i.e., academic vs. military contexts). Given this finding, it would be a relatively easy adjustment to make the environment more structured for those individuals requiring more of a framework for learning.

FUTURE DIRECTIONS

The tutors will allow us to predict various properties of the acquisition process for different Air Force related knowledge and skills from measures developed within the LAMP project. In addition, the measurements of the course of acquisition and its variability across individuals can be used to shape and confirm extensions to current theories of knowledge and skill acquisition as well as to document the critical

individual differences that arise during this process.

Three types of learning progress indices will be used to measure different aspects of the course of learning. These include measurements of an individual's rate, quality and durability of learning. Specifically, the three measures are: performance criteria (e.g., the number of times tutor advice was required), categories of acquisition trajectories (e.g., change in performance speed as a function of practice) and process measures (e.g., plans that a subject develops for solving a problem).

Currently, we are contracting to have intelligent tutoring systems developed on PC AT-compatible machines (mini-tutors). These systems will consist of job skills extracted from the larger tutors such as: declarative knowledge acquisition of electrical circuits, procedural knowledge of graph interpretation, and so on. These mini-tutors, lasting only 1-3.5 hours instead of 7 days, will allow us to refine hypotheses and measures with the mini-tutors criteria before actually testing them out on a large scale. We will be able to more precisely analyze the learning of specific productions underlying complex skills. These systems will also be considerably more cost effective than the larger tutors in terms of subject hours and hardware costs.

SUMMARY

Assessing individual differences in cognitive processes using experimental learning tasks is just one aspect of the LAMP effort. Another, and more exciting feature, is the mechanism we are concurrently using to extend our findings from the simpler, often contrived environments to more complex, real world types of environments via intelligent tutoring systems. Thus, the LAMP program and its use of ITS's as experimental testbeds represents an innovative twist on an old stream of research: investigating individual differences in learning as it relates to successful on-the-job learning and performance. Our ITS's, as intermediate criteria, will enable us to assess the same kind of learning as occurs in real world tasks, but in controlled environments with rich traces of the active, ongoing learning processes. This can be contrasted to the paper and pencil tests historically (as well as currently) used by the Air Force to assess learning and abilities. These tests only provide post hoc, static measures or depictions of learning, with many unanswered questions regarding the route to that end. The ITS's let us look at a range of individual differences in learning from simple cognitive processes such as information processing speed (and its various components, such as encoding, comparing, choosing, retrieving, attention shifting and memory searching) to more complex problem solving processes such as means ends analysis and hypothesis generation and testing.

References

Anderson, J.R., "Using intelligent tutoring systems to analyze data", Paper presented at the COGNITIVE SCIENCE SOCIETY CONFERENCE, Seattle WA, July 1987.

Fairbank, B.A. Jr., Tirre, W.C. and Anderson, N.S., "Measures of thirty cognitive tasks: Analysis of reliabilities, intercorrelations, and correlations with aptitude battery scores", Paper presented at the NATO ADVANCED STUDY INSTITUTE IN COGNITION AND MOTIVATION, Athens, Greece, December 1984.

Kyllonen, P.C. & Christal, R.E., "Cognitive modeling of learning abilities: A status report of LAMP", In R. Dillon & J.W. Pellegrino, ADVANCES IN TESTING AND TRAINING. New York: Academic Press, in press.

Shute, V.J. & Glaser, R., "An intelligent tutoring system for exploring principles of economics", In R.E. Snow & D. Wiley (Eds.), STRAIGHT THINKING, San Francisco: Jossey Bass, in press.

Shute, V.J., Glaser, R., Raghavan, K., "Discovery and inference in an exploratory laboratory", In P.L. Ackerman, R.J. Sternberg, & R. Glaser (Eds.), LEARNING AND INDIVIDUAL DIFFERENCES, San Francisco: Freeman, 1987.

Tirre, W.C. & Rancourt,C.R., "Individual differences in learning by accretion: It all begins with comprehension", Paper presented in R.E. Snow (Chair), Determinants of individual differences in learning, AMERICAN EDUCATIONAL RESEARCH ASSOCIATION, San Francisco, CA, April 1986.

Woltz, D.J., "Activation and decay of word meanings: An individual differences investigation of working memory", Unpublished manuscript, AFHRL, Brooks AFB, TX, 1987.

Woltz, D.J., "The role of working memory in associative learning", Paper presented in R.E. Snow (Chair), Determinants of individual differences in learning, AMERICAN EDUCATIONAL RESEARCH ASSOCIATION, San Francisco, CA, April 1986.

AN INTELLIGENT TUTOR FOR THE SPACE DOMAIN

*Dr. Kathleen Swigger**
Harry Loveland

North Texas State University
Computer Sciences Department
Box 13886
Denton, Texas 76203

ABSTRACT

This paper describes an intelligent tutoring system for the space domain. The system was developed on a Xerox 1108 using LOOPS and provides an environment for discovering principles of ground tracks as a direct function of the orbital elements. This paper also looks at some of the more practical design and implementation issues associated with the development of intelligent tutoring systems. It attempts to offer some solutions to the problems and some suggestions for future research.

INTRODUCTION

The Intelligent Tutor for Ground Tracks (nicknamed OM) is designed to teach students how to "deduce" a satellite's orbital elements by looking at a graphic display of a satellite's ground track. In order to help the student understand these relationships, the system was given a special interface that allows student to freely investigate different options and "discover" relationships between various parameters. If, however, the student does not "discover" these principles and concepts, then OM intervenes and directs the student toward specific goals.

One of the basic missions for space operations personnel is the continuous monitoring of the exoatmospheric arena through ground and space surveillance. For example, NORAD, through its Space Defense Center, maintains a worldwide network that senses, tracks, and analyzes the characteristics of orbiting systems. In order to monitor and plan for satellite missions, space operations crews must be able to read and understand ground tracks. Ground tracks are two-dimensional displays that show the portion of the earth that a satellite covers in one orbit. The ground track is a direct function of the orbital elements, so proper understanding of these functions and their effect on the shape of the ground track is critical for anyone interested in satellite operations.

One way to teach students how to deduce orbital elements from a satellite's ground track is to present the various formulas that are used to compute the orbital elements and then show students how to apply these formulas to situation-specific tracks [Bates et al., 1971; Astronautics, 1985] In contrast to this approach, we discovered that experts store ground tracks as graphical representations, indexed by feature and shape. Based on previous experience, experts learn how to detect any features such as size, number of loops, direction, etc., and then use this information to "estimate" the orbital elements. In order to duplicate this process, we decided to build a qualitative model of how the expert predicts orbital elements, given specific shape descriptors, and then use this model as a basis for teaching students the effects of different orbital parameters on the shape of the ground track.

*This research was supported, in part, with a grant from the Office of Scientific Research which was conducted at the Air Force Human Resources Laboratory (AFHRL), San Antonio, Texas.

STUDENT/COMPUTER INTERACTION

As previously mentioned, the microworld for the Ground Track problem offers a number of online tools that permit students to discover relationships between orbital parameters and ground tracks. This environment consists of an elaborate ground track display (Figure 1) and a number of interactive tools designed to encourage systematic behaviors for investigating ground track related problems. The student initiates a discovery activity by changing one or more orbital parameters or changing the injection parameters. This task is accomplished by positioning the cursor over the individual parameters and pressing the left mouse button to increase the value or the middle button to decrease the value. The injection point is changed by positioning the cursor over a particular point on the map and pressing the left mouse button, which automatically sets both the longitude and latitude. A student can observe the results of these changes by selecting "Generate Ground Trace" from the main menu. After investigating the effects of changing different parameter values for different ground tracks, the student can advance to the Prediction window where he can make a hypothesis regarding the particular shape of a ground track.

In the Prediction portion of the program, the system displays a list of words that describe various features about ground tracks such as shape, size, and symmetry (Figure 2). From this list of descriptors, the student selects the words that "best" describe the current ground track under discussion. The student then tests his prediction by selecting this option from the menu and comparing the inputs to the Expert's conclusions. The student can then interrogate the Expert System by placing the cursor over any of the descriptors and pressing the left mouse button. A "why" pop-up menu appears on the screen which enables a student to receive an explanation of the Expert's reason for the correct descriptor. A student may also interrogate his own selections by placing the

cursor over his "input" selection and pressing the right button. In this instance, a "why not" pop-up menu appears and displays the reasons why a particular descriptor was an inappropriate selection. The student can continue in this manner until he understands the various relationships between the shape of a ground track and the different orbital parameters.

After making several successful predictions, the student enters an Orbit Prediction environment which is designed to check the student's predictive powers by asking him to perform a task in the reverse order of the one described above. The student is shown a specific type of ground track and asked to enter actual orbital descriptors of the ground track. If the student is successful, then he can continue to explore different types of ground tracks. If the student is unsuccessful, then he receives information about why his answers are incorrect.

TOOL DESCRIPTION

There are three major online tools that can be used by the student to gather information and to understand concepts and principles about ground tracks. These tools are a) a History tool that allows the students to overlay previously generated ground tracks and note relationships between parameters b) an Orbit window that displays a two-dimensional representation of the orbit (Figure 1); and c) a Definition/Example tool which displays factual information about different orbital parameters (Figure 1) and orbital descriptors.

The History tool is specifically designed to help students recognize relevant patterns between and among previously generated ground tracks. As the student generates various ground tracks, the system collects and stores each transaction. The student can retrieve any of this data by selecting the History option from the main menu. A list of the past twenty ground tracks appears on the screen from which the student can select one or more related ground tracks. The system then overlays the selected ground tracks onto a single map. Again, the student observes the results of this exercise.

For any given set of orbital parameters, the student can obtain a two-dimensional display which shows the position of the satellite in relationship to the earth. The student selects the option labelled Orbit window and gains immediate access to this particular display. The Orbit window is especially useful for demonstrating the relationship between the ground track and the actual orbit and for illustrating the effect of perigee on elliptical orbits.

The Definition/Example tool provides the student with the factual knowledge about the domain. A student can obtain definitions and examples for orbital parameters and the shape descriptors by simply placing the cursor over the keyword in question and pressing the right mouse button. A pop-up menu appears on the screen from which the student can select definitions, examples or explanations. The explanations for the orbital parameters are generated according to the context that they appear.

Thus by using the available tools, a student can obtain facts about the orbital world (through the Definition/Example tool), see relationships between different ground tracks (through the History window), and understand certain principles about satellite operations (through the Orbit window). A student has the option of using any of these tools at any time during the computer/student interaction. If, however, the student is not making sufficient progress, the system interrupts and directs the student to use a specific tool to achieve an objective.

DESIGN OF THE SYSTEM

Overview

Although the system is composed of five logical units (an Expert system, a Curriculum, a State Module, a Student Model, and a Coach), the Tutor is actually implemented as a series of LOOPS classes and objects. Thus, the Tutor's logical units do not necessarily correspond to specific programming segments. The Expert, for example, is implemented as a series of Shape Objects* which contain both the rules and inference procedures used to deduce shape descriptors from a set of orbital parameters. These Shape Objects also contain the major concepts associated with the ground track curriculum (the Curriculum). The State Module contains a list of appropriate behaviors for exploring the microworld. There is also a series of methods** used to evaluate the student's answer, analyze student errors, and update counters. The Student Model resides within the Expert Objects in the form of counters and threshold values which reflect the student's current state of knowledge of both ground tracks and effective tool use. Finally, there are a series of Coaching methods that tell the system when to intervene. The system makes its decision based on information regarding the student's current state of knowledge. A more detailed description of each logical unit is presented below.

The Expert

This module contains the rules and procedures used to deduce shape descriptors (e.g., closed-body, symmetrical, vertical; compressed, lean-right, hinge-symmetry, with loops) from a set of orbital parameters (eccentricity, period, semi-major axis, argument of periapsis, inclination). The Expert is invoked only when the student is making a shape or orbit prediction. The general problem solving strategy employed by the Expert is to determine a shape descriptor by examining a specific orbital element. If this fails, then the system looks at another shape descriptor and attempts to find its value, or looks at a combination of two or more orbital elements to see if the system can deduce a shape descriptor. For example, the Expert determines the symmetry shape goal by asking whether this is a circular orbit. If the orbit is classified as a circular orbit, then its eccentricity must be equal to zero. If the orbit is elliptical then its eccentricity is not equal to zero and the Expert must look at the orientation descriptor, which in turn must look at the argument of periapsis. In this manner, the Expert Module can determine a set of shape descriptors for a given set of orbital parameters (and vice versa). During the process of deducing shape descriptors, the Expert also determines the optimal "procedure" for deriving the shape descriptors. Thus both declarative and procedural knowledge is available to the rest of the tutor.

*Objects is a trademark for data types in the LOOPS programming environment, Xerox, Corp.

**Methods is a trademark for procedures in the LOOPS programming environment, Xerox, Corp.

At the implementation level, the Expert shape descriptors are organized as classes and sub-classes (Figure 3). The Expert operates by calling a "metal-rule" that sends a message to all the objects to test the rules associated with each of the objects and return the values from the rules that are true. Along with the Expert's If...then rules, each object contains the definition, explanation templates, examples, special counters indicating the number of times the student predicts the shape descriptors correctly and incorrectly, tutoring strategies, conflict resolution strategies, and special buggy rules. This particular data proved to be a very effective way of organizing the knowledge.

Another function of the Expert is to deduce parameter descriptors (such as a Circular, Synchronous orbit) at the same time that the system is deducing the shape descriptors. These parameter descriptors are used to determine the essential skills that are necessary to understand a given ground track. Since the rules for determining the Curriculum are used by the Expert rules, we now describe the organization of the Curriculum.

The Curriculum

Along with knowledge about shape descriptors for ground tracks, a student must also understand how this information relates to specific orbit types. For example, an orbit which has a semi- major axis equal to 42,250 kilometers is said to be in a synchronous orbit. This term applies to all ground tracks that have a semi-major axis equal to 42,250 kilometers, regardless of the numbers that might appear for the other orbital parameters. Thus it is important that students recognize the relationship between the specific domain knowledge and the qualitative model produced by the Expert. Therefore the System organizes this knowledge in the Orbit Objects (Figure 4) which contain the specific content that is used to categorize different orbit types. The knowledge stored in the Orbit Objects is then used (and deduced) by the Expert. For example, the Expert System determines whether an orbit is circular or elliptical as it deduces the symmetry goal. The knowledge about shapes and orbit types are an integral part of the Expert.

This particular organization also provides a very powerful tool for relating the content areas and for determining various levels of difficulty. For example, the rules that determine the shape descriptors associated with circular orbits tend to have fewer constraints attached to them, and also tend to be fired first, and, as a result, tend to be easier for the student to learn. The hierarchy of orbit types as represented in the Orbit Objects shows both the order that the knowledge should be learned and the relationships between the knowledge. This information can be used by to recommend easier problems whenever the student becomes confused.

The State Module

The State Module contains a list of goals and subgoals which presumably indicate acceptable procedures for exploring the microworld. As the student proceeds through each of the states, the tutor records his/her actions. The authors have hypothesized that a student indicates appropriate experimental behaviors if explores a microworld by generating ground traces. The student then moves on to "making predictions," followed by testing and validating tests, and then generalizing these principles. Each one of these states, in turn, has separate subgoals which may or may not be met. The Tutor uses the State Module in two ways. First, if the student is performing poorly, then the Tutor checks to see if the student has proceeded through each state in an appropriate manner. Second, the system uses the State Module to reflect different "instructional" strategies. For example, if the student is conducting experiments (as defined as "making predictions") then the system gives a higher status to using tools correctly. If the student is "testing," then OM will switch its strategy and try rules that check for skill deficiencies.

The Student Model

The Student Model in embedded within the Expert and Orbit shape Objects as a series of counters that reflect the student's current understanding of both the domain knowledge and investigative behaviors. Whenever the student tests a prediction, OM records a list of the rules that the student understands. The Student Model maintains a series of counters for each rule indicating the number of times a rule is used appropriately, inappropriately, or ignored (a "missed-opportunity" as defined in Carr and Goldstein, 1977). If the missed-opportunity counter exceeds the used-appropriate counter, then the Coach recommends intervention.

The system also records the number of times that an online tool is invoked. In addition to this counter, an effectiveness measure is maintained for the History tool, the Orbit window, and the Definition/Example tool. If the student demonstrates inefficient behavior as indicated by one of the effectiveness measures, then the system intervenes and offers advice.

The Coaching Strategy

OM also maintains a series of rules and procedures that direct the teaching portion of the Tutor. The Ground Track Microworld is designed for two major purposes: 1) to teach students about the relationships between/among orbital elements and ground tracks, and 2) to teach students how to use systematic behaviors to investigate this domain. Thus, the system intervenes when either one of these conditions is not satisfied. The system monitors the student's actions and determines when the student needs advice. Intervention occurs only when the student is making erroneous predictions for either the Shape or Orbit descriptors.

The general or high-level teaching strategy is as follows:

If the student has made No errors
and if the student is completing
curriculum materials efficiently
then record progress

If the student has made No errors
and if the student is NOT completing
the curriculum materials
efficiently
then recommend an easier curriculum

If student has made error
then
a) Check ruleset for
satisfaction of preconditions
b) Check ruleset for
Correct Tool Use
c) Check ruleset for
Skill remediations

The author made the general assumption that when the student is in the Prediction Mode, then the system should help students discover the objectives by having them use the tools correctly. If this fails, then the system should address individual skill errors. This strategy is reversed whenever the student enters the Orbit Prediction Testing State.

The system's overall intervention strategy is to check whether the student has completed the necessary preconditions (as determined by the values stored in the State Module). If the student has satisfied all the preconditions for an exercise, then OM checks the measures for effective inquiry skills. The list of effective inquiry skills as originally defined in Shute and Glasser [1987] include skills such as: Systematic experimental behaviors such as making Predictions, asking for Definitions and Examples or accessing the Orbit window.

Every time a student enters a prediction for Shape descriptors or Orbital parameters, the system evaluates the student counters and determines if intervention is required. If the student's effectiveness measures are low, then the Coaching methods propose possible remediation and offer assistance. In the event that the student fails to attain a level of proficiency after receiving instruction on effective tool use, then the system addresses the student's domain knowledge inadequacies.

At the present time, OM uses the information stored in both the Tool Objects and the Expert Objects to advise the student concerning errors. Initially, the system suggests that the student use one of the available tools to correct his errors. If the student continues to have difficulty, then the system may display the definitions, examples or explicitly state the relationships between various parameters.

Whenever you design and implement a computer system, especially an AI system, you always discover some interesting things about the problem that you wish to share with colleagues and friends. This particular project proved NOT to be the exception to this rule. As the Tutor took shape, and as we better understood the domain, we learned several things that will be helpful as we develop the NEXT tutor. What follows is a discussion of some of these ideas.

Don't be afraid to admit that you are ignorant.

Unfortunately, most people tend to believe that they are all- knowing or, at the very least, too proud to admit that they are not all-knowing. This can be a real problem if your job is to design an expert system. One of the reasons that you need to perform knowledge elicitation is that someone or something has more information (and procedures) than anyone else. Thus, you must perform the painful task of questioning the Expert in order to "discover" the knowledge and procedures that are necessary to perform the task. This requires that the Knowledge Engineer admit ignorance, ask stupid questions, and generally try to become student-like. This is a humbling experience and a difficult one at best.

It is better to be a software engineer than a hacker...even in AI environments.

There is a basic myth among people that the words "AI programmer" and "Hacker" are synonomous. Despite such myths, we learned that it was absolutely necessary to use "good" programming practices throughout the development of the Tutor. Thus, the Expert System was changed five times in pursuit of just the "right" data structure. At each stage, we looked at the code and asked if it could be easily maintained, documented and understood. The final system fulfills all of these requirements.

Experts do not always make good teachers.

We initially assumed that if our expert knew how to solve the problem, then she would also know how to teach people how to solve problems. This is not necessarily true. One of the most important qualities of a good teacher is that they are able to reduce most complex problems to a series of very simple, clear procedures. Most good teachers have mastered the art of explaining even the most complex of ideas. They have also mastered the art of knowing what to say when students make mistakes. In short, a good teacher can make sense and order out of chaos. This particular quality is not always present in most experts. They may have performed a task or job because it "feels right", or "looks right." Also, they don't always know what to say to a student when they get the problem wrong. When this occurs, it is necessary to go find a teacher who can tell you how to teach.

Computer programmers, not educators, develop intelligent tutors.

There is a recurring theme in computer education literature that the teacher should be able to sit down at the computer and develop lessons, create interesting curriculum, and program a computer to interact with the student. This is what sold most people on the idea of Authoring Systems for Computer Based Education. It has also been proposed for the creation of Intelligent Tutoring Systems. It is a worthy dream. Yet, the reality of the situation is that programmers, not educators, develop courseware. This is true for traditional intelligent computer systems. Hopefully, this will change at some future time. At the moment, we are stuck with the fact that programmers, not educators, develop curriculum.

SUMMARY AND FUTURE DIRECTION

The current ground track microworld uses a qualitative model to teach the basic concepts of orbital mechanics. This microworld provides the student with a discovery environment which allows him to explore relationships between orbital parameters and ground tracks. The microworld also has intelligence. It knows about the domain, about how to estimate orbital parameters from a ground track, and about how to use the inquiry tools effectively to achieve goals. As a result, if the student fails to make satisfactory progress toward the stated goals, then the system intervenes and offers appropriate assistance. This type of intelligent simulation provides a more active and adaptive environment for reinforcing training skills.

C-2

The initial prototype is now complete and has been formatively evaluated by members of the NORAD crew and instructors at the Space School. The authors performed further tests during the Spring Semester of '87 with students from the Space School at Lowry Air Force Base to determine if the Tutor is more effective than traditional classroom experience. This data will also be used to improve the diagnostic portion of the tutor.

Several areas of research are currently being investigated. Because one of the primary purposes for developing this Tutor was to create a vehicle for testing hypotheses for training effectiveness, we want to investigate specific questions dealing with this area such as: What happens in an instructional environment when you vary the order of the State Module? (Is it better to state a hypothesis and then conduct experiments?) What happens in the instructional environment when you vary the order of remediation? (Tool use versus Skill Diagnosis?) Finally, how can the information we obtain from these studies be made a dynamic part of the system so that it can adapt to individual student's needs? These and other issues will be explored in the coming months and should contribute to our understanding of how to build more effective training systems.

ACKNOWLEDGEMENTS

The authors would also like to thank the instructors and students at the Unified Space Training School (UST) for their assistance with this project.

REFERENCES

Astronautics 332. USAF Academy, Colorado, 1985.

Bates, Roger, Mueller, Donald and White, E. Fundamentals of Astrodynamics. Dover Publications, New York, 1971.

Carr, B., Goldstein, Ira. Overlays: a theory of modelling for Computer Aided Instruction. MIT AI Memo 406 Memo 40, 1977.

Shute, Valerie, and Glasser, Robert. An Intelligent Tutoring System for Exploring Principles of Economics. In Press.

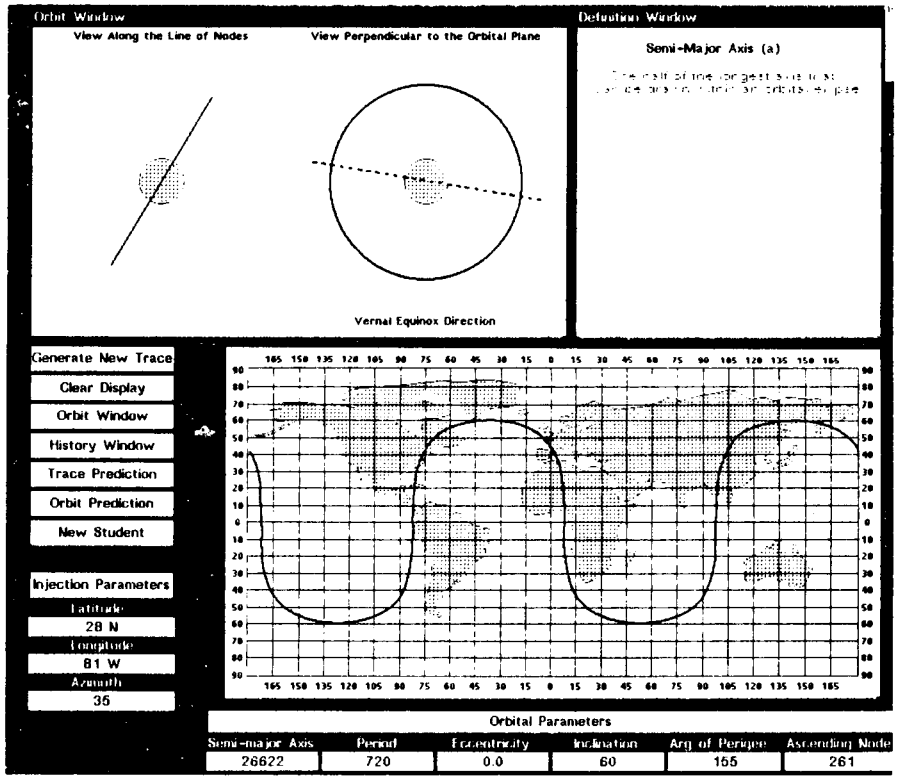


Figure 1: Generate Trace Window

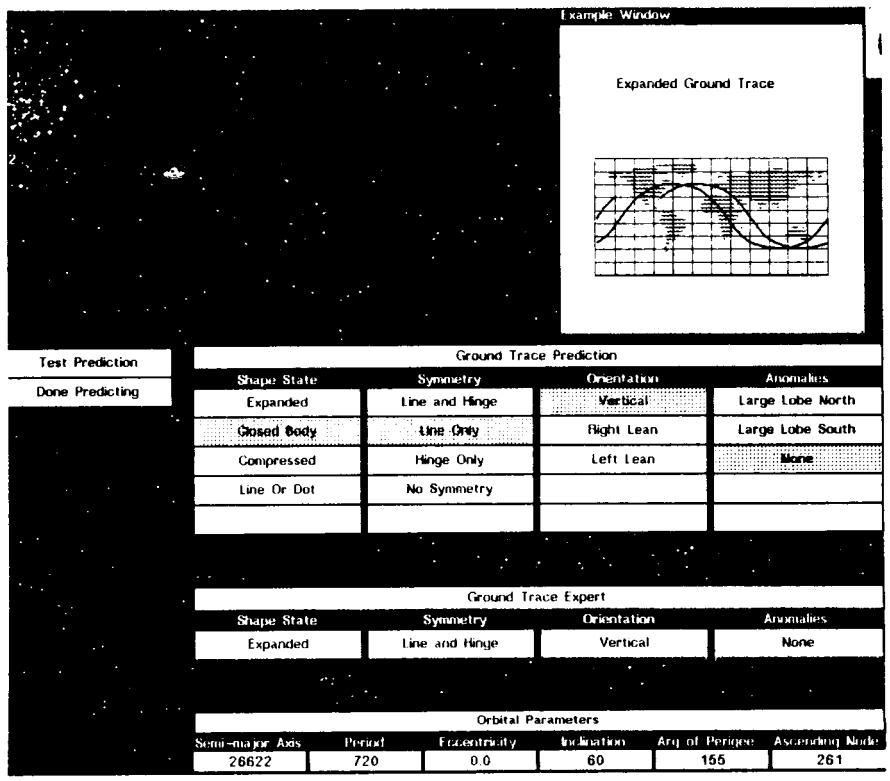


Figure 2: Trace Prediction

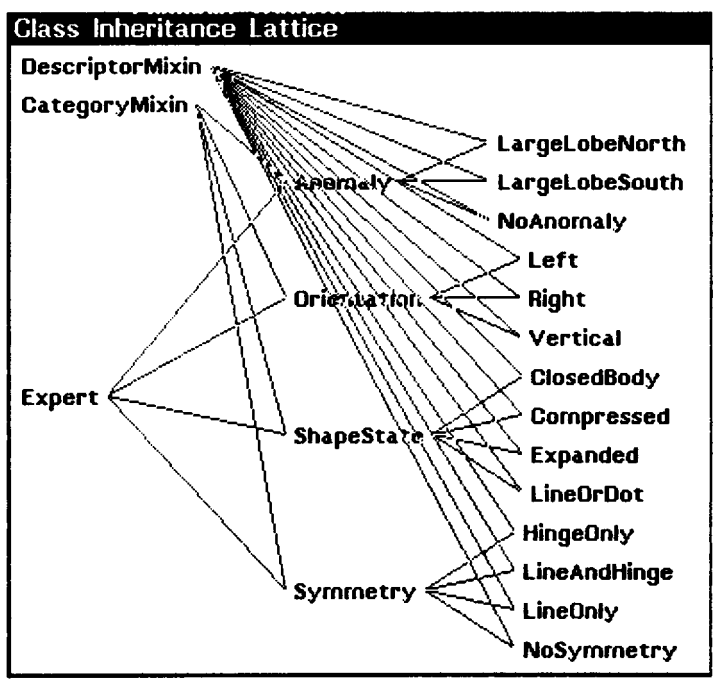


Figure 3: Shape Objects

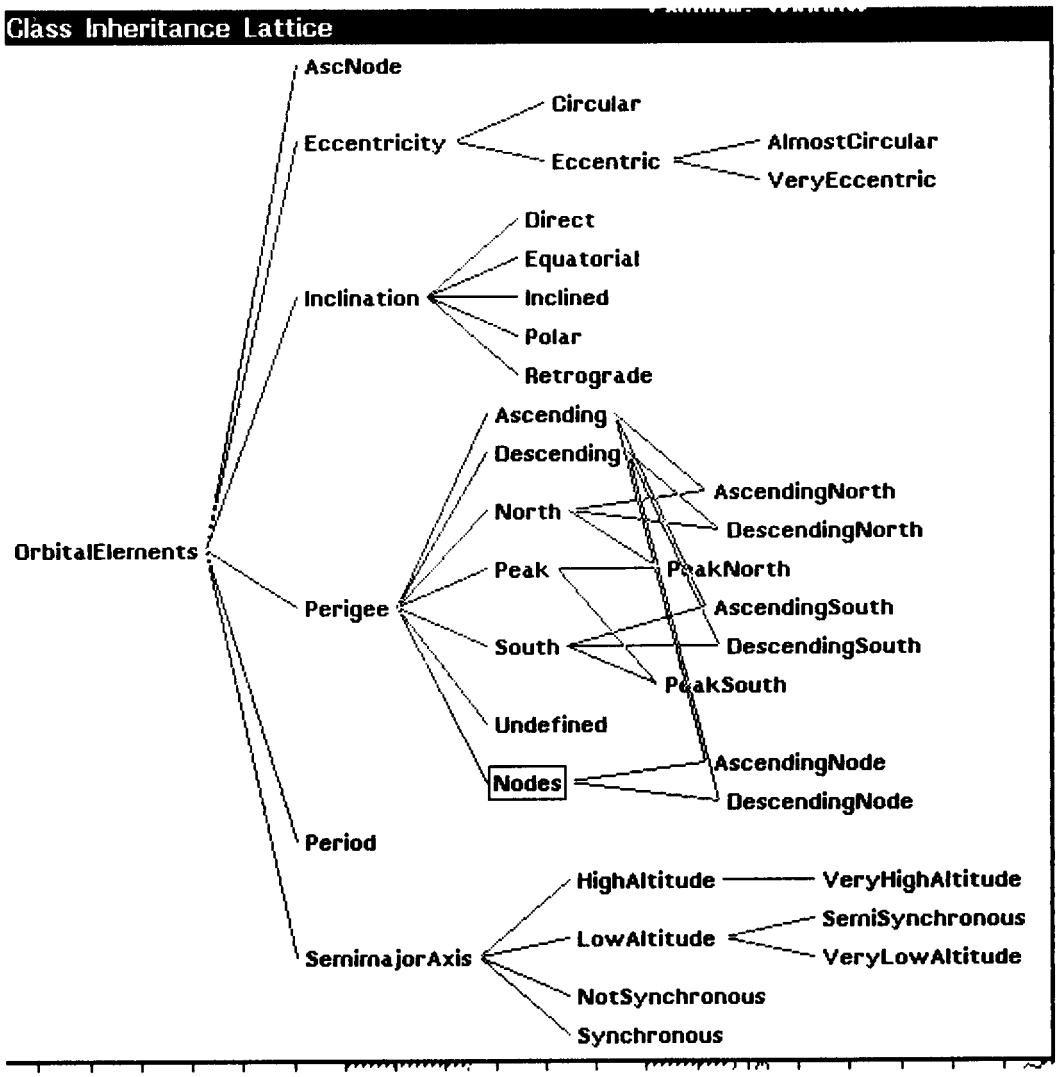


Figure 4: Orbit Objects

NEURAL NETWORK BASED ARCHITECTURES FOR AEROSPACE APPLICATIONS

by

Richard Ricart, 2Lt
System Avionics Division
Avionics Laboratory
Air Force Systems Command
United States Air Force
Wright-Patterson Air Force Base, Ohio

ABSTRACT

The recent fervor and reemergence of research in neural networks has its reasons. The most important are the ability of these systems to store vast numbers of complex patterns, the ability to recall these patterns in O(1) time (i.e., speed of pattern retrieval is independent of number of stored patterns), and the ability to recall these stored patterns using fuzzy or incomplete cues.

In this paper, a brief history of the field will be reviewed and some simple concepts will be described. In addition, some neural network based avionics research and development programs will be reviewed. The concluding remarks will stress the need for the United States Air Force (USAF) and the National Aeronautics and Space Administration (NASA) to assume a leadership role in supporting this technology.

INTRODUCTION

The System Evaluation Branch of the Avionics Laboratory at Wright-Patterson Air Force Base is currently working under a charter to transfer learning research to exploratory development of intelligent electronic combat systems. Neural networks have been identified by this group as having great potential for solving a variety of difficult problems encountered in military avionics.

The purpose of this paper is to show the need for new approaches in developing intelligent systems for the USAF. It can be argued that this need also applies to NASA and the aerospace industry in general. The argument for why neural networks have the potential for satisfying this need will be given by introducing some important properties of neural networks. A brief historical perspective of the field and the current trends in the technology will also be provided. In addition, a brief description of the research and development programs being conducted and planned by the Software Development Group will be given.

NEED FOR NEW APPROACHES IN THE AIR FORCE TO DEVELOPING INTELLIGENT SYSTEMS

The environments in which our military aircraft and weapons systems must operate in have become increasingly complex and hostile with the advancement of technology. Survival will depend on developing autonomous, flexible avionics systems that can adapt and learn from a highly dynamic and hostile environment. However, this is a tremendous challenge due to the complexity of these systems and their environments. The usual problem domains encountered in electronic combat systems, for example, can be characterized as follows: A usually small number of resources must be managed and allocated to satisfy multiple constraints and optimization criteria. These systems are capable of multiple responses under multiple threat and/or target environments. Changes to the environment usually occur very rapidly and sometimes unexpectedly. These systems must process a tremendous amount of information under conditions of novelty, deception, incomplete data, and noise. A further crucial requirement is that these processes must be accomplishable in real-time.

Artificial Intelligence (AI) is one approach to developing "intelligent" systems, but current AI technology has many limitations. The problem domains under which most AI technology has been developed are very different than the problem domains of many military and aerospace applications. The problem domains most expert systems have dealt with have been quite narrow, ideal, and free of noise. Most importantly, processing time has not been a critical factor. AI and other traditional problem solving techniques have had difficulty dealing with many areas such as machine vision, automatic target recognition, situation assessment, and resource planning and control, to name a few. The real-time constraints have been one of the factors contributing to the difficulty in developing AI based solutions to the problems mentioned above.

O'Reilly & Cromarty (1985) have formally defined real-time system performance as the requirement that a system's response to environmental stimuli occur in *provably* finite time (i.e., O(1) time response). The authors show that current AI and traditional problem solving approaches cannot prove this time response and go on to say:

> "...our analysis indicates that there is no reason to expect conventional system design approaches from either school to yield effective, provable real-time performance."

They further propose that parallelism is one way of achieving this performance.
This analysis is consistent with the general acceptance

in the AI community for the need of parallelism in their problem solving approaches.

There has been considerable work in recent years in parallel processing, but developments in hardware have far outstripped the programmers ability to effectively use these systems. We are having problems developing parallel algorithms. This problem is exemplified by the title of a recent paper, *Programming for parallelism: The state of the art of parallel programming and what a sorry state that art is in* (Karp, 1987).

Because of the limitations and slow progress in current AI research and development, especially as it relates to real-world military and aerospace operations, there has been a growing need to re-evaluate research strategies. One alternative approach which has a strong potential for satisfying the Air Force's need for intelligent systems, is *neural networks*.

NEURAL NETWORK SYSTEMS: THEIR PROPERITIES

Neural networks have properties which seem to offer solutions to many of the difficult problems encountered in machine learning, vision, speech, pattern recognition, and real-time resource planning and control. These properties are all interrelated, making it difficult identify the most important one. The remainder of this section will concentrate on explaining these properties.

Most neural networks are modeled after or resemble some of the structure and function of biological brains and nerve cells (neurons), thus their name.* These systems are composed of interconnected processing elements (PEs) or "neurons" which process information in parallel. The PEs have multiple inputs (from the output of other neurons or from external stimuli) and a single output. This output may in turn branch out to yet other PEs or the outside world. Neural networks are inherently parallel processing systems.

An important class of neural networks have the ability to learn and adapt in response to environmental changes. In these neural networks, the PE's have self adjusting weights associated with their input channels (i.e., the conductance of the interconnections change with experience). This self adjusting of network parameters is the basis of learning in neural networks and is one of the most important characteristics of these systems.

One very useful way of interpreting the dynamics of a neural network is as an energy field undergoing changes over time. One can think of this energy field as a flat sheet (it is actually a multidimensional surface). As the network interacts with its environment, wells or basins are created or formed on this flat sheet over time. If the job of the network is to identify or categorize signals of some kind, these wells represent the learned categories. The input stimuli can be thought of as marbles. As new marbles (input stimuli) fall onto this contoured sheet (energy field), the marbles will roll into the closest basin. Marbles that fall into a particular well are similar to the marbles that created the well to begin with. This brings us to the next set of related properties of neural networks. These systems are capable of associating arbitrary input states with the nearest energy basin (identification, category, or response). In addition, these systems decide what the appropriate features of the input states are in order to make the classifications or responses. Therefore, neural networks can act as associative memories, nearest neighbor pattern classifiers, and feature detectors (Kohonen, 1984; Kosko, 1986 and 1987a).

A very important result in neural network research, the Cohen/Grossberg Theorem (Cohen & Grossberg 1983), was popularized in a similar finding by Hopfield (1982 and 1984). This theorem states that the energy of a class of neural networks, called Crossbar Associative Networks (CANs), converges to a finite set of equilibrium points. The energy of the system is defined as a global Liapunov function and the equilibrium points are the local minima of that function. Not only is convergence guaranteed, but the time required to converge to those equilibrium points does not depend on the number of those points. In other words, CANs respond in O(1) time. This is a characteristic of every neural network.

Just as in conventional AI programs, knowledge representation is of utmost importance in neural networks. But knowledge is distributed throughout a massively interconnected processor architecture.* For example, a certain neural network might have the concept of an airplane represented in its network. That concept will be distributed among many PEs and each PE will contain small pieces of information about other concepts; maybe tank, helicopter, jeep, etc. Due to the networks ability to utilize distributed knowledge representations which are supported by massive numbers of parallel elements, these networks are fault tolerant. Neural networks have been shown to exhibit graceful degradation of performance as more PEs become inoperative (Anderson, 1983). One can understand why this occurs from the example of the airplane above. If one or two elements which contain information about that airplane are damaged, the remainder of the network may contain enough of the concept "airplane" to use that information effectively in some type of process. If any piece of hardware or software in conventional computers becomes corrupted, there will be system failure.

*Although there is still considerable disagreement among psychologists on the principles of information processing of the brain, and many neurological functions and cellular mechanisms have not been resolved, mathematical models of certain structures and functions of the brain have been developed with characteristics similar to known neurological functions.

*Many networks have been developed in which knowledge was not distributed. Each PE represented one and only one concept. In these experiments other properties and capabilities of neural networks were being examined which did not require distributed representations.

One final, very important characteristic which is sure to have a considerable impact on the aerospace industry, is that these systems process information without the use of computer programs. What is required is the specification and development of an architecture of interconnected PEs for a given problem. Each PE of the neural network is governed by a system of mathematical equations which can be implemented directly in electronic circuitry. After an architecture is defined, the neural network is then put through a training or learning stage. It is in this stage that the system learns the appropriate I/O mappings with either the help of a "teacher" or "critic", or on its own if enough a priori information is built into the system.* Still other systems can learn continuously as they interact with their environments.

Before leaving this section, a brief, high level description of the mathematical equations governing a neural network will be given. The typical PE is governed by usually two or three differential equations (or difference equations when dealing in discrete time). One equation determines the activity or state of the PE, another determines the change in conductances (or the final values of the conductances after the network settles to a stable state) of the PE's input channels, and the third equation determines the output of the PE. When the PEs are governed by two equations, the activity of the PE is usually incorporated into the output equation. The activity equations are usually some function of the sum of the weighted inputs. The output equations are usually a nonlinear function of the activity (either sigmoid or linear threshold). And the change in input conductances are usually some function of the inputs, output, and the conductances themselves.

These dynamical equations come in a variety of forms which have either evolved or have been added over the years to give us a large and rich repertoire today. This variety reflects the diverse and interdisciplinary background of the researchers in the field: neuroscience, psychology, physics, mathematics, engineering, and computer science.

A number of important attributes of neural networks have been discussed. It must be emphasized, however, that the engineering process of developing architectures, especially for real world problems, is still in its infancy. Convergence theorems for many classes of important neural networks have not been found. Fortunately, we do have enough empirical data to suggest that convergence proofs to some of these systems may be found. Other problems include strict limits on the amount of data storage imposed on a neural network of given size and the ability of associative memories to create spurious energy minima (Kosko 1987a). The important point to stress is that neural networks offer a tremendous *potential* for solving many difficult problems which solutions have not been previously, or acceptably found. But before this potential is realized, much work needs to be done.

---

*More detailed discussions on the different classes of learning and how these are accomplished in neural networks are discussed in Duda & Hart (1973), Barto & Sutton (1981), and Barto (1985).

HISTORY AND CURRENT TRENDS

In this section, a brief history and the current trends in neural network research will be introduced in order to give a general feel for the field. For a comprehensive review see Levine (1983). Barto (1984) also presents a more in depth review than the one found here. Probably the best introduction to neural networks is provided by Rumelhart, et al (1986). This work also includes research more appealing to those with AI and cognitive science backgrounds.

The early concepts of processing information by a network of simple linear threshold elements were introduced by McCulloch and Pitts (1943). They developed very simple linear threshold processing elements with boolean output which were interconnected via positive and negative input lines. Their results generated much excitement for they showed any logical function could be performed by some configuration of such networks.

The next major milestone was achieved by Hebb (1949) when he postulated a mechanism for long term memory. This mechanism required a structural change to the neuron:

> "When the axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased (Hebb, 1949, p. 64)"

This hypothesis was later interpreted mathematically as

$$\frac{d}{dt} w_i = x_i y$$

where w_i is the weight associated with the ith input to the neuron, x_i is the ith input signal from another neuron, and y is the output of the neuron in question. This rule has had a tremendous impact on neural network research for it has been used in one form or another in virtually every learning neural network conceived.

It wasn't until the late 50s and early 60s that neural networks were developed along the lines in which we are familiar with today. McCulloch and Pitts' ideas of interconnected linear threshold elements and Hebb's ideas of long term memory were integrated into very useful devices. Two such systems deserve special attention: The Perceptron, developed by Rosenblatt (1962) and the Adaline (for adaptive logic element), developed by Widrow (1962). Both of these systems were similar in that they were based on a single adaptive layer of neurons and on an error correcting mechanism. The difference between the desired response and the actual response was fed back to the adaptive layer through a series of training trials until the network converged to a solution in provable finite time.

Unfortunately for Rosenblatt and for neural network research for the next 20 years or so, Rosenblatt made claims which seemed unfounded to several of his contemporaries. This led to Minsky and Papert's (1969) critical response to the Perceptron (see Rumelhart, et al, 1986, pp. 151-159). Minsky and Papert showed a number of limitations of single layered adaptive networks and

also raised the issue of the credit assignment problem in multilayered networks of error correcting elements. The credit assignment problem arises as a result of the inability of cells within the interior layers of the network to know what fraction of the total error they are responsible for. These problems have been solved in a variety of ways since then (Parker, 1982 and 1985, and Rumelhart, et al, 1985), but in those days they raised alarming questions.

Minsky and Papert's book was devastating to Rosenblatt and neural network research. The book was a sign to many that research should be directed towards symbolic processing and heuristics. This approach is what we know as AI today. The push for this approach was also being heavily influenced by the growing field of cognitive psychology (for a historical view from this perspective see Gardner, 1985). At this same time, behaviorist psychology was in decline. This also helped sway research funds away from neural networks since the issues involved in neural networks were highly reminiscent of the issues the behaviorists were grappling with: stimulus/response chains, reinforcement, and behavior based on microstructural concepts.

Widrow, on the other hand, was extremely successful applying his Adaline and Madaline (for many Adalines) to signal processing. His adaptive signal processing techniques have been applied to system modeling (i.e., imitating system behavior), inverse system modeling, adaptive control systems, adaptive interference canceling, and to adaptive antenna arrays (Widrow, 1985). It is also interesting to point out Widrow's achievements in the 60s. His Knobby Adaline (a hardware implementation of his adjustable threshold element, Widrow 1962) was able to recognize patterns regardless of noise (10%), rotation (90 degrees), left and right translation, and size (25%). The Avionics Laboratory, at Wright-Patterson Air Force Base owns a film* of a pole balancing experiment performed by Widrow. A small cart with a pole attached to the top of the cart by a pivot was placed on a short track. The Madaline was able to keep the pole balanced by controlling the cart's movement after a series of training trials. In that same film, Widrow's students are shown training a Madaline to translate spoken words in three languages to type written English. One may wonder whether the neural network "nuclear winter" that ensued would have taken place if these results would have been marketed as vigorously as the limitations to the technology at the time.

Although neural network modeling fell from grace after Minsky and Papert's book, very important work continued throughout the 70s. Fukushima (1975) and von der Marlsburg (1973) developed systems based on the visual structures of biological brains. Kohonen (1972) and Wilshaw, et al (1970 and 1971) were early pioneers in the area of associative memories. Amari, et al (1974 and 1977) made an important contribution through his research in associative memories and their relation to thermodynamics. Klopf (1972 and 1979) introduces the concept of the neuron as a goal oriented or goal seeking agent (heterostat). Rescorla and Wagner (1972) developed a model which exhibited a variety animal learning phenomena.

*The USAF supported Widrow's research as well as other neural network research in those early days.

The most prolific contributor to this field has been Stephen Grossberg, of the Center for Adaptive Studies, at Boston University. Grossberg has addressed all the main issues in neural networks from 1967 (Grossberg, 1967) to date in approximately 130 papers and 4 books. He has approached his research with rigorous mathematics and has proved some of the most important theorems in the field. He has investigated and written about memory, animal learning behavior, cognition, speech, language, vision, and motor control. He's collected his most important work in three volumes (Grossberg, 1982, 1987a, and 1987b).

John Hopfield of Caltech and Bell Labs is, perhaps, the one most responsible for reigniting the field. In two articles (Hopfield, 1982 and 1984), Hopfield, expanding on previous work on crossbar associative networks (CANs), made connections between CANs and Ising spin glass models of ferromagnetism. Hopfield's papers made a strong impact on the physics and optics community in a series of conference presentation. Hopfield and Tank (1985) further publicized the information processing capabilities of neural networks by developing a CAN system which had the ability to find near optimum solutions to a traveling salesman problem. In other words, they developed an O(1) time approximate solution to a NP-hard problem using neural networks (see Hecht-Nielsen, 1986 for this discussion). Interest in the field has mushroomed in academia, the Department of Defense, and throughout the aerospace industry since Hopfield's 1982 paper.

Today, theoretical work continues at a fast pace from many of the original pioneers mentioned above and from scores of others entering this exciting field. Over 200 papers were presented in 16 sessions at the Institute of Electrical and Electronics Engineers (IEEE) sponsored First Annual International Conference on Neural Networks in San Diego, California, between the 21st and 24th of June, 1987. Nearly 2,000 people were attracted to this event. The 80s have also brought much needed work in the hardware implementation of neural networks. In the past, almost all work was simulated on general purpose computers. Experiments could run for days in those early years. Special purpose processors are coming to market today which can significantly increase processing speed. For many applications, these are sufficient for real-time processing. For more difficult problems such as vision or target recognition, much larger networks will be required. If these networks are to be flown in spacecraft and aircraft, they'll have to be implemented in silicon, optics, or a combination of both. Fortunately, work is well under way addressing this need. The following is only a small sampling of optical and electronic neural network research: Cruz-Young & Tam, 1985; Dunning, et al, 1986; Graf & deVegvar, 1987a and 1987b; Fisher, et al, 1986; Psaltis and Abu-Mostafa, 1985; Psaltis and Farhat, 1985; Sivilotti, 1985; and Soffer, 1986.

SYSTEM EVALUATION BRANCH's (AAAF) R&D EFFORTS

Basic research in neural networks has matured to a point suitable for translation into exploratory development. AAAF's efforts are aimed at advancing neural network research in both the signal processing and cognitive processing areas for avionics applications.

The ultimate goal is to merge both areas of research and develop the technology for providing intelligent avionics sensor systems for the USAF. We are specifically addressing the avionics domain from the level of sensors and emitters in electronic combat applications. This research is part of a long term program, *Intelligent Avionics*, which is in general addressing the issue of making avionics adaptive. Both contracted and in-house efforts will be conducted under this program.

Two contracts are currently being managed in the neural network area. The first is the *Adaptive Network Cognitive Processor* (ANCP), a one year effort which was awarded to TRW in San Diego, California. The purpose of this program is to develop a prototype system which builds an inner model of its environment in the form of cognitive maps and uses this model for reasoning, planning, or problem solving. The exact problem domain is a high level situation assessment and response system for pilot aiding. This is a "proof of concept" program. A TRW Mark III neurocomputer is being used for neural network design and simulation and will be delivered as part of the prototype.

The second program, the *Adaptive Network Sensor Processor*, will apply neural network associative memory and pattern recognition technology to a military radar warning system for providing identification, categorization, and classification of previously experienced and novel radar signals in a noisy and corruptive environment. A comparison between this new approach to radar signal identification and conventional means of signal processing will be accomplished before system delivery. There are two contractors working on this program: Booz-Allen & Hamilton, Inc. from Arlington, Virginia and Texas Instruments from Dallas, Texas. A Hecht-Nielsen Neurocomputer Company (HNC) ANZA neurocomputer will also be used by Booz-Allen & Hamilton. Texas Instruments will be using their array processor board, the Odyssey, in conjunction with their Explorer work station for developing and simulating the neural network and environment.

Follow-on efforts for both of these programs are being planned. The *Adaptive Network Avionics Resource Manager* (ANARM) will apply what is learned in ANCP to a specific electronic combat system. The *Adaptive Network Radar Signal Processor* will integrate ANSP with a response module to provide closed loop learning. Hardware implementation issues will also be investigated. These programs are scheduled to start in fiscal year 1988 and fiscal year 1989 respectively.

In-house research is also being conducted under a program entitled *Real-time Adaptive Avionics*. As part of this effort, a neural network design tool was developed and implemented on an LMI (now Giga Mos) Lambda LISP Machine. The *Artificial Neural Design Environment* (ANDE) has been used to investigate the application of Klopf's (1986) Drive-Reinforcement Neuronal Model to a simulated avionics control problem. The ultimate goal of this research is to transfer a neural network architecture to electronic combat groups which can perform real-time, adaptive resource management and control. Support for this research is being pursued from the Air Force Office of Scientific Research (AFOSR).

CONCLUSION

The United States Air Force and the National Aeronautics and Space Administration should assume a leadership role in advancing neural network research and development efforts because of the tremendous potential for providing adaptive, fault-tolerant aerospace systems. We have available to us a viable technological alternative which offers potential solutions to such complex problems as data fusion, machine vision, automatic target recognition, resource planning and control, and adaptive system control. The important characteristics of neural network, which are summarized below, must be exploited and used in innovative ways in order for this potential to be realized. Neural networks are parallel processing systems that can respond in O(1) time. These systems can learn and adapt to their environment and are fault tolerant to damage. And finally, neural networks can process information without the need of computer programs. The foreseen software explosion and crisis could be diminished or alleviated.

Neural network technology will not supplant current computer science and software development where these are more appropriate. Rather, hybrid systems are envisioned with each technology performing what it does best. New developments in neural network technology, however, have the potential to revolutionize and greatly enhance intelligent information processing for our country's defense and space science. It is also clear that the USAF and NASA should steer the research efforts in this area in order that neural network technology develops in a manner suited to aerospace requirements. The System Evaluation Branch of the Avionics Laboratory at Wright-Patterson Air Force Base is committed to develop and exploit this "new" technology for developing intelligent avionics systems.

ACKNOWLEDGEMENTS

Special thanks to Lt Mark Peot for his valuable comments and assistance.

REFERENCES

Amari, S., "A Method of Statistical Neurodynamics", *Kybernetik, Biological Cybernetics, 1974*, Vol. 14, pp. 201-225.

Amari, S., Yoshida, K. and Kanatani, K.,"A Mathematical Foundation for Statistical Neurodynamics", *SIAM J. Appl. Math*, Vol. 33, No. 1, 1974, pp. 95-126.

Anderson, J.A. "Cognitive and Psychological Computation with Neural Models", *Transactions on Systems, Man, and Cybernetics*, Vol. SMC-13, No. 5, Sept/Oct, 1983, pp. 799-815.

Barto, A.G., ed., "Simulation Experiments with Goal-Seeking Adaptive Elements", *Air Force Wright Aeronautical Laboratories/ Avionics Laboratory Technical Report AFWAL-TR-84-1024*, Wright-Patterson AFB, OH., 1984.

Barto, A.G., "Learning by Statistical Cooperation of Self-interested Neuron-like Computing Elements", *Human Neurobiology*, Vol. 4, 1985, pp. 229-256.

Barto, A.G. and Sutton, R.S., "Goal Seeking Components for Adaptive Intelligence: An Initial Assessment", *Air Force Wright Aeronautical Laboratories/ Avionics Laboratory Technical Report AFWAL-TR-81-1070*, Wright-Patterson AFB, OH., 1981.

Cohen, M. and Grossberg, S., "Absolute Stability of Global Pattern Formation and Parallel Storage by Competitive Neural Networks", *IEEE Trans. of Sys., Man, and Cybernetics, SMC-13*, 1983, pp. 815-926.

Cruz-Young, C. and Tam, J.Y., "NEP: An Emulation Assist Processor for Parallel Associative Networks", IBM Palo Alto Scientific Center Report No. 6320-3475, 1985.

Dunning, G.J., Marom, E., Owechko, Y., and Soffer, B.H., "Optical Holographic Associative Memory Using a Phase Conjugate Resonator", *Proc. SPIE*, 1986, p. 625.

Duda, R.O. and Hart, P.E., *"Pattern Classification and Scene Analysis"*, Wiley, New York, 1973.

Fisher, A.D., Jukuda, R.C. and Lee, J.N., "Implementation of Adaptive Associative Optical Computing Elements", *Proc. SPIE*, 1986, p. 625.

Fukushima, K., "Cognitron: A Self-Organizing Multi Layered Neural Network", *Biol Cybernetics*, 20, Nov., 1975, pp.121-136.

Gardner, H., *"The Mind's New Science"*, Basic Books, Inc., New York, 1983.

Graf, H.P., and deVegvar, P., "A CMOS Associative Memory Chip Based on Neural Networks", *Digest, ISSCC*, Vol. 304, New York, 1987a.

Graf, H.P., and deVegvar, P., "A CMOS Implementation of a Neural Network Model", *Proc., Stanford Conf. Advanced Research in VLSI*, March, MIT Press, 1987b.

Grossberg, S., "Nonlinear Difference-Differential Equations in Prediction and Learning Theory", *Proc., Nat'l Acad. Sci.*, Vol. 58, 1967, pp. 1329-1334.

Grossberg, S., *"Studies of Mind and Brain"*, Boston: Reidel Press, 1982.

Grossberg, S. ed., *"The Adaptive Brain I: Coginition, Learning, Reinforcement, and Rhythm"*, Amsterdam: North-Holland, 1987a.

Grossberg, S. ed., *"The Adaptive Brain II: Vision, Speech, Language and Motor Control"*, Amsterdam: North-Holland, 1987b.

Hebb, D.O., *"The Organization of Behavior"*, Wiley, New York, 1949.

Hecht-Nielsen, R., Technical paper "Artificial Neural Systems Technology", Unpublished technical paper, TRW,Rancho Carmel AI Center, San Diego CA., 1986.

Hopfield, J.J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", *Proc., Nat'l Acad. Sci.*, Vol. 79, 1982, pp. 2554-2558.

Hopfield, J.J., "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons", *Proc. Nat'l Acad. Sci.*, Vol. 81, May, 1984, pp. 3088-3092.

Hopfield, J.J. and Tank, D.W., "Neural: Computation of Decisions in Optimization Problems", *Biol Cybern.*, Vol. 52, 1985, pp. 141-152.

Karp, A.H., "Programming for Parallelism: The State of the Art of Parallel Programming and What a Sorry State That Art Is In", *Computer*, Vol. 20, No. 5, Computer Society of the IEEE, 1987, pp. 43-57.

Klopf, A.H., "Brain Function and Adaptive Systems: A Heterostatic Theory", *Air Force Cambridge Research Laboratories Research Report AFCRL-72-016 4*, Bedford, Mass, 1972..

Klopf, A.H., "Goal-Seeking Systems from Goal-Seeking Components", *Cogn and Brain Theory Newsl. 3*, No. 2., 1979.

Klopf, A.H., "A Drive-Reinforcement Model of SingleNeuron Function: An Alternative to the Hebbian Neuronal Model", In J.S. Denker (Ed.), *Neural Networks for Computing*, AIP Conference Proceedings 151, New York: American Institute of Physics, 1986, pp. 265-270.

Kohonen, T., "Correlation Associative Memories", *IEEE Transactions on Computers"*, Vol. C-21, 1972, p. 353.

Kohonen, T., *"Self-Organization and Associative Memory"*, Springer-Verlag, New York, 1984.

Kosko, B., "Fuzzy Associative Memories", In A. Kandel Ed., *Fuzzy Expert Systems*, Addison-Wesley, 1986.

Kosko, B., "Bidirectional Associative Memories" *Proc., SPIE: Image Understanding*, Vol. 758, Jan, 1987a.

Kosko, B., "What is an Associative Memory?" In review, 1987b.

Levine, D.S., "Neural Population Modeling and Psychology: A review", *Mathematical Biosciences*, Vol. 66, 1983, pp. 1-86.

McCulloch, W.S. and Pitts, W., "A Logical Calculus of the Ideas Immanent in Nervous Activity", *Bulletin of Mathematical Biophysics*, Vol. 5, 1943, pp. 115-133.

Minsky, M.L., and Papert, S., *"Perceptrons: An Introduction to Computational Geometry"*, Cambridge, MA: MIT Press, 1969.

O'Reilly, C.A. and Cromarty, A.S., "Fast Is Not Real-Time: Designing Effective Real-Time AI Systems, *Applications of Artificial Intelligence II*", Proc. SPIE 548, 1985, pp. 249-257.

Parker, D.B., "Learning Logic", *Invention Report, 581-64, File 1, Office of Technology Licensing*, Stanford University, Oct, 1982.

Parker, D.B., "Learning Logic", *TR-47, Center for Computational Research in Economics and Management Science*, MIT, April, 1985.

Psaltis, D. and Abu-Mostafa, Y.S, "Computation Power of Parallelism in Optical Computers", California Institute of Technology", 1985.

Psaltis, D. and Farhat, N., "Optical Information Processing Based on an Associative-Memory Model of Neural Nets with Thresholding and Feedback", *Optics Letters*, Vol. 11, Feb, 1985, pp.118-120.

Rescorla, R.A. and Wagner, A.R., "A Theory of Pavlovian Conditioning: Variations in the Effectiveness of Reinforcement and Non-Reinforcement", In Black, A.H. and Prokasy, W.F. Eds., *Classical Conditioning II: Current Research and Theory*, Appleton-Century-Crofts, New York,1972.

Rosenblatt, F., *"Principles of Neurodynamics"*, Spartan Books, Washington, 1962.

Rumelhart, D.E., Hinton G.E., and Williams, R.J., "Learning Internal Representations by Error Propagation", *Institute for Cognitive Science Report 8506*, UCSD, Sept,1985.

Rumelhart, D.E., McClelland, J.L., eds., *"Parallel Distributed Processing", I & II*, Cambridge: MIT Press, 1986.

Sivilotti, M., Emerling, M., and Mead, C., "A Novel Associative Memory Implemented Using Collective Computation", *Proc. Chapel Hill Conference on VLSI*, 1985.

Soffer, B.H., Dunning, G.J., Owechko, Y., and Maron, E., "Associative Holographic Memory with Feedback Use Phase-Conjugate Mirrors", *Optics Letters*, 11, Feb, 1986, pp. 118-120.

von der Marlsburg, C., "Self Organization of Orientation Sensitive Cells in the Striate Cortex", *Kybernetik*, Vol. 14, 1973, pp. 85-100.

Widrow, B., "Generalization and Information Storage in Networks of Adaline "Neurons"", In Yovits, M., Jacobi, G., and Goldstein, G., Eds., *Self-organization systems*, Spartan Books, 1962, pp. 435-461.

Widrow, B., *"Adaptive Signal Processing"*, Prentice-Hall,Inc., Englewood Cliff, N.J., 1985.

Wilshaw, D.J., *"Models of Distributed Associative Memories"*, Ph.D. Thesis, University of Edinburgh, 1971.

Wilshaw, D.J. and Longuet-Higgins, H.C., " Associative Memory Models", in B. Meltzer and O. Michie Eds., *Machine Intelligence*, Edinburgh: Edinburgh University Press, 1971.

DEVELOPMENT AND EXPERIMENTATION OF AN EYE/BRAIN/TASK TESTBED

Nora Harrington, Ph.D., Analytics, Willow Grove, PA

James Villarreal, NASA/JSC, Houston, TX

An SBIR (Small Business Innovative Research) Phase I was awarded to Analytics to investigate the feasibility of an innovative concept that uses an operator's brain waves as a control mechanism for computer systems. The Phase I reported that the present brain wave recording technology is incapable of using these signals for direct data transmission. But the development of such technologies as super conductive materials at near room temperature and biomagnetism is advancing rapidly. A direct application from conventional MEG or EEG sensing systems could determine an operator's state of awareness.

The principal objective of Phase II is to develop a laboratory testbed that will provide a unique capability to elicit, control, record, and analyze the relationship of operator task loading, operator eye movement, and operator brain wave data in a computer system environment. The ramifications of an integrated eye/brain monitor to the man and machine interface are staggering. The success of such a system would benefit users of space and defense, paraplegics, and the monitoring of boring screens (FAA, nuclear plants, Air Defense, etc.)

INTRODUCTION

A variety of man-machine interface concepts have been developed in recent years in an attempt to: (1) increase the flow of relevant information between the system and the operator, and (2) alleviate the need for complex programmer-oriented input protocols. These concepts have concentrated on the presentation, selection, or display aspects of the interface. Another component of innovative interface design is control mechanisms for computer systems. Development of advanced hardware and software systems for mission planning and control is desirable to enhance the human operator's job performance, especially during periods of high workload.

METHODOLOGY

The Phase I research was an investigation of the feasibility of using brain waves as a control input to a computer system. Currently, there are a number of devices, such as the mouse and the touch screen, that allow for more direct and intuitive control than do conventional keyboards. Use of these devices requires only simple software for managing hardware communication protocols, but the approach to controlling a system via brain waves requires more sophisticated software for the interpretation of encephalographic data. Although, in the absence of pilot studies, it is premature to assume that brain wave sensing is capable of conveying complex instructions to a computer, it seems plausible that brain waves are capable of conveying coarse information.

In order to establish the feasibility of the concept of using brain wave sensing for computer control several research questions were addresses. A review of several technologies was undertaken in order to evaluate the relative merits of each technology to the application.

Another issue considered in the Phase I research was the current status of hardware necessary for measuring brain waves. The field of neuromagnetometry is advancing rapidly, but is still in its early stages of development. It was clear that if conclusions were based on existing instrumentation and methods of data analysis the results of the Phase I feasibility study would have been that the control of systems through brain wave signals was not very practical. Therefore, the scope of the Phase I research was expanded to include an evaluation of anticipated future developments in instrumentation.

The approach to the Phase I feasibility study involved several research techniques. Initially, an extensive literature search was conducted to determine the state-of-the-art in the application of MEG technology. The literature review revealed that MEG had advantages over the conventional EEG, however the scope of the MEG was limited.

Electroencephalography (EEG)

In the domain of the spontaneous EEG only a very limited band of EEG activity could be of possible use. Alpha activity is of large amplitude and it is strongly associated with activity of the visual cortex in the relaxed, wakeful but visually inattentive individual. In principle, the modulation of alpha by changes in the level of visual attention could be used for control of computers. However, this would be a very primitive level of control, as the changes in level of alpha activity are quite slow as compared

to the speed with which a person can type instructions at a computer keyboard. To date, studies of the effects of changes in level of attention have been based on very simple experimental paradigms. The results of these studies do not provide conclusive evidence regarding the variables that affect the alpha wave form. Therefore, alpha and its modulation should be studied further using procedures that provide a much greater degree of experimental control over the amount and type of mental work being done by the subject.

Magnetoencephalography (MEG)

Magnetic recording techniques offer several advantages for monitoring specific neural activity in the brain with regard to computer control. However, some of the problems associated with the EEG are also present in MEG. Specifically, alpha activity is indeed the strongest MEG spontaneous signal, it is not certain precisely how it is affected by states of attention, as well as intentions, even when these states are under very good experimental control. Clearly the problem resides in the cognitive processes of the human operator, and not within the recording techniques. For example, subjects given a vigilance task in which they simply monitor an oscilloscopic display are required to make decisions based upon changes in the visual information. The relative frequencies with which such decisions must be made, the difficulty in making the "correct" decision, the properties of the display itself (e.g., spatial frequency content, flicker or temporal frequency, moving or static targets), and the duration of an experimental session are all factors that could affect MEG output.

Eye/Brain/Task Monitor Concept

A system for monitoring operator task activity can now be built around the manual control operations necessary to perform various task procedures. This is possible because a computer can easily be made aware of these events. A certain degree of task-level ambiguity is inherent in such operations but most of these could be resolved with a fair degree of certainty by reference to fixed task-domain knowledge. However, a more severe criticism can be leveled against such a system on the grounds that its results are of too coarse a grain. Decisive action is usually taken after a considerable period of "silent" mental activity -- analytical tasks performed by an operator are not likely to be identifiable in the sequence of manual control operations. Unfortunately it is just such tasks which are of critical importance to decision aiding and intelligent problem solving systems. Analytics believes that a system which records and correlates human ocular and brain wave activity can bridge the gap between isolated manual control operations.

The examination of ocular activity can clarify what is going on during discrete control events. Eye data is ambiguous when used to identify an ongoing task: scanning out a straight line could as easily mean that the operator was tracking a moving object, estimating a path, or even briefly glancing from one point to another. Since the effect of visual attention and vigilance on brain wave activity is extremely robust, it is expected that components within specific wave bands can be used to disambiguate ocular behavior vis-a-vis operator performance of analytical tasks. This is not to say that brain wave data will not itself prove ambiguous, for it will undoubtedly do so. The entire task identification problem is characterized by the need to resolve potential ambiguities and conflicts in and between all the various levels and types of available information -- eye position, brain wave readings, manual events and fixed domain knowledge.

In order to handle mutual disambiguation, an actual system must be capable of passing information both upwards and downwards between levels to achieve a "best fit" between the low-level information and the high-level task hypotheses. This type of processing has been applied successfully in the domain of speech recognition (for example, in the HEARSAY system), where low-level phoneme and word recognition is permitted to interact with higher-level notions of syntax and semantics. Errors and ambiguities in word recognition can be corrected by determining what "makes sense" in the context. This kind of approach is generally termed "hypothesize-and-test", since there are several independent knowledge sources and the interpretation of each can be evaluated against the interpretation of the other. Also termed "iterative guess-building", the reinterpretation ceases when some predetermined level of confidence has been attained for the interpretation system as a whole. For the eye/brain/task monitor it is expected that eye data, brain wave data, manual control data, and knowledge of the mission task domain (a task syntax) can be fused to build a continually updated task history which can be extended as needed for purposes of prediction. In the context of this application, the feasibility of using brain wave information to contribute to computer system control appears highly plausible.

The successful application of brain wave data to intelligent systems revolves around a thorough understanding of the complex linkage of task structure, operator eye-movement, brain wave response, and task syntax. The definition of that linkage at a level sufficiently specific to provide the basis for distributed intelligence system algorithms requires that a testbed be developed that focuses specifically on the issue of eye/brain/task linkage (Figure 1).

Analytics, under contract to NASA, has pioneered development of the application of eye-sensing technology to computer control and has successfully integrated an eye/voice controlled interface into a complex task/scenario generator. This unique system, called OASIS, has been refined and demonstrated as a working prototype. OASIS will provide a baseline system that will be further refined with the addition of brain sensing capability into a functional prototype testbed that will focus directly on the issue of eye/brain/task linkage.

TECHNICAL FEASIBILITY OF COMPUTER CONTROL

In almost every field where computer hardware is employed, operator work stations are characterized by growing complexity and continuously increasing data flow. In general, two major issues are of prime concern: 1) the increased operator workload and 2) the reduced habitability which typically results when older control technologies are extended to support increased functionality.

Workload problems are believed to be responsible for operator errors in critical tasks and more generally for reductions in operator effectiveness or productivity. A competent workstation design attempts to reduce workload by efficiently organizing the entire suite of operator tasks. More recently, system developers have begun to focus on the possibility of creating additional channels for operator/machine communication and of redistributing workload across the resulting range of control options. This is of special interest when a continuous control task such as steering must be integrated with a variety of other operations. Offloading to new control channels os also of interest in the context of special environmental conditions such as high G forces where normal operator functioning may be highly restricted. The development of commercially available voice systems is the most obvious result of this approach to the issue of operator workload, although other technologies such as control by head position are already in use. Voice interaction has been of particular interest to developers in the aerospace industry where hands-busy and eyes-busy operation are common and where workload redistribution is an attractive solution.

Humans and machines are rapidly becoming components in distributed intelligence systems where tasks are performed cooperatively. When tasks are complex, the passive role of "ready servant" requires that operator needs be anticipated, much as the nurse attending a surgeon must know what is likely to be requested before the request is made. As the computer begins to take on a more active role, the need for machine knowledge of operator activity and intentions becomes essential. Now the machine may need to query the operator regarding his actions or plans, as well as spontaneously criticize or offer alternative solutions. Ideally, a smart system would know when intervention was appropriate. By analogy to the situation of human cooperation, it is obvious that in all but the most critical situations the appropriateness of intervention is dependent on an understanding of what the operator is doing or is about to do.

In order to cooperatively solve a problem, humans depend on shared knowledge regarding the problem domain and available courses of action. Techniques are already available for providing machines with this type of intelligence. However, humans also depend on observation of their partners, frequently utilizing subtle cues to assess the significance of more easily recognized actions. For example, facial expressions and posture are usually taken as indicators of a person's relative satisfaction with the results attained by specific goal directed activity. Efficient human cooperation requires just this kind of inference in order to continuously adjust individual strategies as a problem develops over time. Unfortunately, the cues used by humans themselves are by no means completely understood and many, such as facial expressions, would require major research efforts before the sensing equipment itself could be developed. In order to provide a machine with the inference capabilities required for efficient cooperation, all available resources will have to be focused on machine understanding of operator behavior. This will require a dedicated, focused laboratory facility such as the EBT testbed.

The successful application of brain wave data to intelligent systems revolves around a thorough understanding of the complex linkage of task structure, operator eye-movement, brain wave response, and task syntax. The definition of that linkage at a level sufficiently specific to provide the basis for distributed intelligence system algorithms requires that a testbed be developed that focuses specifically on the issue of eye/brain/task linkage.

NEXT STEPS

The Phase II effort will concern itself with the development of a prototype Eye/Brain/Task (EBT) testbed, and through applied research and development, the refinement and optimization of the system. The principal objective of the proposed Phase II effort is to develop a laboratory testbed that will provide a unique capability to elicit, control, record, and analyze the relationship of operator task loading, operator eye movement, and operator brain wave data in a computer system environment. Additionally, the testbed will have the capability to serve as the vehicle for demonstrating computer control using brain waves at a future time.

ACKNOWLEDGEMENTS

The research presented in this paper was sponsored by NASA under the Small Business Innovative Research (SBIR) program. In addition to the authors, major contributions to this effort were made by James Deimler and James Stokes. The authors would also like to acknowledge the contributions of Dr. Lloyd Kaufman and Dr. Samuel J. Williamson of New York University. At Analytics, special thanks are extended to Ms. Phyllis H. Martin for her technical assistance in the preparation of this paper.

NASA JSC NEURAL NETWORK SURVEY RESULTS

DAN GREENWOOD, NETROLOGIC, INC., 4241 Jutland Drive, San Diego, CA 92117

ABSTRACT

VERAC conducted a survey of Artificial Neural Systems in support of NASA's (Johnson Space Center) Automatic Perception for Mission Planning and Flight Control Research Program. Several of the world's leading researchers contributed papers containing their most recent results on Artificial Neural Systems. These papers were broken into categories and descriptive accounts of the results make up a large part of this report. Also included is material on sources of information on Artificial Neural Systems such as books, technical reports, software tools, etc. This paper is an abriged version of the report to NASA.

ACKNOWLEDGEMENT

VERAC gratefully acknowledges the contributions of all the participants of this survey and to Professor **Terence Smith** of the University of California at Santa Barbara and Dr. **Christopher Bowman** of VERAC for their advice concerning the approach to the survey and elucidating many subtle points concerning the subject matter. **Cris Kobryn** played a key role in obtaining survey data and analyzing public domain Artificial Neural Systems software. The foresight of our NASA sponsors, **Robert Savely** and **James Villareal**, in realizing the importance and promise of this exciting new field is greatly appreciated. **Sue Lani Andrassy** and **Susan Foster**, of VERAC, kept everything under control and imparted order where there was chaos.

1.0 INTRODUCTION

Artificial Neural Systems (ANS's) have captured the interest of many computer scientists, robotic engineers, mathematicians, and neurophysiologists as a result of progress made in solving problems which have eluded solution by conventional computer approaches. The aim of this study was to establish a database for NASA which contains the recent results of researchers in this rapidly growing and dynamic field. Since the field is so dynamic and the time to publish current research is often protracted, it was decided to broadcast an appeal for contributions of recent papers/reports or pre-prints to help form NASA's database and this survey. The response was overwhelming both from the standpoint of quantity and quality. So much so that the initial plan to perform a survey approaching the standards set by **Daniel Levine** [LEVINE] was soon abandoned and it was decided to:

1) Describe the papers as they appeared to us,
2) Not evaluate the papers or try to provide an integrated point of when different authors covered related ANS subject matter.

No effort was made to ascertain the accuracy of data or the validity of mathematics from any of the papers and, many of the papers were submitted as preliminary versions. The topics were broken down into the following areas:
1) ANS theory, 2) Computation and optimization, 3) Memory, 4) Learning, 5) Pattern recognition, 6) Speech, 7) Vision, 8) Knowledge processing, 9) Robotics/control, which reflects the format of the International Conference on Neural Networks.

The phrase, Artificial Neural Systems, was selected for this study over: connectionist models, parallel distributed processing, and neutral networks, although neural networks seems to be gaining the edge in terms of general acceptance and preference. Perhaps, it is not too late to introduce yet another word to encompass the same meaning associated with the above terms and even a little more. The word "Netrology" seems to be one which includes neural networks and possible expansions which supersede neural networks as they are commonly understood. It would encompass, for example, units which are not neuron-like in their behavior but which, nevertheless, exhibit interesting or useful properties. It seems that a key component belonging to networks falling under the concept of netrology should be that a network be able to learn; thereby circumventing the ordinarily difficult problem of programming on ensemble of parallel processors.

Based on reviewing the papers submitted in support of this study, the following issues are considered to be of importance to future progress in netrology:

1) A rigorous definition of "structure" or "regularity" which is often attributed to networks which discover features. Psychophysical measurements and fractal concepts (such as fractal dimension) will probably be necessary to define net "structure" rigorously.
2) The construction of netrological experiments and concepts which help to define ANS situational awareness, task management, and planning.
3) A rigorous definition of similarity corresponding to the efforts made in numerical taxonomy and classical pattern recognition so that net recall of "distorted" images, taxonomy and classical pattern recognition so that net recall of "distorted" images, etc. really corresponds to the goals of an application. Nets may have to be more or less discriminating per application.
4) The integration of sensory data from sensors of the same or different types (e.g., nets with three eyes and four ears) and a priori data concerning the environment and constraints.
5) Endowing nets with desirable human-like traits wuch as artificial modesty, humor, perseverance, honesty, etc. This will be of importance in merging net-workers with human workers in real world industrial, academic and military applications (user friendliness/congeniality is the goal here).
6) Establishing bounds of net autonomy. Asimov's robotic laws are anthropocentric. Future neural networks may look more kindly at us early designers if we make an effort to ensure their autonomy and provide the means for gratifying their creative instincts.
7) Training humans to be tolerant and accepting of net solutions to problems. Ohm was ridiculed for 30 years, and everyone knows about Galileo, so this issue is not as farfetched as it may seem.
8) Establishing a taxonomy of computational devices which shows which problem domains are best suited for systolic arrays, neural networks, symbolic processors, signal processors and conventional processors.
9) Establishing neural net design rules which facilitate configuring a neural net per problem application.
10) Establishing ANS figures of merit so the value of a particular learning rule or net design can be meaningfully estimated.

Considering issues 4 through 7 is certainly fun – now back to more immediate concerns. This ANS review contains an overview of the small but rapidly growing number of commercial ANS products, public-domain research tools, and some ANS books and educational materials. [LIPPMANN] contains one of the best short introductions as well as a penetrating analysis of ANS's as they are today, and, undoubtedly, the proceedings of the 1987 International Conference on Neural Networks will represent the state-of-the-art for ANS's when it is published.

2.0 GENERAL SOURCES OF INFORMATION ON ANS's

Since the recent re-birth of interest in ANS's, there has been a virtual flood of papers in engineering as well as scientific journals. So many technical papers on ANS's currently exist that are scattered among journals such as **Biological Cybernetics, Behavioral and Brain Sciences, Psychological Review,** and the **Journal of the**

National Academy of Sciences, that retrieving the papers alone is a time consuming and tediously difficult problem. Once a given paper is retrieved, it is then a major task to decipher the often new definitions, notation and technical style. For an engineer whose sole interest is to understand the potential of an ANS to solve a problem in pattern recognition or image understanding, the neurophysiological as well as the psychophysical flavor of many of the often-cited articles poses a major obstacle. The small number of books that exist for the most part are collections of papers submitted to technical journals or conferences. ANS courses are given at a few universities and there is a video tape of Dr. **Robert Hecht-Nielsen's** week-long course on ANS's. Training sessions are available both at TRW and HNC with the purchase of TRW's Mark-III Neurocomputer and HNC's ANZA Neurocomputer, respectively.

In view of the widely scattered and varied information on ANS's, this section will be devoted to describing that information on ANS's which is of a general, educational nature. The remaining sections are reserved for presenting the results of the survey based on the papers, reports and discussions generously provided by researchers throughout the world.

2.1 ANS Books

The books listed below are available for purchase and in many university libraries. There are, as yet, no textbooks on ANS's although the books: "Self-Organization and Associative Memory", by **T. Kohonen**, and "Parallel Distributed Processing", Volumes 1 and 2, by Rumelhart, McClelland and the PDP Research Group are semblances of textbooks.

2.1.1 Parallel Distributed Processing (Volumes 1 and 2), (MIT Press, 1986)

These volumes are the best introductory books to the field, the members of the PDP research group at the Institute for Cognitive Science at the University of California at San Diego, under the leadership of **James McClelland** and **David Rumelhart**, combined their talents to write (in various combinations of authors) both tutorial and research-oriented chapters on "Parallel Distributed Processing".

The history of ANS's is traced in a fair amount of detail and a wide range of related topics are covered. Basic mechanisms such as feature discovery by competitive learning, information processing in dynamical systems (Harmony Theory), Boltzmann machines, and back propagation are covered with many excellent examples.

2.1.2 Self-Organization and Associative Memory (T. Kohonen, Springer-Verlag, 1984)

This book was written before the days of back propagation and is mainly concerned with linear transformations. Even with these restrictions, it is a good source on adaptive filters, optimal associative mappings, and self-organizing feature maps. There is a good discussion of, with examples of topology preserving mappings but in general, many of the applications and alternative approaches in ANS's are not considered. The book complements the "Parallel Distributed Processing" book as a result of the extra attention to mathematical rigor and its linear systems perspective.

2.1.3 Parallel Models of Associative Memory (G.D. Hinton, J.A. Anderson, Lawrence Erlbaum Associates, 1981)

This book contains a collection of papers by well-known researchers in ANS such as **T. Sejnowski, S. Fahlman, G. Hinton**, etc. Topics covered are: models of information processing in the brain, a connectionist model of visual memory, holography, distributed associative memory, representing implicit knowledge, implementing semantic networks in hardware, and many other topics.

2.1.4 Neural Networks for Computing (G.S. Denker, Editor, American Institute of Physics, 1986)

This book contains 64 short papers by leading ANS researchers. The papers encompass applications, mathematical theory, implementations, and biological modeling. A paper by **Lapedes** and **Farme** presented an interesting method for circumventing the limitations of a Hopfield Network. Another paper by **Personnaz**, et al. introduces a simple modificationto Hebbian learning to give a more biologically plausible selectionist learning scheme.

2.1.5 Brain Theory: Proceedings of the First Trieste Meeting on Brain Theory, 1984 (G. Palm, A. Aertsen, Editors, Springer-Verlag)

"Brain Theory" contains papers by researchers primarily concerned with the workings of the brain itself and, secondarily, with methods for defining and exploiting information processing principles obtained along the way to understanding brain operations.

2.1.6 Competition and Cooperation in Neural Nets (S. Amari, M. Arbib, Editors, Springer-Verlag, 1982)

The proceedings of a 1982 conference on neural nets are presented in this book. Leading neural net theoreticians and brain theorists such as **S. Grossberg, M. Arbib, A. Pellionisz, T. Kohonen, S. Amari** presented papers at the conference.

2.1.7 The Adaptive Brain (Stephen Grossberg, Editor, North Holland, 1987)

Professor Grossberg and members of the Center for Adaptive Systems at Boston Universith (which Grossberg leads) wrote the papers for this highly theoretical book. Chapters of the book cover: psychophysiological theory of reinforcement, drive motivation, and attention, psychophysiological and pharmacological correlates of a developmental cognitive and motivational theory, conditioning and attention, memory consolidation, a neural theory of circadian rhythms, and other topics.

2.2 Reports

2.2.1 How the Brain Works: The Next Generation of Scientific Revolution (by David Hestenes, Third Workshop on Maximum Entropy and Bayesian Methods in Applied Statistics, University of Wyoming, Aug 1-4, 1983)

Professor Hestenes, a mathematician from Arizona State University, was persuaded by a former student (now Dr. **Robert Hecht-Nielsen** of HNC, Inc.) to spend some time and hard work getting familiar with the work of **Stephen Grossberg**. Hestens came away from his efforts as a firm believer in Grossberg's approaches and outlined the basis for his beliefs in a tutorial report dedicated exclusively to Grossberg's work.

2.2.2 Neural Network Models of Learning and Adaptation (J.S. Denker, AT&T Bell Laboratories, N.J.)

This Bell Labs technical report provides a good overview of neural network basics. Hopfield's ideas are clearly presented as are discussions of simple Hebbian learning, ADALINE, Geometric and Pseudo-Inverse rules, and the practical effects of clipping. The report ends with an interesting presentation of open questions in neural network theory.

2.2.3 Performance Limits of Optical, Electro-Optical, and Electronic Neurocomputers ("Optical and Hybrid Computing", SPIE, Vol. 634, 1986)

Hecht-Nielsen did an excellent job of summarizing neural network theory and implementations up to 1986. He covered ANS modeling philosophy, technology organization, theory, neurocomputers and their performance limits, the Cohen/Grossberg Adaptive Resonance Network Learning Theorem, Hopfield's and Kohonen's theories and their implications for implementation issues.

2.2.4 Neural Population Modeling and Psychology: A Review (D. Levine, Mathematical Biosciences, 66: 1983)

Professor Levine's excellent review if highly recommended for anyone interested in neural networks including those with either theoretical or applications oriented interest. This well written review addresses all of the major neural assembly models from 1938 to 1983. The works of **Grossberg, Barto, Sutton, Klopf, Anderson, Uttley, von der Malsburg**, and others are presented in a tutorial fashion and the significance of the respective models in relation to neurophysiological and psychological data is addressed in detail.

2.2.5 Stochastic Interated Genetic Hillslimbing (D. Ackley, March 1987, CMU-CS-87-107, Carnegie-Mellon University)

David Ackley's PhD dissertation contains a new method for performing function optimization in high dimensional binary vector spaces. The method can be compactly implemented in a neural network architecture and provides an effective network training rule which combined genetic search algorithms properties with hillclimbing algorithm properties.

2.2.6 A Survey of Artificial Neural Systems (P. Simpson, Unisys, San Diego)

Patrick Simpson of Unisys reported the results of a survey of ANS's in [SIMPSON, abs]. This survey, completed in early 1987, discusses some of the well known neural models and contains computer codes for different learning rules (Hebbian, Hopfield, Boltzmann) and recall rules. Many of the ANS's such as the Sejnowski/Rosenberg NETtalk, are described. A brief history of ANS's is also included.

2.2.7 Efficient Algorithms with Neural Network Behavior (S. Omohundro, April 1987) Report No. UIUCDCS-R-87-133, Department of Computer Science, University of Illinois at Urbana-Champaign)

Although this report is not concerned with ANS's per se, it does discuss alternatives to ANS approaches and, in so doing, sheds light on the capabilities and properties of ANS's. Using hierarchical data structures well known in computer science (arrays, hashing, tries, trees, adaptive grids) Omohundro was able to show very significant implementation advantages in solving problems where ANS's are now being applied. In explaining why his data structure based approach is in many cases much more efficient than corresponding ANS approaches, Omohundro claims:

1) ANS's must evaluate each neuron's activity and consider the effect of each weight each time an input is made, while the "algorithm approach" only looks at stored values along a path of logarithm depth.
2) ANS learning requires that all weights be updated with each input, but data structures only modify parameters where regions are relevant in determining the output on the given input.

2.3 Other Sources of General ANS Information

The following media provide additional sources of ANS information:

2.3.1 Neuron Digest

Subscribers to ARPANET can avail themselves of neural network information on upcoming lectures, publications, conferences, abstracts, government research grants, opinions and needs. The **Neuron Digest** is distributed each month and is a great way to keep informed about this rapidly growing field.

2.3.2 A Video Tape on Artificial Neural System Design

A video tape of **Dr. Robert Hecht-Nielsen's** 5-day course on ANS design is available through HNC, Inc. or the University of California at San Diego through the University Extension. The course is an excellent way to get introduced to the state-of-the-art of ANS's by one of the field's leading experts and educators. The full spectrum of ANS's from deep theoretical issues (Grossberg, et al.) to practical ANS implementations are covered.

2.3.3 HNC, Inc. Month-Long Course on ANS's

A hands-on course in ANS is included in the purchase price of an HNC Neurocomputer (the ANZA). The course emphasizes the practical aspects of applying ANS methods to problems in sensor processing, knowledge processing, control, optimization, data base management and statistical analysis. The course is aimed at enabling students to become productive ANS experts in a short period of time.

2.3.4 TRW Mark-III Neurocomputer Training

TRW's ANS Center, headed by **Michael Myers**, provides a one-week training with the purchase of a TRW Mark-III Neurocomputer. The very powerful machine with a staff experienced at solving problems (much beyond the text-book variety of most applications of ANS's) provides an effective way for ANS's users to augment their skills.

3.0 IMPLEMENTATIONS OF ARTIFICIAL NEURAL SYSTEMS

The resurgence of interest in ANS's and critical evaluations of their potential have resulted in new commercial enterprises whose charters are to bring ANS products to market. There also exist activities in many businesses aimed at developing hardware and software available commercially and under development by members of the commercial sector.

3.1 Currently Available Commercial ANS Products

The following companies sell ANS products and may be contacted directly to obtain literature containing ANS product descriptions.

3.1.1 HNC (Hecht-Nielsen Neurocomputer Corporation)

HNC is a San Diego based company founded by **Robert Hecht-Nielsen** and **Todd Guschow** who developed the Mark series Neurocomputers while at TRW. HNC's main products are the ANZA Neurocomputer and the ANZA basic "Netware" (neurocomputer software) package. A month-long course is offered and course participants are expected to have a basic knowledge of college level mathematics, and problems of interest to them will be addressed in the course.

The Netware packages are loaded into the neurocomputer (in combinations of one or more), and their constants and parameters tuned and selected by the user to fit the application problem at band.

3.1.2 Nestor, Incorporated

Nestor is a publicly traded ANS company which currently sells two products developed by nobel laureat, **Leon Cooper**, and Brown University physics professor, **Charles Elbaum**. The company is located in Providence, Rhode Island. ANS methods used in current Nestor products are also being considered for postal sorting, robotic mission systems, fingerprint and voice identification, speech and speaker identification, medical diagnostic systems, check processing and encoding and credit card identification/validation.

3.1.3 Neuraltech, Incorporated

Dr. John Voevodsky founded Neurotech, Incorporated and sells a software product called "PLATO/ARISTOTLE". The software package is a neural-based expert system for the IBM PC-AT and COMPAQ 286/386 personal computers.

3.1.4 Texas Instruments

Andrew Perry and **Richard Wiggins** of Texas Instruments developed a digital signal processor for accelerating neural network simulations [DENKER]. Using TI's Odyssey boards, they developed a system which was forty times faster than the VAX 8600 for ANS applications.

3.1.5 TRW

TRW sells a neurocomputer called the Mark-III and provides a one-week on-site (San Diego, CA) course in the purchase price. TRW researchers **Michael Myers** and **Bob Kuczewski** are defining the state-of-the-art in applications of ANS methods to signal processing, spatial temporal pattern learning, classification of time varying spectrograms, and image analysis.

3.1.6 SAIC's Σ-1 Neurocomputer

SAIC announced a Neurocomputer, called the "Σ-1", which is expected to be available in October of 1987. The machine was developed by a SAID research team headed by **Dr. James Solinsky**, a renown computer vision researcher. the Σ-1 software includes shells for most well known learning rules.

3.2 Currently Available Public Domain ANS Products

At the present time software ANS simulation packages can be obtained without charge from Brown University and the University of Rochester for use by ANS researchers.

3.2.1 The Brown University ANS Simulation

Professor **James A. Anderson** of Brown University released an ANS simulator based on his "Brain State in a Box" neural network model. The software was developed over the last 12 years and continues to be a useful tool for ANS experimentation.

3.2.2 The University of Rochester Simulation

The University of Rochester reported on two ANS software packages. One is intended to be executed in a BBN Butterfly Multi-Processor [FANTY] and another can be executed on either a VAX minicomputer (with UNIX) or the Sun Microcomputer.

3.3 ANS's and Components Being Developed but Not Currently Available

Several high technology companies and laboratories are sponsoring internal research and development programs aimed at producing commercial ANS's or ANS components. The following subsections given an overview of such activities for some of the companies initiating product oriented research programs.

3.3.1 AT&T Bell Laboratories

Bell labs has a neural network working group with **H. Graf, L. Jackel, J. Denker,** et al. as members. This group has successfully implemented 54 neurons and 54 input channels with 3000 synapses (interconnects) connecting the input channel with each neuron. Standard CMOS was used in the implementation. Design work for a 256 neuron-chip and 512 neoron-chip has been completed. In addition, an associative memory with an analog processor and digital I/O was reported. Details of the chip designs are given in [GRAF, JACKEL, et al.].

Joshua Alspector and Robert Allen of Bell Communications Research described a VLSI implementation of a modified Boltzmann Machine in [ALSPECTOR]. Their paper contains a short review of Pitts formed neuron, ADALINE, Hopfield's model, the Boltzmann Machine and back-propagation.

3.3.2 IBM

Under Dr. **Claude Cruz's** leadership, IBM's Palo Alto Research facility has developed a "Network Emulation Processor (NEP) with an IBM PC/XT host computer and professional graphics adapter yields a workstation to interactively design, debug, and analyze networks.

3.3.3 California Institute of Technology and the University of Pennsylvania

P. Mueller of the University of Pennsylvania and **J. Lazzaro** of the California Institute of Technology have assembled an ANS of 400 analog neurons for analyzing and recognizing acoustical patterns (including speech) [MUELLER]. Up to 100,000 interconnects can be made and synaptic gains and time constants are determined by plugging in

resistors and capacitors. The system currently performs adequately for well articulated phonemes and diphones. New energy consonants present problems but it is expected that different coding schemes and better understanding of the invariant clues for speech perception will lead to improvements.

3.3.4 **Hughes Research Laboratories** (Malibu, CA)

Researchers **G. Dunning, E. Maron, Y. Owenchko,** and **B. Soffer** at Hughes Research Laboratories have developed and tested an all optical nonlinear associative memory using a hologram in an optical cavity formed by phase conjugate mirrors [SOFFER]. They were able to store multiple superimposed images and reconstruct a complete image, inputting only a portion of the stored image.

3.3.5 **UCSD Analog Back Propagation System**

Stan Tomlinson, a graduate student at UCSD, discovered a method for increasing the speed of a back propagation ANS. Tomlinson defined a "continuous time back propagation ANS" where the forward pass, backward pass, and weight modifications are performed simultaneously.

3.3.6 **Oregon Graduate Research Center**

Dan Hammerstrom and his colleagues at the Oregon Graduate Research Center are in the process of designing a water-scale integrated silicon system that implements a variety of ANS's [HAMMERSTROM]. In their investigations of implementations of the back propagation learning rule, their simulations show that nodes can compute asychronously and that the rule is robust enough to accommodate incomplete information on each learning cycle.

3.4 **ANS Properties**

K.L. Babcock and **R.M. Westervelt** of Harvard University investigated the stability and dynamics of electronic neural networks with added inertia and reported their results in [BABCOCK]. Babcock and Westervelt reviewed the Hopfield ANS model and then expanded it by adding an inertial term to the rate equations.

A. Guez, V. Protopopescu, and **J. Barnden** of the Oak Ridge National Laboratory Studies the stability, storage capacity and design of nonlinear results in [GUEZ]. They determined sufficient conditions for the existence and asymptotic stability of any ANS's equilibrium. The conditions take the form of a set of piecewise linear inequality constraints solveable by a feedforward binary network or other methods such as Fourier Elimination.

4.0 ANS MODELS AND APPLICATIONS

Despite essentially continuous research and development since the introduction of computers in the late 1940-s, the connection between brain processing and computer processing is still undergoing theoretical development. No attempt is made at forming a complete integrated view of the large amount of material of the different ANS topics covered in the papers and reports from the survey participants. Time restrictions of the study prevent undertaking the very important tasks of interpreting, evaluating, and integrating the many excellent and undoubtedly important contributions by ANS researchers. These crucial tasks remain to be done in the future.

It is difficult to make a clear distinction between brain theory and ANS (neural network or connectionist) theory since ANS researchers typically proceed by abstracting the essence of a theory aimed at explaining results obtained from neurophysiological or psycho-physical experiments and then derive a method for designing processors (electronic or optical) which can obtain the same or nearly the same experimental results. [FRISBY] contains an excellent and well illustrated discussion of Poggio's and Marr's approaches at modeling steroptics, and his book serves to indicate the basic practice of going from pure brain theoretic modeling to computer based ANS models. It is generally accepted that no completely satisfactory brain theory exists although some models, such as von der Malsburg's model of the visual cortex, exhibit behavior that experiments confirm. In spite of sometimes incomplete theories, ANS researchers often attempt to develop real world applications of any plausible theories in areas such as vision, speech recognition, etc. Many of the current ANS theories basing their roots in brain research are rich enough in information theoretic content to have, so to speak, lives of their own. The current successful ANS model seems to be one reflecting human-like information processing capabilities, implementable in some computational device which provides reasonable results in close to real-time. The present conventional AI impasse reached by attempts at computer vision and artificial intelligence so ably described in [LERNER] and [DREYFUS] leaves no other alternative than the ANS based approaches presented in the papers described below. [von der Malsburg] Contains many modern brain models.

4.1 **Brain Theory Applicable to ANS's**

The question of how much brain theory is enough to enable the design of processors which can produce acceptable human (or animal)-like processing is very difficult to answer. It is also very difficult to ascertain the level of detail required of a model: is it sufficient to characterize the average behavior of assemblies of neurons as in much of Grossberg's work or are individual cells behavior required? How to choose between and extract the "essential" properties of Grossberg's, Freeman's, Edelmans's and Reeke's models, for example is not at all clear. Recent ANS/computer theory history compels modern computer scientists and robotic engineers to achieve at least some familiarity with the different brain theories represented in this subsection.

4.1.1 **Nonlinear Dynamics with Chaotic Solutions (Laboratory Measurements and Models)**

Walter Freeman, Christine Skarda, and **Bill Baird** developed models of the formulation and recognition of patterns in the rabbits olfactory bulb. Baird gave an excellent overview of neural modeling which relates the well known ANS models to results from laboratory measurements [BAIRD]. Baird simplified Freeman's ANS model of the rabbit's olfactory bulb while capturing the essence of the pattern formation/recognition behavior. A key definition for Baird's work is "pattern formation": the emergence of macroscopic order from microscopic disorder. Freeman, Skarda, and Baird define dynamical systems which have chaotic behavior (fractal solutions) for ground states as opposed to fixed point attractors, as in Simulated annealing or the Hopfield model. It is speculated that such ground state behavior is essential for real-time continuous perception.

Freeman argues, very congently, that neural dynamic system destabilization provides the best description of the essentials of neural functioning, and Baird finds that the mechanism of competing instabilities (nonlinear mode selection) is implicit in dynamical associative memories and provides the key ingredient in pattern recognition. While admitting that their neural dynamical models have many similarities with well known connectionist models, they point out significant dissimilarities essential for recognition and discrimination. Freeman and Baird's models are unique in that they possess dense local feedback between neuron assemblies. Such feedback is necessary to generate chaotic and limit cycle system ground states.

Gail Carpenter and **Stephen Grossberg** developed a neural network model for mammalian circadian rhythms [CARPENTER] which can have chaotic solutions for some system parameter ranges. Their model explains many phenomena in mammal behavior such as the role of eye closure during sleep and the stability of the circadian period.

4.1.2 **Brain Models of Knowledge, Conditioning, Perception, and Learning Processing**

Stephen Grossberg and **Ennio Mingolla** in their paper [GROSSBERG, MINGOLLA] show how computer simulations can be used to guide the development of neural models of visual perception. They use cooperation/competition mathematical models to simulate textual segmentation and perceptual grouping as well as boundary completion.

Daniel Levine, the brain theorist who authored the very extensive and well written neural modeling review [LEVINE] also recently contributed two papers to brain theory. In "A Neural Network Model of Temporal Order Effects in Classical Conditioning", Levine demonstrated in a computer simulation that one of Grossberg's neural networks can reproduce the experimental findings that the strength of a conditioned response is an inverted U-function of the time interval between conditional and unconditioned stimuli. The network also can reproduce blocking of a neutral stimulus by another stimulus that has been previously conditioned. Levine also traces the history of conditioning models in this paper and reviews Grossberg's theories [LEVINE, 1986].

A very novel model of cortical organization was invented and investigated by **Gordon Shaw** and colleagues [SHAW] at the University of California at Irvine. Their model was motivated by **V.B. Mountcastle's** organizational principle for neocortical function and **M.E. Fisher's** model of spinglass systems. Their network is composed of interconnected "trions", units which have three possible states (−1, 0, +1) which represent firing below background, at background, and above background respectively. Trions represent a localized group of neurons and symmetrical interaction between trions exhibit behavior where hundreds of thousands of quasi-stable periodic firing patterns exist and any can be selected out and enhanced, with only small changes in interaction strengths, by using a Hebbian-type of algorithm.

Dana Ballard presented his local representation ideas for modeling the cerebral cortex in a stimulating paper in the Behavioral and Brain Sciences [BALLARD]. A large part of the interest associated with the paper resulted from the peer commentary following the paper (members in the Veer group included Grossberg, Hopfield, Edelman). In the paper Ballard presented his local representation model and value unit concept. He included methods for perceiving shape and motion by exploiting his model.

John Hopfield, whose famous ANS for solving the traveling salesman problem, generated a great amount of interest and enthusiasm for, once again, applying ANS's to real world problems, also performed theoretical and experimental work on the nervous system of the Limax maximum (A terrestrial mollusk). In [HOPFIELD] a description is given of behavioral and neurophysiological attributes of Limax learning. A model which also includes a memory network is related to experimental data and predictions are made.

George Hoffmann presented some very novel ANS concepts in a paper which explored analysis between the brain and the immune system [HOFFMANN]. His model involved a neuron with hysteresis which eliminated the need for learning with modifiable synapses. The Hoffmann ANS learns through interacting with the environment and being driven to regions in phase-space. The system has 2N attractors for N neurons.

In their paper "Selective Neural Networks and Their Implications for Recognition Automation" [REEKE], **George Reeke** and **Gerald Edelman** observed that the models of

McCulloch and Pitts, Marr, Hopfield, Hinton and Anderson, and McClelland and Rumelhart all deal with the mechanisms for acquiring and processing information, but not with the ways that the categories of information and the mechanisms to process them come to exist. As a consequence, Reeke and Edelman devised a model which exploits basic biological principles (Darwin's principles of natural selection) to explain the discovery of perceptual categories, the representation of categories without pre-arranged codes, the manipulation of retrieval keys, and the selection of actions on the basis of imperfect and inconsistent information without a program.

A. Pellinisz and **R. Linas** wrote many papers on the last ten years where *"Tensor Network Theory" is introduced* [PELLIONISZ]. *Pellionisz asserts that the conventional* representations of McCulloch, Pitts, Kohonen, Hinton and Anderson are expressed in extrinsic, orthogonal systems of coordinates and represented coordinates in Euclidean vector space. He finds that new important insights into the control nervous system are gained by describing the central nervous system in terms of intrinsic coordinates using tensor analysis.

Michael Jordan continued the tradition of excellence in ANS research associated with the UCSD Institute for Cognitive Sciences with the publication of his paper 'Serial Order: A parallel Distributed Processing Approach" [JORDAN]. ANS trajectories (attractors) follow desired paths as a result of learning with constraints which generate the required serial order. His report contains an ANS tutorial section, an approach to the coarticulationtion problem in speech, and examples from various simulations.

David Zipser, also from UCSD's Institute for Cognitive Science, authored two reports dealing with problems of the representation of spatial entities [ZIPSER]. He described a map retrieval mechanism based on an ANS and illustrated mapping tasks such as recognition of previously visited locations, path finding using landmarks, and finding a path between two locations that do not share landmarks. He also points out similarities between ANS features associated with map representation, and known features of the hippocampus.

James McClellang and **David Rumelhart** presented a very thorough exposition of the distributed models of memory and learning and compared their model to other well known models from cognitive science (MCCLELLAND]. They point out that their distributed model is capable of storing many different patterns, determining the central tendency of a number of different patterns, create perceptual categories without using labels, and capturing the structure inherent in a set of patterns with or without prototype characterization.

Jerome Feldman of the University of Rochester performed a theoretical analysis of the behavior of connectionist mode, in [FELDMAN]. He analyzed ANS's using energy concepts such as the Hopfield, Hinton, and Sejnowski, and Smolensky models and pointed out that such approaches are most relevant for problem domains lacking significant structure and questioned the utility of such approaches in highly structured cognitive domains (such as compiling or parsing). He also discussed the relevance of automation theory and control theory to ANS formalization.

In a paper entitled "On Applying Associative Networks", submitted to the IEEE First Annual Conference on Neural Networks, **A.D. Fisher** treated approaches to formulating basic organizational principles for mapping problems onto associative processors. He addressed goal directed learning and structures for knowledge representation, and configuring a simulation environment for evaluating and developing the organizational principles.

4.2 Computation/Optimizatin

The computational generality of ANS's and the ability of ANS's to solve some interesting optimization problems are now widely known [DENKER]. In the last few years some interesting papers appeared which view ANS's from complexity theory, computational, and mathematical perspectives and which strive to characterize ANS's in more classical systms theoretic manner. This subsection describes these types of papers from various contributors to the survey.

Ian Parberry and **Gearo Schnitger,** of the University of Pennsylvania, investigated the relationship between boltzmann Machines and conventional computers [PARBERRY]. They concentrated on determining the computing power of Boltzmann machines which are resource bounded. They measured machine running time and hardware requirements as functions of problem size. They found that:

1) The connection graph can be made to the acyclic (no feedback loops)
2) Random behavior can be removed from the machine
3) All synapse weights can be made equal to one.

These properties make the machines equivalent to a combinatorial circuit and the resulting machine will have its running time increased by a constant multiple and its hardware requirement increased by a polynomial.

Terence Smith, Omar Egecioglu, and **John Moody** analyzed computational complexity issues in ANS's such as the generalize feed-forward networks, perceptrons, and Hopfield devices [EGECIOGLU]. For each ANS type they describe programming the ANS, functions computable by the ANS, complexity issues, and the architecture. They point out the need for the expansion of traditional computer science to include dynamical systems and statistical mechanics.

Pierre Baldi of UCSD (formerly of the California Institute of Technology) established an upper bound for generalized Hopfield type models. That is, models with energy functions or Hamiltonians of degree d:

$$H(x) = \sum_{i_1 \cdots i_d}^{N} T_{i_1,\ldots,i_d}\, x_{i_1} \ldots, x_{i_d},$$

Based on counting arguments they established a storage capacity upper bound of $O(N^{d-1})$ for ANS's with dynamics which minimize H(x). Baldi points out that these higher order systems have local updating rules as in the quadratic (Hopfield) case and they recur very naturally in optimizing problems [BALDI].

John Hopfield, of the California Institute of Technology, and **David Tank,** of AT&T Bell Laboratories, introduced a new conceptual framework and a minimization principle which provide increased understanding of computation in neural circuits [HOPFIELD]. They derive a model abstracted from knowledge of biological neurons and discuss how their model dispenses with many known properties of neurons while still capturing those aspects necessary for performing computations essential to organism adaptations and survival. The classical model of the neural dynamics is taken as a point of departure and Hopfield and Tank show that the assumption of interconnect (weight) symmetry is not overly restrictive and that many functions such as edge detection, stereoptics, and motion detection can be cast as optimization problems and solutions result from the convergence of symmetric dynamic neural systems.

Morris Hirsch, a mathematician from the University of California, investigated convergence in neural nets [HIRSCH]. He discussed the Liapunov functions discovered by Cohen and Grossberg and the connection to Hopfield's results. He was able to remove some restrictions on the state equations (neuronic equations) that show that important convergence properties still hold.

Harold Szu, of the Naval Research Laboratory, presented a method for speeding up the conventional simulating annealing algorithm [SZU]. Szu's non-convex optimization method, called a Cauchy machine, is derived from the Boltzmann machine by using the Cauchy/Lorentzian probability density function in place of the Gaussian probability density function. It is shown that the "cooling" schedule varies inversely with the time versus the inverse logarithm of time for conventional simulated annealing.

4.3 Memory in ANS's

A large body of ANS research is devoted solely to the modeling of human or biological memory (see [LEVINE] for a review). This section presents descriptions of recent work on ANS memory and is broken into theory, applications, and implementation categories.

4.3.1 ANS Memory Theory

Despite the extensive amount of research, there is still no complete and universally accepted theory of human or biological memory. However, as in other categories of ANS modeling, many useful models of memory have been identified in the pursuit of devising an accurate theory of biological memory. The work on the theory of memory immediately below is representative of modern theoretical advances in ANS memory theory.

David Rumelhart and **Donald Norman** of the UCSD Institute for Cognitive Science are important contributors to the development of parallel distributed ANS's and they also did important work in Artificial Intelligence. They published an ICS report on memory, "Representation in Memory," which gives an excellent overview on AI approaches in the theory of human memory. Though ANS's are not addressed in this report, it is, nevertheless, recommended for gaining insights into the issues concerning the representation of knowledge. They discuss spreading activation in semantic networks and discuss the ideas of Fahlman and Anderson.

Stephen Grossberg and **Gregory Stone** devised models of the effects of attention switching and temporal-order information in short term memory. In their paper, "Neural Dynamics of Attention Switching and Temporal-Order Information in Short-Term Memory" [GROSSBERG, STONE], they argue that attention-switching influences initial

storage of items in short term memory, but competitive interactions among representations in short term memory control the subsequent dynamics of temporal-order information as new items are processed.

Teuvo Kohonen summarized his current views on the theory of memory in a paper titled, "Self Organization, Memorization, and Associative Recall of Sensory Information by Brain-like Adaptive Networks" [KOHONEN]. Kohonen asserts that the two main functions of memory are: 1) to act as mechanisms which collect sensory information and transform it into various internal models or representations, and 2) to interrelate the signal processes in these representations and store them as collective state changes of the neural network.

In 1984, **Pentti Kanerva** published his PHD thesis entitled "Self-Propagating Search: A Unified Theory of Memory" [KANERVA]. His dissertation introduces the sparsely distributed memory model and the concept of using a neuron as an address decoder for accessing memory. Kanerva's memory model overcomes limitations in the Hopfield memory model such as dependence of storage capacity or the number of neurons (.14N, N = # of neurons), the inability to store temporal sequences, symmetric interconnects, and a new limited ability to store correlated inputs.

The dissertation includes background material on related ideas by Marr and Kohonen and includes mathematical estimates of convergence rates and memory capacity. Kanerva further elaborated upon his memory model in a paper, "Parallel Structures in Human and Computer Memory," and discussed the application of his ideas to the "frame" problem of Artificial Intelligence and showed that the part of the problem concerned with manipulating vast quantities of data about the real world can be handled with his spare distributed memory concepts.

James Keeler of UCSD compared Kanerva's model with Hopfield's model in a Research Institute for Cognitive Science Paper: "Comparison between Sparsely Distributed Memory and Hopfield-Type Neural Network Models", [KEELER]. In this very well written and mathematically rigorous paper, Keller developed a mathematical framework for comparing the two patterns. Keeler extended Kanerva's sparse distributed memory model and showed that Hopfield's model was a special case of this extension. Keeler showed that Kanerva's model corresponds to a three layer network with the middle layer consisting of many more neurons and that Kanerva's formulation allows context to aid in the retirevel of stored information.

Alan Lapedes and **Robert Farber**, of the Theoretical Division of Los Alamos National Laboratory, reported on a new method for designing a content addressable memory which is free of major limitations associated with the Hopfield content addressable memory [LAPEDES]. Their method consists of dividing a network into two groups of neurons. One group, called the Master net function, basically has a Hopfield optimization network while the other group, called the Slave net can have asymmetric connections. Advantages associated with this master/slave formulation are: 1) two bases of attraction may be merged together, 2) weighting of certain components of a fixed point so that it attracts more strongly (sculping a basis of attraction), and 3) biologically plausible division of neurons into excitatory and inhibitory sub-groups.

Tarig Samad and **Paul Harper** of Honeywell used back-propagation in linear array of fully connected units to construct a content addressable memory (SAMAD]. They cite several advantages in their method over the Hopfield content addressable memory: 1) asymmetric weights, 2) their method is guaranteed to recognize stored patterns, 3) close to perfect recall if a retrieval cue is not very far from any stored memory, 4) up to 2^N patterns can be stored (N = number of neurons and bits in the patterns), 5) 20% perturbations in learned weights and thresholds effected performance by less than 1%, and 6) robustness in degradation of weights, learning rates, and stored pattern cue hamming distance.

Demetri Psaltis and **Cheol Hoon Park** of the California Institute of Technology, designed an associative memory with a quadratic disriminant function [PSALTIS]. Their neural net memory can be shown to have a capacity proportional to N^2 where N is the number of bits in a storage sector. The square law nonlinearity is conducive to an optical implementation. The added capacity can be combined with the shift invariant property of an optical correlator to yield a shift invariant associative memory.

Santosh Venkatosh and **Demetri Psaltis** of the California Institute of Technollgy, discovered an associative memory which uses the spectrum of a linear operator [VENKATOSH]. They showed that their method has a capacity which is linear in the dimension of the state space while that of the outer-product method has a capacity asymtote of $n/(4 \log n)$. Their method requires full connectivity and, consequently, is more suitable for an optical implementation. The larger storage capacity of the "spectral" method is paid for with increased pre-processing cost.

4.3.2 **Applications of ANS Memory**

An interesting application of an ANS memory was devised by Michael Mozer of the Institute of Cognitive Science as UCSD [MOZER]. Mozer's ANS performed inductive information retrieval. The ANS retrieval system takes dynamic use of the internal structure of test databases to infer relationships among items in the database.

The inferred relationship helped the system overcome incompleteness and imprecision in request for information as well as in the database. The ANS used neuronic equations from McClelland and Rumelhart's interactive activation model of word perception. The model handles queries in a document retrieval application by activating a set of descriptor units and seeing which document units become active as a result.

4.3.3 **Implementations**

Shift Invariant Optical Associative Memory implementations were analyzed by **D. Psaltis, J. Hong,** and **S. Venkatesh** in [PSALTIS]. They found that without special encoding techniques associative memories with linear interconnections did not retrieve shifted images well. Two systems, one with a square law interconnection and the other with a novel encoding scheme, were found to be shift invariant and to achieve the performance of the outer product method.

Arthur Fisher, Robert Fukuda, and **John Lee** discussed the implementation of parallel processing architectures consisting of multiple optical adaptive associative modules [FISHER]. The modules adaptively learn and store a series of associations in the form of electronic charge distributions in an optical control device termed a micro-channel spatial light modulator. The optical adaptive associative modules have a gated learning capability, where adaptivity is easily switched on or off. The associative modules have an accumulative learning capability where even one exposure to an associated pair of vectors produces a weak association and subsequent exposures improves to the optimum pseudo-inverse solution.

4.4 **Learning**

It is well known that progress in Artificial Neural Systems came to an abrupt halt shortly after the publication of the Minsky and Papert treatise; "Perceptrons." The ability to program a multilayered network proved to be a very stubborn problem until the work of Kohonen, Rumelhart, Parker, Hopfield, Barto, and Sutton showed how such nets could be programmed. This section describes the recent results in ANS learning.

In this monumental PhD dissertation, **David Ackley** describes a multi-dimensional space search strategy which combines hill climbing methods and search methods based on genetic algorithms [ACKLEY]. Ackley's method called "Stochastic Iterated Genetic Hillclimbing", has a coarse-to-fine search strategy as in the case for simulated annealing and genetic algorithms but the convergence process is reversible. That is, in the implementation it is possible to diverge the search after it has converged and recover coarse-grained information about the space that was suppressed during convergence. Successful optimization typically has a series of converge/diverge cycles.

Ronald Williams, formerly of the Institute for Cognitive Science at UCSD and currently with Northeastern University, investigated a class of algorithms designed to allow the self-organization of feature mappings in ANS's [WILLIAMS]. Williams introduces a measure termed "faithfulness" which is intended to measure how well an input pattern can be constructed from knowledge of an output pattern. The algorithm he devised for feature detection maximizes the faithfulness measure. Williams defines an algorithm called "Symmetric-error correction" and he proves that if a system is completely linear and symmetric of rank R greater than m, the number of output units, then when the algorithm converges the resulting weight vectors are of unit length, orthogonal, and span the space spanned by n eigenvectors having the largest eigenvalues.

Andrew Barto of the University of Massachusetts, discusses an ANS based on the concepts introduced by **Harry Klopf** [BARTO]. Barto's ANS is called A_{R-P} (associative reward-penalty) and has a learning rule which adjusts weights according to four types of information: 1) presynaptic signals, 2) postsynaptic signals, 3) a reinforcement signal from the environment that reflects the consequences of a neuron's activity, and 4) a signal that indicates what a neuron usually does for a given stimulas pattern.

Barto shows how the A_{R-P} ANS solves the "exclusive or" problem as well as other non-trivial problems. He also includes a very informative comparison of "associative reinforcement learning" (to which A_{R-P} corresponds), supervised and unsupervised learning and points out subtle distinctions between these basic learning types. He argues, for example, that unsupervised learning is more accurately regarded as supervised learning with a fixed built in teacher. Theoretical convergence and comparative analyses of A_{R-P} are also included in the report.

Gail Carpenter and **Stephen Grossberg** collaborated on paper on associative learning, adaptive pattern recognition, and cooperative-competitive decision making by neural networks [CARPENTER]. The paper contains a discussion of the "univeral theorem" (proved elsewhere) which show how arbitrarily many cells computing arbitrary transfer functions can interact asynchronously through complex nonlinear feedback and experience no learning bias resulting from cross-talk of their feedback signals – a result called "absolute stability". There are also discussions of feature discovery, category learning, adaptive pattern recognition, and Adaptive Resonance Theory (ART). ART is an ANS which self-organizes its recognition code and the environment can also modulate the learning process and, as a result, carry out a teaching role.

Harry Klopf discussed drive reinforcement learning in a paper for the IEEE First Annual International Conference in Neural Networks (June 1987) [KLOPF]. Klopf's learning model uses signal levels and changes in signal levels in such a way to yield an unsupervised learning which predicts classical conditioning phenomena such as delay and track conditioning, stimulus duration and amplitude effects, second-order conditioning, extinction as well as other phenomena. In his model, sequentiality replaces simultaneity and is an extension of the Sutton-Barto model (1981).

D.G. Bounds, of the Royal Signals and Radar Establishment, performed an analysis of Boltzmann Machines and reported the results in [BOUNDS]. Bound's paper contains a discussion of the Boltzmann machine algorithm and the encoder problem for the task of communicating information between components of a parallel network. An extensive simulation was conducted which assessed the effect of temperature on learning rate, and a comparison between the Boltzmann Machine Hamiltonian and the Sherrington-Kirkpatrick Spin-glass Hamiltonian.

W.S. Stornetta and **B.A. Huberman** of the Xerox Palo Alto Research Center presented an analysis of the back-propagation algorithm in [STORNETTA, ICCN,87]. They capitalized on the fact that if an input is zero there will be no modification to the weights extending out from that unit and, consequently, only half of the weights from the input to the hidden unit layer will be changed. The back-propagation algorithm was modified so that the dynamic range of all units was (-1/2, 1/2) rather than (0,1). The input and output patterns consists of series of -1/2's and 1/2's and the squashing function is given by:

$$-1/2\ [\exp(-\ w_{ij}0_j\ +\ bias_i)\ +\ 1]^{-1}$$

and the rest of the conventional back-propagation algorithm remains unchanged.

Bart Kosko, of VERAC Corporation, devised a new learning method associated with Bidirectional Associated Memory (BAM) and described its ANS properties in [KOSKO]. This very well written and informative paper contains a review of classical associative memory theory and a tutorial section on BAM. An earlier proof by Kosko that energy matrix is bivalently bidirectionally stable is reviewed and BAM correlation encoding is discussed.

Continuous BAM's, introduced in Kosko's earlier work, are reviewed and a proof that every matrix is continuously bidirectionally stable is given. BAM learning, the main interest of the paper, is achieved by programming the BAM connection matrix to adapt according to a generalized Hebbian learning law where adaptive resonance occurs, that is, neurons and interconnections quickly reach equilibrium. The connection weight learning law is called the Signal Hebb law and is given by:

m_{ij} = m_{jj} + $S(a_j)S)b_j)$

A_j, B = activation

S( ) = sigmoid function

Adaptive BAM's are characterized as to their classification properties and it is shown that they converge to **local** energy minima. In addition, the capabilities of the adaptive BAM are compared and contrasted to Grossberg's Adaptive Resonance Theory (ART) ANS.

Kosko extended adaptive BAM's to competitive adaptive BAM's by including lateral-inhibitory interactions [KOSKO]. When field inhibitory connections are taken to be symmetric, stability can be shown, although it was found that in practice, non-symmetric within field connections exhibit stability in many cases.

Variations on the Grossberg adaptive resonance model were explored by **R.W. Ryan** and **C.L. Winter,** of Science Applications International Corporation and reported in the Proceedings of the ICNN (1987). Ryan and Winter found that the adaptive resonance model may activate a coded recognition node whose top-down pattern bears little resemblance to the corresponding input pattern and there is no guarantee that the adaptive resonance circuit (ARC) will reset and, hence, the ARC will be recoded by the input pattern unless the set of weights is prevented from changing.

4.5 Pattern Recognition

Bill Baird of the Department of Biophysics at the University of California, reported the results of an analyses of pattern recognition in the olfactory bulb of the rabbit [BAIRD]. Baird simplified Freeman's model by neglecting synaptic delay and did not perform a full simulation of the rabbits olfactory system but achieved universality as a result. His mathematical analysis explains how an oscillating system can pattern recognize. Baird conducted experiments with an array of 64 electrodes which yielded EEG patterns which showed the emergence of order from disorder and indicated that specific EEG patterns are correlated with specific recognition responses. A theory which combines the mathematical description of the emergence of order by instability with the mathematics of associative memory is required to model learning and memory in neural networks.

Robert Hecht-Nielsen, of HNC, showed how banks of matched filters can be used as pattern classifiers for complex spatiotemporal pattern environments such as speech, sonar, radar, and communication [HECHT-NIELSEN]. He defiend an ANS, called a "simple avalanche matched filter bank" which closely approximates the theoretical classifier to be introduced. He defined a "nearest" match filter, discussed its error rate, and observed that such classifiers can only carry out the first "local in time" stage of pattern recognition and that context must be expoited to achieve high performance pattern recognition.

Paul Gorman, of Bendix Aerospace and **Terrence Sejnowski,** of John Hopkins University, collaborated on a paper which discussed the application of back-propagation to classify sonar targets [GORMAN]. The signal representation used for input to the network was selected as the result of experiments with human listeners. A short term Fourier transform using 60 frequency samples per temporal slice was generated for each signal. By starting at the onset of the signal and increasing the position of the time slice, approximately a linear FM/chirp, essentially formed the diagonal of a 60 x 60 time/frequency matrix. Normalized values from the matrix served as input to the matrix. Results from experiments indicated that the network is able to discover target classification features from examples of sonar signals with performance comparable to human experts.

Results on applying back-propagation to handwritten numerical recognition and spoken numeral recognition were reported by **D.J. Burr** of Bell Communications Research [BURR]. Burr gave tutorial explanations of related geometric hyperplane analysis and back-propagation learning in addition to thorough treatments of recognition of handwritten and spoken numerals. For the handwritten numerals a two-stage process of normalization followed by feature extraction was used. It was found necessary to subtract a constant representing signal level from all feature vectors. Removing the D.C. component in this was dramatically increased the learning rate of the network. The neural networks were configured with up to 64 hidden units, but it was found that a maximum recognition score around 98% occurred with 6 and 14 hidden units for the written and spoken recognition tasks respectively. Burr's results compared favorably to nearest neighbor pattern recognition applied to the same problem.

Alan Kawamoto and **James Anderson** extended Anderson's Brain State-in-a-Box (BSB) model to define a new ANS which models multistable perception [KAWAMOTO]. The BSB extension allows the ANS to shift between hyperspace box corners in a way corresponding to multistable perception. After explaining the analytical properties of their new ANS, Kawamoto and Anderson showed how their ANS qualitatively agrees with published psycho-physical results on multi-stable perception regarding bias, adaptation, hysteresis, and dynamics. Most of the results discussed concern visual ambiguities but speculations concerning lexical ambiguity resolution are presented also.

4.6 ANS's Applied to Vision

Maureen Gremillion, Arnold Mandell and **Bryan Travis** reported on their design and implementation of a neural net model of the mammalian visual system in [TRAVIS]. These researchers from Los Alamos National Laboratory modeled a scaled down version of primary visual cortex, the lateral geniculate nucleus, and a 45,000 neuron retina. Research types are embedded in two-dimensional layers and differentiated cell types are distributed in space. Several different retina models from the visual modeling literature were tested.

Kunihiko Fukushima, of NHK Science and Technical Research Laboratories in Japan, discussed a multilayered hierarchical ANS called the "Neocognitron", in [FUKUSHIMA]. The Neocognitron consists of alternating layers of feature-extracting neurons and neurons which are fed outputs from feature-extracting neurons and fire only if one input is active. Inhibitory cells exist to suppress irrelevant features. The Neocognition is capable of supervised or unsupervised learning and can recognize shifted or deformed variations of a pattern. Excessive distortion results in windowed recognition response. A Neocognition with selective attention is currently being investigated by Fukushima.

Ronald Williams, of Northeastern University, investigated the ability of self-organizing networks to infer spatial relations [WILLIAMS]. Williams was motivated by the dipole pattern neural network experiments conducted by Rumelhart and Zipser where, without a priori knowledge of the spatial layout of the input, an ANS learned apsects of input spatial structure. He mathematically formalized the notion that if two patterns are strongly correlated then they must be nearer in some metric than if their values are weakly correlated. The concept of a spatial relationship or pattern element is formalized in terms of a distance metric on pairs of elements and the notion of an environment with spatial structure is formalized as a random field over a metric space.

Richard Golden, of Brown University, developed an ANS which models the process of visual perception of a letter in the context of a word. Interconnections between neurons represent any spatially or sequentially redundant and transgraphemic information in displays of letter strings. Golden's model uses Anderson's BSB model and enhancements derived from commonly accepted principles of information processing in the central neurons system. In Golden's model, a word is represented as a pattern of neural activity over a set of position-specific feature neurons.

Image restoration involves removing degradations from images arising from blur from optical aberrations, atmospheric turbulence, motion, defraction, and noise. The application of ANS methods to image restoration is currently being investigated by **Y.T. Zhou, R. Chellapa** and **B.K. Jenkins** of the University of Southern California [ZHOU]. They designed an ANS containing redundant neurons to restore gray level images degraded by shift invariant blur function and noise.

Ralph Linsker, of the IBM Thomas J. Watson Research Center, addressed the origin and organization of spatial-opponent and orientation-selective neurons in ANS's based on biologically plausible roles for development [LINSKER]. Linsker treats the emerging of network structures from spontaneous electrical activity and simple biologically based rules for synaptic modification.

Ralph Siegel, currently of Rockerfeller University and formerly of the Salk Institute, analyzed the abilities of Rhesus monkeys and human subjects to detect the change in three-dimensional structure of cylinder using only motion cues [SIEGEL]. Siegel implemented a three layer ANS with 100 neurons in the input layer homologous to the neurons in the middle temporal area of the brain. Each input neuron was tuned for a given velocity at a retinotopic location and corrected to all ten units in the middle layer. The middle ten units were connected to only one neuron in the output layer which indicated structure as constructural motion.

J.M. Oyster, W. Broadwell and **F. Vicuna** of the IBM Los Angeles Scientific Center, investigated the application of associative metworks to robot vision in [OYSTER]. The approach taken to robotic vision by these IBM researchers was to show how the well known methods for robotic vision, such as image acquisition, segmentation, object recognition, etc., can be implemented in an associative network. They rigorously convert a discrete convolution to an associative network and then argue that the standard low-level edge-detection operators, such as the Roberts, Sobel, and Laplacian and Gaussian operators can be implemented as ANS's.

Christof Koch, Jose Marroquin and **Alan Yuille** demonstrated how Hopfield ANS models can be generalized to solve nonconvex energy functionals corresponding to functions of early vision such as computing depth from two stereoscopic images, reconstructing and smoothing images from sparsely reconstruction can be formulated in terms of minimizing a quadratic energy function and, subsequently, show how quadratic variational principles fail to detect image discontinuities. An energy function containing cubic terms is defined which is shown to handle discontinuities.

G. Cottrell, P. Munro and **D. Zipser** applied back-propagation to compressing images [COTTRELL]. Current (1987) image compression techniques are briefly treated as in back-propagation. Image compression is considered to be a type of encoder problem in that an identity mapping over some set of inputs must be performed An ANS is forced to perform the mapping of a narrow channel of the network and thereby causes an efficient encoding. Two noteworthy facts are: 1) the network developed a compact representation of its environment, 2) although the algorithms were developed as supervised learning schemes, the problem really involves learning without a teacher since the input and output are identical, i.e., the ANS self-organizes to encode the environment. Most of their image compression results were obtained with a three layer network: G4 inputs, 16 hidden units and 64 outputs.

Stephen Grossberg developed a theory of vision which offers an explanation of the coherent synthesis of three-dimensional form, color, and brightness percepts [GROSSBERG]. The theory identifies several uncertainty principles that limit the extraction of visual information. Particular modules are resolves by hierarchical parallel interactions between many processing stages. Grossberg asserts that when a neural processing stage removes one type of uncertainty from any input pattern it often generates a new type of uncertainty which is passed to the next processing stage. That is, information is not progressively reduced in a succession of neural processing stages. Based on results of monocular boundary segmentation and feature filling-in and the interaction between these processes, Grossberg suggests that the commonly accepted hypothesis of *independent modules in visual perception is wrong and misleading.*

Harold Szu and **Richard Messner** derived multiple-channel novelty filters of associative memory from a retina ANS point-spread function [SZU]. They present a novelty filter as a remainder operator and mathematically drive the multiple-channel model from associative memory formulation. Their ANS is shown to be scale and rotation invariant and simulation results are cited. They also point out a new relationship between adaptive novelty filtering and adaptive associative memory.

Michael Mozer of the Institute for Cognitive Science at UCSD, investigated early parallel processing in reading [MOZER] and developed an ANS capable of recognizing multiple words appearing simultaneously on an artificial retina. The ANS is called "BLIRNET" since it builds location-independent representations of multiple words. BLIRNET is a multilayered hierarchical network which learns via back-propagation.

4.7 ANS's for Speech and Language

Bryan Travis, of the Los Alamos National Laboratory, described a layered ANS model of sensory cortex in [TRAVIS]. The goal of Travis' research program was to construct a model of the human sensory system which reflects what is known structurally and physiologically at several levels from the ear to the midbrain nuclei to the cortex. Travis made simplifications in scale to accommodate current computer technology constraints but claimed that his model contained the following desirable features: 1) more structure (based on neurophysiological data) than previous models, 2) inputs based on biological data, 3) emphasizes dynamics, and 4) provides a means of testing theories about sensory perception.

D.W. Tank and **J. Hopfield** developed an analog ANS capable of solving a general pattern recognition problem for time-dependent input signals [TANK]. In order to solve such sequence recognition problems, Tank and Hopfield expanded the Hopfield ANS's Energy function to include time dependence. In the case of a time-dependent energy function, a convergent computation can occur if the problem's data produce a channel on the space-time surface which guides the circuit trajectory to a position corresponding to a correct solution. A key concept for applying a time-dependent Hopfield model is the using of a set of delay filters as a sequence detector. An energy function is defined which, when presented with a known sequence builds a deep pit on the space-time energy surface with a wide valley leading to it.

An analysis of the hidden structure of speech was performed by **Jeffrey Elman** and **David Zipser** of the Institute of Cognitive Science at the University of California at San Diego [ELMAN]. Using back-propagation, Elman and Zipser taught an ANS, a series of speech recognition tasks, and after examining the resulting ANS internal representations found that the representations often corresponded to known speech representational units such as: diphones, context-sentence allophones, phonemes, syllables, and morphemes.

Michael Meyers, Robert Kuczewski and **William Crawford** of TRW's AI Center in San Diego, ran experiments to investigate ANS self-organization and temporal compression methods using Lenglish (a form of artificial speech) based on text from a childrens' book [MEYERS]. They report that the development of a self-organization ANS which concurrently performs dimension reduction, pattern recognition, and new pattern learning. Pre-processing consisted of a linear shift invariant KCM transform and using such a stabilized high dimensional vector time series the ANS learns hierarchical features and their correlations.

Tariq Samad of Honeywell, Incorporated, described an application of back propagation to determine the correct set of features corresponding to words in an input sentence [SAMAD]. Samad states that human cognitive functions such as the acquisition of concepts, tolerance of error and noisy input, graceful degradation have eluded solutions in traditional computer approaches but can be solved as side-effects with ANS's. He reviewed previous related work on parsing, case-role assignment, and word-sense disambiguation and related his ANS to recent work by McClelland and Kawamoto.

1) The ANS outputs, an association of features with input words where the ANS learned concepts such as "proper noun", "animate-common-noun", and "inanimate-common-noun".
2) The ANS outputs were as in 1) concepts such as "plural" and "read", the grammar was also extended preferred associations.
3) More connections were made with central (in a window) words than off-center words to enable preferred associations.

K. Torkkola, H. Rittinen and **T. Kohonen** reported on the results of a microprocessor-based word recognizer for a large vocabulary (1000 words) in [TORKKOLA]. Their system is capable of phonemic recognition using an ANS in the form of a phonotopic map. The map consists of a two-dimensional array of processing units which constitute matched filters to different phonemes. Each unit is tuned to a particular acoustical spectrum and the spectral templates of the units have a distribution which corresponds to the optimal clustering of the various phonemes. Word recognition is performed by comparison of phonemic transcriptions with reference transcriptions stored in a dictionary.

M. Cohen and **S. Grossberg**, of the Center for Adaptive Systems at Boston University, presented a computational theory explaining how an observer parses a speech stream into context-sensitive language representations in [COHEN, abs]. Cohen and Grossberg's theory stress the real time dynamical interactions that control the development of languages as well as learning and memory. Properties of the performance of language result from an analysis of the system constraints governing stable language learning. The process whereby internal language representations encode a speech stream in a context-sensitive fashion are analyzed. Cohen and Grossberg also show how organizational principles, important for visual processing, can be applied to language processing and, thus, a similar model can be used for spatial processing as well as temporal processing.

Stephen Grossberg and **Gregory Stone** presented models of the neural dynamics of word recognition and recall in [GROSSBERG]. A major goal of this paper was to synthesize the many experiments and models of human language processing and to show that learning rules and information rules are intimately connected. Grossberg and Stone maintain that to understand word recognition and recall data, it is necessary to analyze the computational units that subserve speech and language and it is also necessary to consider how computational units acquire behavioral memory by reacting to behavioral inputs and generating behavioral outputs. Furthermore, they assert that explanations of hidden processing assumptions that go into a model along with tests of their plausibility and ability to arise through self-organization are required. Models such as the Logogen model, verification model, and the Posner and Snyder model are reviewed using concepts such as automatic activation, limited capacity, attention, serial search, and interactive activation.

4.9 **Knowledge Processing**

Lokendra Shastri, currently of the University of Pennsylvania, investigated evidential reasoning in ANS's for his PhD thesis at the University of Rochester [SHASTRI]. The following two issues were considered by Shastri to be crucial to making progress in knowledge representation:

1) The necessity of identifying and formalizing inference structures that are appropriate for dealing with incompleteness and uncertainty. An agent cannot maintain complete knowledgte about any but the most trivial environments, and therefore, he must be capable of reasoning with incomplete and uncertain information.
2) The importance of computational tractability. An agent must act in real-time. Human agents take a few hundred milliseconds to perform a broad range of intelligent tasks, and we should expect agents endowed with artificial intelligence to perform similar tasks in comparable time.

Shastri regards these two issues to be intimately related and stressed that computational tractability is not solely concerned with efficiency, optimizing programs, or faster machines. The main issue, he believes, is to establish the existence of a computational account of how an ANS may draw valid conclusions within time constraints which a given environment allows. Shastri contends that the full power of parallelism can be exploited only if it is taken, as an essential premise used to guide searches for interesting problem solutions in the space of possible knowledge representation frameworks, as opposed to finding serial solutions and subsequent parallel implementations.

John Barnden of Indiana University, devised an abstract computational architecture that can embody complex data structures and associated manipulations [BARNDEN]. Barnden addresses many of the issues raised in an NSF sponsored 1986 workshop on ANS's (MCCLELLAND, J., FELDMAN, J., BOWER, G., and MCDERMOTT, D., "Report of a Workshop on Connectionism Instigated by NSF, 1986]. In particular, an opinion expressed at the workshop that "Connectionism has not yet shown its adequacy for dealing with complex – perhaps deeply nested-representations and connectionist problems concerning crosstalk, explosive proliferation, binding and control, are addressed by Barnden in his exposition of his architecture. Barnden's architecture is based on two-dimensional arrays called "configuration matrices" which contain positioned occurrence of basic symbols.

James A. Anderson of Brown University described cognitive capabilities such as concept formation, inference, and guessing associated with the "Brain-State-in-a-Box" (BSB) ANS are nearly identical to those used by Hopfield for continuous valued systems. Information handling capabilities were characterized as follows: 1) Poor handling of precise data, 2) Inefficient use of memory in a traditional serial computer, 3) Several parameters must be tuned, and 4) Outputs may be distorted (or incorrect).

Anderson contends that the above information processing deficiencies are the cost that must be paid for the advantages obtained from ANS's, like BSB when applied to knowledge databases.

James Anderson, Richard Golden and **Gregory Murphy** discussed an ANS with a Hebbian learning rule in [ANDERSON]. They showed that the Brain-State-in-a-Box (BSB) model is similar to a gradient descent algorithm and how the behavior of many ANS's can be viewed in a probabilistic framework. They further showed that Hebbian learning and autoassociative Widro-Hoff learning can be considered to be ways of estimating the form of an associated probability density function of the form:

$$P(x) = k \exp(-(x^T Ax)/2)$$

where "A" = a weight matrix and the pdf can adequately represent any arbitrary pdf of binary-valued stimulus vectors that may occur n the model's environment.

Concepts in distributed representations, implications of error correction for concept formation, retrieval, redundancy, and disambiguation by context were also discussed in the paper.

Stephen Grossberg, of Boston University and **William Gutowski,** of Merrimack College, analyzed the neural dynamics of decision making under risk [GROSSBERG]. This paper contains a review of models of human decision making under risk. Utility theory and prospect theory are discussed and model deficiencies are pointed out. Prospect theory, for example, does not account for non-rational decision theory such as preference reversal where in a binary choice situation an individual prefers an alternative which has been judged to be worth less than the nonpreferred alternative. A new theory of decision making under risk, called affective balance theory, is described which is an application of previous Grossberg theories of how cognitive and emotional processes interact. The previous theory was used to model perception, attention, motivation, learning, and memory. Affective balance theory is based on psychophsyiological mechanisms and processes derived from analyzing relevant data.

Data Ballard of the University of Rochester, investigated a completely parallel connectionist inference mechanism based on energy minimization [BALLARD]. Ballard used a relaxation algorithm to produce inferences in first order logic derived from a very large knowledge base. He showed that for first order predicate calculus formulas and inferences rules a proof producer (resolution) can be uniquely expressed as a neural network with a very simple form. The implementation of first order logic constraints yield two coupled networks, namely, a clause network that represents clause syntax and a binding network representing relationships between terms in different clauses.

Michael Cohen and **Stephen Grossberg** of the Center for Adaptive Systems at Boston University, discussed a network with very extensive capabilities in hypothesis formation, anticipation, and prediction [COHEN]. The network is called a "masking field" and is a multiple scale, self-similar, cooperative-competitive feedback network with automatic gain control. Cohen and Grossberg discuss context sensitive grouping in recognition processes, cognitive rules arising from network interactions, and how a masking field possesses predictive anticipatory or priming capabilities. Their analytic arguments contain several references to simulation results obtained in their research on masking fields.

Claude Cruz, William Hanson and **Jason Tom** of the IBM Palo Alto Scientific Center, described the research activities in knowledge processing in [CRUZ, abs]. They introduced a conceptual scheme for representing knowledge and for performing inferences called "Knowledge Representation Networks" [KRN]. The network consists of a set of knowledge entities (KE's) having varying states of activity. It is postulated that only three basic KE's are required to produce more complex KE's, namely: 1) features, 2) relationships between features, and 3) operations. A KRN inference results from a change in state of a KE which leads to a change in state of another KE. A KRN contains five mechanisms which results in knowledge processing: 1) **Implication:** a forward-chaining evidential reasoning mechanism; 2) **Context:** the KRN's current state affects subsequent sensory processing; 3) **Goal-decomposition:** a forward chaining mechanism reducing the operations to sets of coordinated sub-operations; 4) **Goal-biasing:** a mechanism which enables events to trigger operations in a production-rule-like manner; and 5) **Expectations:** a mechanism which enables operations to be executed in a closed loop position using knowledge of events expected to occur as the operation is executed.

4.10 **Robotics/Control**

Stephen Grossberg presented an analysis of adaptive sensory-motor control in [GROSSBERG]. The analysis considered the developmental and learning problems that an ANS (also real brain system) must solve to enable accurate performance in a dynamic real-world environment. The main emphasis of Grossberg's sensory-motor control analysis was on visually guided motor behavior and his results are considered to be relevant to issues such as localization, orienting, sensorimotor interfacing, and the design of motor pattern generating circuitry. Grossberg illustrates general organizational principles and mechanisms through analyzing the mammalian saccadic eye movement system. The issue of infinite regress, where changes in serial subsystems can interact to undo changes in other subsystems, was addressed by introducing attentionally mediated interesting sensory cues.

Geoffrey Hinton of Carnegie-Mellon University and **Paul Smolensky** of the Institute for Cognitive Science at the University of California at San Diego, analyzed mass-spring model of motor control using a neural-network [HINTON]. ANS methods which can solve the problems of finding necessary torques and generating internal representations of desired trajectories for reaching movements of arm and body are identified. Hinton and smolensky comparatively analyzed many conventional methods of robotic control and found, for example, that desired final robot configurations via using length-tension characteristics to set end points is ineffective control. They also discussed Raibert's massive memory control table approach and the Luh-Walker-Paul sequential control algorithm.

5.0 REFERENCES

1. David H. Ackley, Stochastic Iterated Genetic Hillclimbing, CMU-CS-87-107, Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213
2. Joshua Alspector and Robert B. Allen, A Neuromorphic VLSI Learning System, Proc. of the 1987 Stanford Conf. on Adv. Res. in VLSI
3. James A. Anderson, Richard M. Golden, and Gregory L. Murphy, Concepts in Distributed Systems, S.P.I.E. Institute on Hybrid and Optical Computing, Dept. of Psychology, Brown University, Providence RI 02912
4. James A. Anderson and Gregory L. Murphy, Psychological Concepts in a Parallel System, Physica D 22 318-331, North-Holland, Amsterdam, Dept. of Psychology & Center for Cognitive Science, Brown University Providence, RI
5. James A. Anderson, Disordered Systems and Biological Organization, Nato Advanced Research Workshop
6. K.L. Babcock and R.M. Westervelt, Stability and Dynamics of Simple Electronic Neural Networks with Added Inertia, Physica 23D (1986) 464-469 North-Holland, Amsterdam
7. K.L. Babcock and R.M. Westervelt, Complex Dynamics in Simple Neural Circuits, Harvard University
8. Bill Baird, Nonlinear Dynamics of Pattern Formation and Pattern Recognition in the Rabbit Olfactory Bulb, Physica 22D (1986) 150-175, North-Holland, Amsterdam, Department of Biophysics, University of California, Berkeley, CA 94611
9. Pierre Baldi, Neural Networks, Orientations of the Hypercube and Algebraic Threshold Functions, IEEE Transactions on Information Theory, Dept. of Mathematics, University of California, San Diego, La Jolla, CA 92093
10. Pierre Baldi and Santosh S. Venkatesh, Number of Stable Points for Spin-Glasses and Neural Networks of Higher Orders, Physical Review Letters Vol. 58, Number 9, 1 March 1987
11. Dana H. Ballard, Cortical Connections and Parallel Processing: Structure and Function, The Behavioral and Brain Sciences (1986) 9, 67-120, University of Rochester, Rochester, NY 14627
12. Dana H. Ballard, Cortical Connections and Parallel Processing: Structure and Function, TR 133 (revised) January 1985, University of Rochester
13. John Barnden, Complex Cognitive Information-Processing: A Computational Architecture with a Connectionist Implementation. TR 211, December 1986, Indiana University, Bloomington, IN 47405-4101
14. Andrew G. Barto, Learning by Statistical Cooperation of Self-Interested Neuron-like Computing Elements, COINS Technical Report 85-11 April 1985, University of Massachusetts, Amherst, MA 01003
15. D. G. Bounds, Numerical Simulations of Boltzmann Machines, Proc Neural Network for Computing, Snowbird, Utah 1986
16. D. J. Burr, Matching Elastic Templates, Proceedings of the Royal Society Symposium on Physical and Biological Processing of Images, London, September 27-29, 1982
17. D. J. Burr, Experiments with a Connectionist Text Reader, Bell Comm. Res., 1987
18. D. J. Burr, A Neural Network Digit Recognizer, Bell Comm. Res.
19. David J. Burr and Stephen Jose Hanson, Knowledge Representation in Connectionist Networks, Bell Comm. Res.
20. Gail A. Carpenter and Stephen Grossberg, Associative Learning, Adaptive Pattern Recognition, and Cooperative-Competitive Decision Making by Neural Networks Source: Hybrid and Optical Computing, SPIE 1986
21. Gail A. Carpenter and Stephen Grossberg, Northeastern University, Boston, MA 02115 and Boston University, A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine, "Computer Vision and Image Processing", 37 (1987)
22. Gail A. Carpenter and Stephen Grossberg, Mammalian Circadian Rhythms: A Neural Network Model, Lectures on Mathematics in the Life Sciences, Vol. 19, 1987
23. Patrick Castelaz, John Angus, and Joan Mahoney, Application of Neural Networks to Expert System and Command & Control Systems, Hughes Aircraft, Fullerton, CA
24. Michael A. Cohen and Stephen Grossberg, Masking Fields: A Massively Parallel Neural Architecture for Learning, Recognizing, and Predicting Multiple Goupings of Patterned Data, Applied Optics, in press, May, 1987, Center for Adaptive Systems, Boston University, Boston, MA 02215
25. Garrison W. Cottrell, Paul Munro, David Zipser, Image Compression by Back Propagation: An Example of Extensional Programming, UCSD Institute for Cognitive Sciences
26. William Crawford, Michael Myers, Robert Kuczewski, Application of New Artificial Neural System Information Processing Principles to Pattern Classification, TRW (Rancho Carmel, San Diego) Technical Memo
27. Claude A. Cruz, William A. Hanson, Jason Y. Tam, Knowledge Processing Through Flow-Of-Activation, To Appear in Proceedings of 1987 IEEE First Annual International Conference on Neural Networks IBM Palo Alto Scientific Center, Palo Alto, CA 94304
28. Claude A. Cruz, William A. Hanson, Jason Y. Tam, Neural Network Emulation Hardware Design Considerations, To appear in Proceedings of 1987 IEEE First Annual International Conference on Neural Networks IBM Palo Alto Scientific Center, Palo Alto, CA 94304

29. M. Cohen and S. Grossberg, Neural Dynamics of Speech and Language Coding: Developmental Programs, Perceptual Grouping, and Competition for Short-Term Memory, Human Neurobiology (1986)5:1-22
30. John S. Denker, Neural Network Models of Learning and Adaptation, AT&T Bell Laboratories, Holmdel, NJ 07733
31. G. Ed. Denker, Neural Networks for Computing, American Institute of Physics, 1986, Proceedings 151.
32. H. Dreyfus, From Socrates to Expert Systems: The Lilmits of Calculative Rationality, Bulletin of the Academy of Arts and Sciences, January 1987, No. 4, Vol XL.
33. Omer Egecioglu, Terence R. Smith and John Moody, Computable Functions and Complexity in Neural Networks, Computer Science Dept, U. of Cal. Santa Barbara, Santa Barbara, CA
34. Jeffrey L. Elman, James L. McClelland, Exploiting Law Variability in the Speech Wave, For MIT Conference on Invariance and Variability in Speech Processes October 8-10, 1983
35. Jeffrey L. Elman, Connectionist Approaches to Acoustic/Phonetic Processing, Dept. of Linguistics, University of California, San Diego
36. Mark Fanty, A Connectionist Simulator for the BBN Butterfly Multiprocessor, TR 164 January 1986, U. of Rochester
37. Jerome A. Feldman, Connectionist Models and Parallelism in High Level Vision, Computer Vision, Graphics, and Image Processing 31, 178-200 (1985)
38. Jerome A. Feldman, Energy and the Behavior of Connectionist Models, TR 155, November 1985, U. of Rochester
39. Jerome A. Feldman, Neural Representation of Conceptual Knowledge, TR 189 June 1986, U. of Rochester
40. A. D. Fisher, On Applying Associative Networks, Submitted to: IEEE First Annual International Conference on Neural, Networks, San Diego, CA June 1987.
41. Arthur D. Fisher, Robert C. Fukuda, John N. Lee, Implementations of Adaptive Associative Optical Computing Elements, SPIE, Vol. 625 Optical Computing (1986)
42. W. J. Freeman, Simulation of Chaotic EEG Patterns with a Dynamic Model of the Olfactory, System Biol. Cybern. 55 (1987)
43. Walter J. Freeman, M.D., Hardware Simulation of Brain Dynamics in Learning: The SPOCK, Submitted for the IEEE 1st Annual Intl. Conf. on Neural Networks, San Diego, CA 21-24 January 1987
44. Walter J. Freeman, Semi-Autonomous Control of Input by Nerve Cell Assemblies Through Genesis of Images, CH1935-6/83, 1983 IEEE, U. of Calif., Berkeley
45. Frisy, Seeing, Oxford Unitersity Press.
46. Kunihiko Fukushima, Neocognitron: A Brain-Like System for Pattern Recognition, Denshi Tokyo No. 25(1986) NHK Science & Technical Research Laboratories 1-10-1, Kinuta, Setagaya, Tokyo 157, Japan
47. Kunihiko Fukushima, Sei Miyake and Takayuki Ito, Neocognitron: A Biocybernetic Approach to Visual Pattern Recognition, NHK No. 336 September 1986 (ISSN 0027-657X), NHK Science & Technical Research Laboratories 1-10-11, Kinuta, Setagaya, Tokyo 157, Japan
48. Kunihiko Fukushima, A Neural Network Model for Selective Attention in Visual Pattern Recognition, Biol. Cybern. 55, 5-15(1986)
49. Kunihiko Fukushima and Takayuki Ito, A Neural Network Model Extracting Features from Speech Signals, The Transactions of the Institute of Electronics, Information and Communication Engineers, Japan, Vol. J70-D, No. 2, pp. 451-462
50. Kunihiko Fukushima, Neocognitron: A Brain-Like System for Pattern Recognition, Denshi Tokyo (published by the Tokyo Section, IEEE) No. 25 (1986)
51. Richard M. Golden, A Developmental Neural Model of Visual Word Perception, Cognitive Science 10, 241-276(1986)
52. Richard M. Golden, The "Brain-State-in-a-Box" Neural Model Is a Gradient Descent Algorithm, Reprinted from Journal of Mathematical Psychology Vol. 30, No. 1, March 1986
53. Leslie M. Goldschlager, A Computational Theory of Higher Brain Function, Stanford Computer Science Report
54. R. Paul Gorman, Terrence J. Sejnowski, Learned Classification of Sonar Targets Using a Massively-Parallel Network, Proc. of the Digital Signal Processing Workshop (1986) Sponsored by: IEEE Acoustics, Speech and Signal Processing Society Rev. 4/26/87
55. H.P. Graf, L.D. Jackel, R.E. Howard, B. Straughn, J.S. Denker, W. Hubbard, D.M. Tennat, and D. Schwartz, VLSI Implementation of a Neural Network Memory with Several Hundreds of Neurons, AT&T Bell Laboratories, Holmdel, NJ 07733
56. Stephen Grossberg, Cortical Dynamics of Three-Dimensional Form, Color, and Brightness Perception: I. Monocular Theory, Perception & Psychophysics 1987, 41 (2), 87-116
57. Stephen Grossberg and Ennio Mingolla, Computer Simulation of Neural Networks for Perceptual Psychology, Behavior Research Methods, Instruments, & Computers 1986, 18 (6), 601-607
58. Stephen Grossberg, Cortical Dynamics of Three-Dimensional Form, Color, and Brightness Perception: II. Binocular Theory, Perception & Psychophysics 1987, 41 (2), 117-158

59. Stephen Grossberg and Ennio Mingolla, Neural Dynamics of Surface Perception: Boundary Webs, Illuminants, and Shape-From-Shading, Source: from "Computer Vision and Image Processing", 37, (1987)
60. Stephen Grossberg and William E. Gutowski Neural Dynamics of Decision Making Under Risk: Affective Balance and Cognitive-Emotional Interactions Source: Psychological Review, in press, 1986
61. Stephen Grossberg Cooperative Self-Organization of Multiple Neural Systems During Adaptive Sensory-Motor Control Source: Center for Adaptive Systems, Boston University, Boston, MA 02215
62. Stephen Grossberg and Gregory Stone Neural Dynamics of Attention Switching and Temporal-Order Information in Short-Term Memory Memory & Cognition 1986, 14 (6), 451-468
63. A. Guez, V. Protopopescu, J. Barhen, On the Stability, Storage Capacity and Design of Nonlinear Continuous Neural Networks, Prepared by the Oak Ridge Natl. Laboratory, Oak Ridge, TN 37831
64. Dan Hammerstrom, Casey Bahr, Jim Bailey, Gary Beaver, Kevin Jagla, Norman May, A Development Environment for Wafer-scale Integrated Silicon Neuron-computers, Oregon Graduate Research Center, 1987
65. Dan Hammerstrom, Jim Bailey, and Mike Rudnick, Interconnect Architectures for WSI Neurocomputers, Oregon Graduate Research Center, 1987
66. Robert Hecht-Nielsen, Performance Limits of Optical, Electro-Optical, and Electronic Neurocomputers, HNC 5893 Oberlin Dr., San Diego, CA 92121, SPIE Optical and Hybrid Computing, Vol. 634, 1986
67. Robert Hecht-Nielsen, Nearest Matched Filter Classification of Spatiotemporal Patterns, HNC Hecht-Nielsen Neurocomputer Corporation
68. David Hestenes, How the Brain Works: the Next Great Scientific Revolution, Presented at the Third Workshop on Maximum Entropy and Bayesian Methods in Applied Statistics (U. of Wyoming, Aug. 1-4, 1983)
69. Geoffrey Hinton and Paul Smolensky, Parallel Computation and the Mass-Spring Model of Motor Control, Institute for Cognitive Science, U. of Calif., San Diego
70. Morris W. Hirsch, Convergence in Neural Nets, Dept. of Mathematics, U. of California, Berkeley, CA 94720
71. Geoffrey W. Hoffmann, A Neural Network Model Based on the Analogy with the Immune System, Dept. of Physics & Microbiology, University of British Columbia, Vancouver, B.C., Canada V6T 2A6
72. Geoffrey W. Hoffmann, Maurice W. Benson, Geoffrey M. Bree and Paul E. Kinahan, A Teachable Neural Network Based on an Unorthodox Neuron, Physica 22D (1986) 233-246, North-Holland, Amsterdam
73. J. J. Hopfield, A. Gelperin and D. W. Tank, The Logic of Limax Learning, AT&T Bell Laboratories, Murray Hill, NJ 07974 and (AG) Dept. Biology, Princeton Univ., Princeton, NJ 08544, (JJH) Div. of Chemistry and Biology, Calif. Institute of Tech., Pasadena, CA 91125
74. John J. Hopfield and David W. Tank, Computing with Neural Circuits: A Model, Articles 8 August 1986
75. W. Hubbard, D. Schwartz, J. Denker, H.P. Graf, R. Howard, L. Jackel, B. Straughn, D. Tennant, Electronic Neural Networks, AT&T Bell Laboratories, Holmdel, NJ 07733
76. B.A. Huberman, W. Scott Stornetta, An Improved Three-Layer, Back Propagation Algorithm, ICNN 1987, pre-print
77. B.A. Huberman, T. Hogg, Adaptation and Self-Repair in Parallel Computing Structures, Vol. 52, Number 12, Physical Review Letters, 19 March 1984
78. James M. Hutchinson and Christof Koch, Simple Analog and Hybrid Networks for Surface Interpolation, "Neural Networks for Computing", ed. T.S. Denber, pp. 235-239, American Institute of Physics, New York, 1986
79. Michael I. Jordan, Serial Order: A Parallel Distributed Processing Approach, ICS Report 8604
80. Pentti Kanerva, Sparse, Distributed Memory for Patterns and Sequences, RIACS, NASA AMES
81. Pentti Kanerva, Parallel Structures in Human and Computer Memory, RIACS Technical Report TR-86.2, January 1986
82. Alan H. Kawamoto and James A. Anderson, A Neural Network Model of Multistable Perception, Acta Psychologica 59 (1985)35-65 North Holland
83. James D. Keeler, Comparison Between Sparsely Distributed Memory and Hopfield-Type Neural Network Models, submitted to J. Cog Sci also RIAES Tech. Report 86.31
84. James D. Keeler, Information Capacity of Hebbian Neural Networks, submitted to Phys. Rev. Letters PACS Numbers: 87.30, 89.70
85. A. Harry Klopf, Drive-Reinforcement Learning: A Real-Time Learning Mechanism for Unsupervised Learning, Submitted to the IEEE First Annual International Conference on Neural Networks, San Diego, California 21-24 June 1987
86. Christof Koch, Jose Marroquin, and Alan Yuille, Analog "Neuronal" Networks in Early Vision, Proc. Natl. Acad. Sci. Vol. 83, pp. 4263-4267, June 1986
87. Teuvo Kohonen, Adaptive, Associative, and Self-Organizing Functions in Neural Computing, #135, Helsinki University of Technology, Dept. of Technical Physics Rakentajanaukio 2 C, SF-02150 Espoo, Finland

88. Teuvo Kohonen, Representation of Sensory Information in Self-Organizing Feature Maps, and Relation of These Maps to Distributed Memory Networks, #129, Helsinki University of Technology, Dept. of Technical Physics Rakentajanaukio 2 C, SF-02150 Espoo, Finland
89. Bart Kosko, Adaptive Bidirectional Associative Memories, To appear in "Applied Optics", Nov. 1987
90. Bart Kosko, Competitive Adaptive Bidirectional Associative Memories, ICNN, 1987
91. S. Y. Kung, and H. K. Liu, An Optical Inner-product Array Processor For Associative Retrieval, A reprint from O-E LASE '86
92. Alan Lapedes and Robert Farber, A Self-Optimizing, Nonsymmetrical Neural Net for Content Addressable Memory and Pattern Recognition, Physica 22D (1986) 247-259
93. Alan Lapedes and Robert Farber, Programming a Massively Parallel, Computation Universal System: Static Behavior, Los Alamos National Laboratory
94. Lerner, Computer Vision Research Looks to the Brain, High Technology, May 1980.
95. Daniel S. Levine, A Neural Network Model of Temporal Order Effects in Classical Conditioning Modelling of Biomedical Systems IMACS, 1986, Modeling of Biomedical Systems, Elsevier Science Pub., 1986
96. Daniel S. Levine, A Neural Network Theory of Frontal Lobe Function, Proc. of the Eighth Annual Conf. of the Cognitive Science Soc. (Amherst, MA, 1986), Erlbaum
97. Daniel S. Levine, Neural Population Modleing and Psychology: A Review, Mathematical Biosciences 6:1983.
98. Ralph Linsker, From Basic Network Principles to Neural Architecture: Emergence of Spatial-Opponent Cells, Proc. Natl. Acad. Sci. USA Vol. 83, pp. 7508-7512, October 1986
99. R. Lippmann, An Introduciton to Computing with Neural Nets, IEEE, ASSP Magazine, April 198.
100. Jeffrey Locke, David Zipser, Learning the Hidden Structure of Speech, Report Number ICS 8701, UCSD
101. James L. McClelland, Jeffrey L. Elman, The TRACE Model of Speech Perception, ICS, UCSD
102. Robert J. McEliece, Edward C. Posner, Eugene R. Rodemich and Santosh S. Venkatesh, The Capacity of the Hopfield Associative Memory, IEEE Trans. Inform Theory, 1987 (To Appear)
103. Michael C. Mozer, RAMBOT: A Connectionist Expert System That Learns by Example, ICS, UCSD
104. C. Mozer, Early Parallel Processing in Reading: A Connectionist Approach, ICS, UCSD P. Mueller, J. Lazzaro, A Machine for Neural Computation of Acoustical Patterns With Application to Real Time Speech Recognition, U. of Penn., Phila., PA 19104-6059, Calif. Inst. of Technology, Pasadena, CA
105. Michael H. Myers, Some Speculations on Artificial Neural System Technology, NAECON 1986 Proc.
106. Stephen M. Omohundro, Efficient Algorithms with Neural Network Behavior, Dept. of Comp. Sci, U. of Illinois at Urbana-Champaign
107. J. Michael Oyster, Walter Broadwell, Fernando Vicuna, Associative Network Applications to Robot Vision, Report No. 320-2777, Los Angeles Scientific Center Ian Parberry, Georg Schnitger, Relating Boltzmann Machines to Conventional Models of Computation, Dept. of Computer Science, Pennsylvania Univ.
108. David B. Parker, A Comparison of Algorithms for Neuron-Like Cells, 1986 AIP Proc.: Neural Networks for Computing
109. Barak A. Pearlmutter, Geoffrey E. Hinton, G-Maximization: an Unsupervised Learning Procedure for Discovering Regularities, Proceedings of the conference on "Neural Networks for Computing" American Institute of Physics, 1986
110. Andras J. Pellionisz, Sensorimotor Operations: A Ground for the Co-Evolution of Brain Theory with Neurobotics and Neurocomputers, Proc. IEEE 1st Ann. Internatl. Conf. on Neural Networks, San Diego, CA 1987 June
111. P. Andrew Penz, The Closeness Code: An Integer to Binary Vector Transformation Suitable for Neural Network Algorithms, Texas Instruments Inc., Dallas, TX Tomaso Poggio, Vincent Torre and Christof Koch Computational vision and Regularization Theory Source: Reprinted from Nature, Vol. 317, No. 6035, pp. 314-319, 26 September 1985
112. D. Psaltis, J. Hong, and S. Venkatesh, Shift Invariance in Optical Associative Memories, Presented at Conf. on "Optical Computing', LA, 1986
113. Demetri Psaltis and Cheol Hoon Park, Nonlinear Discriminant Functions and Associative Memories, Calif. Inst. of Tech., Dept. of Electrical Eng., Pasadena, CA 91125
114. George N. Reeke, Jr. and Gerald M. Edelman, Selective Neural Networks and Their Implications for Recognition Automata, The Rockerfeller University
115. Kathleen J. Roney and Gordon L. Shaw, Analytic Study of Assemblies of Neurons in Memory Science, Physics Dept., Univ. of Calif., Irvine, CA 92717 Mathematical Biosciences 51:25-41 (1980)

116. David E. Rumelhart, Geoffrey E. Hinton and Ronald J. Williams, Learning Representations by Back-Propagating Errors, Nature Vol. 323 9 October 1986 David E. Rumelhart, Donald A. Norman, Representation in Memory, Center for Human Information Processing, U. of Calif., San Diego, La Jolla, CA
117. David E. Rumelhart, et al., Parallel Distributed Processing, (Vol 1 & 2) (MIT Press 1986).
118. J.P. Sage, K. Thompson, and R.S. Withers, An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principles, MIT Lincoln Lab.
119. Tariq Samad, Refining and Redefining the Back-Propagation Learning Rule for Connectionist Networks, Submitted to IEEE Systems, Man & Cybernetics Conf.
120. Tariq Samad, A Connectionist Network That Learns to Process Some (Very) Simple Sentences, Honeywell, Inc., 1000 Boone Ave.N, Golden Valley, MN 55427
121. Richard J. Sasiela, Forgetting As A Way To Improve Neural-Net Behavior, American Inst. of Physics, Neural Networks for Computing 1986
122. Lokendra Shastri, Evidential Reasoning in Semantic Networks: A Formal Theory and its Parallel Implementation, TR 166, University of Rochester
123. Gordon L. Shaw and Kathleen J. Roney, Analytic Solution of a Neural Network Theory Based on an Ising Spin System Analogy, Physics Letters, Vol. 74A, number 1,2 29 Oct 1979
124. Gordon L. Shaw, Physics Dept., Univ. of California, Irvine, CA 92717, Space-Time Correlation of Neuronal Firing Related to Memory Storage Capacity, Brain Research Bulletin, Vol 3, pp. 107-113, 1978
125. Gordon L. Shaw, Dennis J. Silverman, and John C. Pearson, Model of Cortical Organization Embodying a Basis for a Theory of Information Processing and Memory Recall, (Hebb synapse/selective adaptive network/axial next-nearest-neighbor Ising model/fluctuations/synchronous time steps) Proc. Natl. Acad. Sci. USA Vol. 82, pp. 2364-2368, April 1985
126. R. M. Siegel and R. A. Andersen, Representation of Head-Centered Visual Space in the Inferior Parietal Lobule of Macaque Monkey, The Salk Inst., San Diego
127. Christine A. Skarda and Walter J. Freeman, Brains Make Chaos to Make Sense of the World, Behavioral and Brain Sciences, June 1987
128. S.L. Small, L. Shastri, M.L. Brucks, S.G. Kaufman, G. W. Cottrell, and S. Addanki, ISCON: A Network Construction Aid and Simulator for Connectionist Models, TR 109, U. of Rochester
129. B.H. Soffer, G.J. Dunning, Y. Owechko, and E. Marom, Associative Holographic Memory with Feedback Using Phase-Conjugate Mirrors, Reprinted from Optics Letters, Vol. 11, page 118, February 1986
130. H. Sompolinsky, Neural Networks with Nonlinear Synapses and a Static Noise, Physical Review, Vol 34, No. 3, Sept. 1986
131. Harold H. Szu, Non-Convex Optimization, SPIE Vol. 698 Real Time Signal Processing IX (1986)
131. Harold H. Szu and Richard A. Messner, Adaptive Invariant Novelty Filters, Proc. of the IEEE, Vol. 74, No. 3, March 1986
132. D. W. Tank and J. J. Hopfield, Neural Computation by Concentrating Information In Time, ATT Bell Labs, 1987
133. J. Ticknor, Harrison H. Barrett, Optical Implementations in Boltzmann Machines, Optical Engineering 26(1), 016-021 (Jan. 1987)
134. Max Stanford Tomlinson Jr., Thesis Proposal: Analog Back Propagation Systems, UCSD, Institute for Cognitive Sciences 1987
135. Santosh S. Venkatesh and Demetri Psaltis, Linear and Logarithmic Capacities in Associative Neural Networks, Preprint submitted Mar 1985 to IEEE Transactions on Information Theory, Revised Nov. 1986
136. John Voevodsky, A Neural-Based Knowledge Processor, Neuraltech, Mountain View, CA
137. [von der Malsburg, C.], Disordered Systems and Biological Organization, Springer (1986)
138. Paul J. Werbos, Building and Understanding Adaptive Systems: A Statistical/Numerical Approach to Factory Automation and Brain Research, IEEE Transactions on Systems, Man, & Cybernetics, Vol. SMC-17, No. 1, January/February 1987
139. Ronald J. Williams, Reinforcement Learning in Connectionist Networks: A Mathematical Analysis, ICS, UCSD
140. Ronald J Williams, Inference of Spatial Relations, Institute for Cognitive Science UCSD C-015 La Jolla, CA 92093
141. Y.T. Zhou, R. Chellappa and B.K. Jenkins, A Novel Approach to Image Restoration, Based on a Neural Network To be presented at First Intl. Conf. on Neural Nets San Diego, June 1987
142. David Zipser, Programming Neural Nets To Do Spatial Computations, ICS Report 8608, UCSD

DESIGN OF A NEURAL NETWORK SIMULATOR ON A TRANSPUTER ARRAY

Gary McIntire
Advanced Systems Engineering Dept. Ford Aerospace, Houston, TX.

James Villarreal, Paul Baffes, and **Monica Rua** Artificial Intelligence Section, NASA/Johnson Space Center, Houston, TX.

Abstract

A high-performance simulator is being built to support research with neural networks. All of our previous simulators have been special purpose and would only work with one or two types of neural networks. The primary design goal of this simulator is versatility; it should be able to simulate all known types of neural networks. Secondary goals, in order of importance, are high speed, large capacity, and ease of use. A brief summary of neural networks is presented herein which concentrates on the design constraints imposed. Major design issues are discussed together with analysis methods and the chosen solutions.

Although the system will be capable of running on most transputer architectures, it currently is being implemented on a 40-transputer system connected in a toroidal architecture. Predictions show a performance level nearly equivalent to that of a highly optimized simulator running on the SX-2 supercomputer.

Introduction

There are several ways to simulate large neural networks. Computationally speaking, some of the fastest are via optical computers and neural net integrated circuits (hardwired VLSI). However, both methods have some basic problems that make them unsuitable for our research in neural networks. Optical computers and hardwired VLSI are still under development, and it will be a few years before they will be commercially available. Even if they were available today, they would be generally unsuitable for our work because they are very difficult (usually impossible) to reconfigure programmably. A non-hardwired VLSI neural network chip does not exist today but probably will exist within a year or two. If done correctly, this would be ideal for our simulations. But the state of the art in reconfigurable simulators are supercomputers and parallel processors. We have a very fast simulator running on the SX-2 supercomputer (200 times faster than our VAX 11/780 simulator), but supercomputer CPU time is very costly.

For the purposes of our research, several parallel processors were investigated including the Connection Machine, the BBN Butterfly, and the Ncube and Intel Hypercubes. The best performance for our needs was exhibited by the INMOS Transputer System.

Our previous neural network simulators have been specific to one particular type of algorithm and consequently would not work for other types of networks. With this simulator our primary goal is to be able to implement all types of networks. This will be more complicated but is deemed well worth the effort. When we are finished, it should be possible to implement a different kind of network in less than a day. The performance reduction will be less than 10 percent for this general-purpose capability.

Example networks

To have examples to work with, let us consider two very typical neural networks. The first is a three-layer feedforward network (fig. 1) that is to be trained with the generalized delta learning algorithm (also called back propagation). Rumelhart et. al.[5] describe this algorithm in detail. We will assume that every node in one layer is connected to every node in its adjacent layer. The network will be trained with a number of I/O pairs which are an encoding of the associations to be learned. The sequence of events to the algorithm can be described as follows. First, an input vector is placed into the input nodes. The weight values of the connections between the input layer and hidden layer are then multiplied by the output value of the corresponding input nodes and the result is propagated forward to the hidden layer. These products are collected and summed at each node of the hidden layer. Once all the nodes in the hidden layer have output values, this process is repeated to propagate signals from the hidden layer to the output layer. When the output layer values are computed, an error can be calculated by comparing the output vector with the desired output value of the I/O pair.

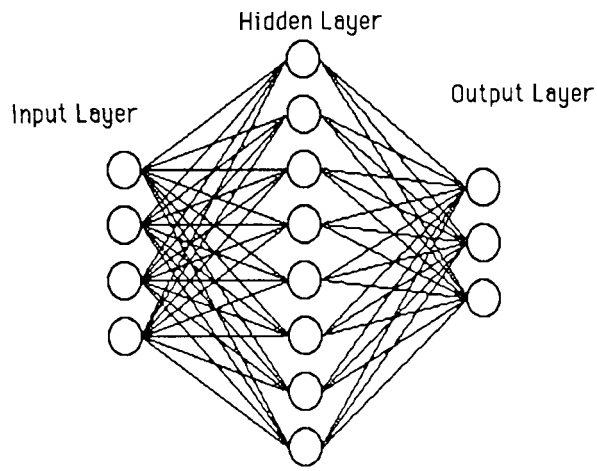


Figure 1. - Feedforward network.

With this error computed, the weights at each connection between the hidden and the output layer can be adjusted. Next, the error values are backpropagated from the output layer to the hidden layer. Finally, each weight can be adjusted for the connections between the input and hidden layers.

This entire sequence is repeated for each successive I/O pair. Note that this algorithm has a sequence of events to it. Other neural net algorithms do not; instead, every node and connection updates itself continuously[2,4]. Such algorithms can be viewed as a sequence of length one.

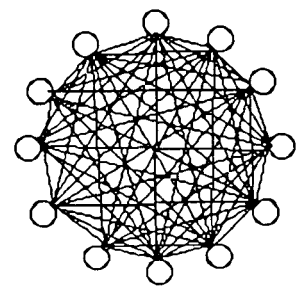


Figure 2. - Hopfield net.

The second network to be considered is the Hopfield network[1] shown in figure 2. It is an auto-associative memory in which every node is equivalent; i. e., they are all used as inputs and outputs. Every node is connected to every other node but is not connected to itself. The connections are bidirectional and symmetric which means that the weight on the connection from node i to node j is the same as the weight from node j to node i.

Other neural networks impose different constraints. For example, the connection scheme may be totally random, or it may be that every ninth node should be connected. The equations used are very often different. The order of sequencing forward propagation, back propagation, weight adjusting, loading inputs, etc., also may be different. Stochastic networks update their values probabilistically. Yet some generalizations can be made for almost all types of networks and these are what we have used as the basis of our simulator design.

The first generalization is that all network computations are local computations. In other words, the computations only involve a node and the nodes to which it is connected.

This is a generally accepted principal in the literature and some authors even use it in the definition of a neural network.

The second generalization is that all neural computations can be performed by applying functions to the state variables of a node and its neighbors. While this is a trivial restatement of Turing equivalence, the emphasis here is that this can be done in a computationally efficient manner.

The third generalization is that any computation performed by a node on its input signals can be decomposed into parallel computations that reduce multiple incoming signals to a single signal. This single signal is then sent to a node to be combined with single signals sent from the other parallel computations. This allows a node computation to be broken apart, and the partial results forwarded to the node which computes the final result. For example, a computation that is common to almost all networks is the dot product which is a sum of products. This can be decomposed into multiple sums of products whose results are forwarded and summed.

Memory Utilization

For maximum memory efficiency, the target design should have one memory cell of the minimum possible size for each state variable which the network must keep. These state variables consist of the values kept for the connections and nodes. For a connection in the two networks above, one memory cell per connection is needed since the only information associated with a connection is its weight. If the connectivity is like that of the above two networks (where everything in one layer in connected to everything in another layer), the connection information can be implicit. At the other extreme, where the connectivity is totally random, a additional pointer between nodes would have to be kept for each connection.

For nodes, a network like the Hopfield network only has to keep one variable: its output value. The size of this output value, however, can vary. Some networks work with 8-bit or 16-bit integers while others use floating point numbers. Other kinds of networks require additional state variables at the nodes and connections. Almost all networks include added representations to ease debugging tasks. Because of these differences in size, we have allowed the user to specify node variables and their types by defining structures in the C language to hold the state variable information. This allows all the flexibility of C (ie. integers, floats, doubles, bytes, bit fields, etc.).

CPU Utilization

Since previous simulations have shown that a neural net simulator spends almost all of its time processing connections (typically, there are many more connections than there are nodes), an examination of execution speed must focus on the calculations done for each connection. The operation common to almost all neural nets is some function of the dot product. This is

$$O_i = \mathbf{f}(\Sigma W_{ij} O_j)$$

where O_i is the output of the i^{th} node, Oj is the output of the j^{th} node, W_{ij} is the connecting weight, and **f** is some function applied to the dot product (note that the time spent executing the function **f** would be relatively small in any sizable network since there would be few nodes compared to the number of connections). Notice that the above computation can be thought of as a loop of multiply-accumulate operations. For each operation the computer must calculate two addresses, fetch the two referenced variables, multiply them together, and add that product to a local register. If the addresses were sequential, calculating a new address could be done by incrementing a register. Otherwise, a randomly connected network would require that the computer fetch a pointer which would be used as the address of O_j.

Without the extra pointer, we could imagine that the weight, W_{ij}, and O_j could be fetched in parallel from two separate memories and pushed into a pipeline where they would be multiplied and summed. Using today's electronic components, memory fetching would be the bottleneck with memory

access times of 100 nanoseconds (ns) per fetch. Thus the execution rate of the fetch-multiply-store loop above would be on the order of 100 ns. If the weight and the output could not be fetched in parallel, the loop would take 200 ns. As a result the best can be hoped for, even with a custom made VLSI chip, is about 200 ns per cycle through the loop. Of course, this can be done in parallel with multiple chips. Said another way, this is 5 million connections per second per memory bank. Since economic and time constraints precluded the design of a VLSI chip, the best commercially available hardware was sought.

Transputer Architecture

A transputer[3] is a 32-bit, 10 million-instruction-per-second (MIPS), single-chip microcomputer manufactured by INMOS, Great Britain's leading semiconductor manufacturer. Transputers are designed to be components in large parallel processing systems and have hardware multitasking for sub-microsecond task switching times. Each transputer has four 10-Megabit per second, full duplex serial links. We purchased the INMOS transputer system ITEM 4000, a 40 transputer parallel processor with capabilities of 400 MIPS, 50 MFLOPS, and 10 Mbytes total local memory (256K per processor). An additional transputer plugs into an IBM personal computer (PC) advanced terminal (AT) with 2 Mbytes of memory. This transputer uses the PC AT as its I/O subsystem. Yet another transputer controls a 512x512x8 graphics board. The development system has both the C language and OCCAM, the parallel processing language of the transputer.

Decomposition of the Matrix

There are several ways to divide the nodes and connections of the network among the processors. One scheme is to copy the node variables to all processors. This makes allocation simpler, but it can use a lot of memory. We have chosen to decompose the connection matrix (fig. 3) into partitions that require only a subset of node variables.

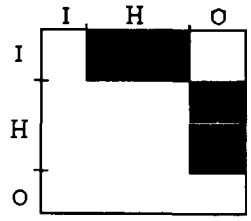


Figure 3. - Connection matrix.

The connection matrix is formed by having one row and one column for each node in the network. Assume that each row of the matrix represents a "from" node (the source of a connecting link) and each column a "to" node. For every connection, a mark is placed at the intersection of its from and to nodes. Figure 3 shows this for a three-layer feedforward network where all of the input nodes are connected to all of the hidden nodes and all of the hidden nodes are connected to all of the output nodes. The nodes and connections are then allocated to processors by sectioning the covered areas of this matrix into the same number of regions as processors. A processor associated with a region must have access to both the from nodes represented by the rows of its region and to the to nodes represented by the columns. When a node is unidirectionally transmitting a signal from one node to another, the state variables of the from node are not the same as the state variables of the to node (in a typical case, the from node must have a variable for its output value but the to node must have an accumulator for its dot product). Therefore, the state variables are separated into "exported values" and "partial results," and only the necessary variables are kept in a processor. Thus the memory allocation for copies of node variables required by a processor when receiving a region is equal to the height of the region (in number of nodes) times the number of bytes for the from node "exported" state variables plus the width of the region times the number of bytes for the to node "partial result" state variables. Let us assume that the number of bytes for from nodes is F, the number for to nodes is T, the number of nodes in the network is N and the number of processors we have is P. Let us next assume the matrix is fully covered, and that we have some number of

processors that is a perfect square. If we leave enough space for every node in a processor the number of bytes allocated is $(NF + NT)P$. However, by partitioning the matrix, the number of bytes allocated is $(NF + NT)P/\sqrt{P}$ (again, assuming square regions). This is a factor of $\sqrt{P}$ savings. When the allocations are contiguous groups of nodes, node variable structures can be stored in an array with minimal overhead both in access time and memory. Other schemes would necessitate pointer or hash table overhead.

Load Balancing

A basic problem with most parallel processing schemes is balancing the load evenly so that all processors can be working most of the time. If one processor is slower than the rest, all of the other processors have to wait for it to finish before they can all synchronize and continue. Since the time of execution is proportional to the number of connections that a processor must process, our matrix decomposition scheme solves this if equal areas can be assigned to the processors. If the matrix is fully covered, this is easy. But when it is covered with many irregularly sized areas, it is more difficult. We are developing a heuristic method for doing this which we refer to as our load balancing algorithm. Because processes may be sequenced, as in the generalized delta algorithm, it is necessary to have a separate matrix for each asynchronous phase group of connections. An asynchronous phase group is defined as a set of connections where all processing can be done in parallel. In the generalized delta rule described above, there are two phase groups: input to hidden and hidden to output. The user specifies the phase when he defines the group.

Mapping Macros

To get a system up and running in a reasonable amount of time, the user is required to modify a few sections of program and recompile the source to create a network with his specifications. This code modification method offers total flexibility. The user specifies the network by calling functions. He also must specify the structures to hold the state variables of nodes and connections. To specify the equations to be applied, a "map-connections" macro is provided which expands to the actual code that goes in each of the slave processors. This macro handles all of the addressing and hands the user pointers to the connection variables and to its two adjacent node variables. The code that he provides can do whatever he wants to the state variables. It can propagate a signal forward, backward, or both ways. It could initialize the weights. It could save the weights to a file or recover them. A similar "map-nodes" macro is also provided. The macros create functions that are called with an argument of the node group or connection group to which the user wants the function applied. This macro approach expands to code that is 90 percent as run-time-efficient as can be handcoded. To execute the sequencing of the generalized delta rule the user would call a predefined function that loads the inputs. The user would then broadcast a message to all processors that would invoke his macro-defined function with an argument of the input to hidden connection group. This function executes in parallel in all of the slave processors. It first checks to be sure it has connections from the input to hidden connection group; if not it just responds "done" to the master. The user then broadcasts to invoke his function that was defined with "map nodes" to process the hidden nodes. The same is done from the hidden nodes to output nodes. Backpropagating is very similar but the user invokes different functions. He probably would name this routine "train_one_input_output_pair" and call it inside a loop to do all of his training. Likewise, the user might define a function called "output_of" that takes an input vector as argument, propagates it through the network, and returns an output vector. If the user used symbolic constants in his functions, he would only have to change constants such as Number_of_input_nodes, Number_of_hidden_nodes and Number_of_output_nodes to change their sizes. Even architectural changes can be made with small changes in the program (such as connecting all input nodes to all output nodes as well). Although this approach allows the total flexibility that many users want, others

will dislike tinkering with the code. Ultimately, a much friendlier user interface will be provided as well. We are considering both a language and menu-driven graphics for specification of the network.

Program Structure

The system architecture we are using is master-slave. The master transputer in the IBM PC acts as an interpreter of the commands from the user interface. In turn, the master issues commands to the slaves whose sole task is to interpret commands from the master. The slaves merely look the command up in a table (index an array) and execute the function associated with that command (whose pointer is stored in the table). The argument to the function is a pointer to the buffer that holds the remainder of the command message which contains the arguments to the function.

Synchronization

Synchronization of the master and slave processes is accomplished by having the slaves respond to every command. When the master gets as many responses as there are processors, he can continue.

Communication

Since processors are only connected in a point to point fashion, a message between two non-adjacent processors must be relayed by intermediate processors. Our communications process inside each processor continually waits for a message to come in from any of the four input ports and, when one comes in, it reads the address in the message and looks in a table (indexes an array) to determine the appropriate output channel for retransmitting the message (a channel is a logical port; it may be a physical port or a location in memory[3]). Buffering is used in each processor to avoid deadlock and to smooth irregularities in transmission rates.

Even though it reduces the total amount of communication required, the method of decomposing the matrix means that some nodes are split across processors and that the partial results accumulated at several of the processors must be shipped to a central location (for this node) to be combined. This location is called the home processor of the node. Each processor is home for a roughly equal number of nodes that are in the same asynchronous processing phase. When a map_nodes function is called, it is applied in the home processor of the node with the variables which it contains. These are referred to as the static variables of the node. Since they do not have multiple copies, the static variables use little memory.

So the bulk of the communications results from processors sending partial results to the home nodes and from home nodes exporting these processed values to several processors. Analyzing the time this will take is very difficult, but there are two potentially limiting factors that we can analyze individually: the limit set by the serial links and the limit set by the CPU cycles required to buffer and relay messages. Surprisingly, the CPU time is the limiting factor. The analysis is as follows.

The number of processors that ship a message home and that the home node ships to can be seen by examining the sections of the connection matrix after the areas have been carved out. Let us assume that a fully covered matrix has been evenly partitioned into 5 rows by 8 columns and that each of the 40 processors gets 1 partition. From a home node processor's point of view, five processors must ship their partial results to the home node processor and this processor must combine these results and ship this new output value to eight processors (let us assume the home processor is not one of these). This means that there are 14 messages shipped for every node. It can be determined by enumeration that the average distance between processors in our array (fig. 4) is 3.5 links.

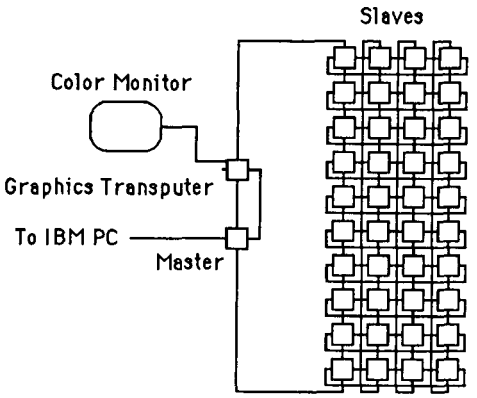


Figure 4. - Processor configuration

Each of these 14 messages must travel over 3.5 links, yielding 3.5 • (5 + 8) ≈ 46 transmissions per node. If each message is 12 bytes (96 bits) long, this is 96 * 46 = 4416 bits transmitted per node. Since each serial link is 10 million bits per second and there are 80 full duplex serial links in our network, there are 2 • 80 * 10million = 1600 million transmitted bits per second possible (This assumes that every bit slot is being used, which is not likely; but if adequate buffering is available and the CPU retransmit speed is sufficiently fast, this is almost the case (> 50 percent of this)). Dividing this by 4416 bits gives 362,318 nodes that could transmit each second. This means that the serial-link is limited to about 362,000 nodes per second. This could become the limiting factor in networks with large numbers of nodes and few connections per node; but, typically, there will only be a few thousand nodes in a network so other things will probably limit throughput before the serial links do.

To examine the CPU speed requirement, remember that there are 46 transmissions per node. For each of these, a processor must receive the message, buffer it, determine the port to which it should be sent, and then retransmit it. The transputer direct memory access (DMA) hardware handles the receiving and transmitting concurrently. All the CPU must do is copy the received message to a buffer, determine where to send it, and later copy it from the buffer. There are 46 transmissions per node, and a serial-link-limited rate of 362,000 nodes per second is 46 • 362,000 = 16,652,000 transmissions per second. When this is distributed over the 40 processors, each one must do 416,300 transmissions or one transmission every 2.4 microseconds (μs). Yet summing instruction execution times shows that the transputer will actually take about 10 μs to retransmit for a rate of 81,600 nodes per second. This is about four times slower than the serial link rate. So the speed of processing networks on this system is still CPU limited.

It takes about 2 μs to fetch operands and do a multiply accumulate on our 20 MHZ transputer. This is what must be done at each connection when propagating forward. Forty transputers can process 20 million connections per second. The time to propagate an input forward to get an output from a feedforward network can be estimated by (number_of_connections / 20 million) + (number_of_nodes / 81,600) since these two terms represent 90 percent of the processing in typical networks.

The 2 μs required for each fetch-multiply-accumulate loop is about 10 times slower than what would be possible with a handcrafted VLSI chip but is significantly faster than most other 32-bit microprocessors. This, along with the capability of packing several transputers in a very small space, makes a transputer array a very cost effective solution to neural network simulation with off the shelf components.

Conclusion

We have discussed the constraints imposed by neural networks on simulation. We have shown what is achievable in terms of memory efficiency and simulation speeds and have compared our design to this. We have discussed a technique for partitioning a neural network to minimize memory waste on a parallel machine. The program structure also was discussed. The communication network was analyzed to determine what the costs of communication are. The resulting design gives us a neural network simulator that has a performance level nearly equivalent to the highly optimized simulator we have running on the SX-2 supercomputer for a cost equivalent to 2 days of CPU time on that supercomputer.

References

1. Hopfield, J., "Neural Networks and Physical Systems with Emergent Collective Computational Capabilities", Proceedings of the National Academy of Science, USA, Vol 79, April, 1982, pp. 2554-2558.

2. Hopfield, J., "Neurons with Graded Response Have Collective Computational Properties Like Those of Two-State Neurons", Proceedings of the National Academy of Science, USA, Vol 81, May, 1984, pp. 3088-3092.

3. INMOS Transputer Reference Manual INMOS Ltd. 72-TRN-006-03, Bristol, UK 1987.

4. Grossberg, S. STUDIES OF MIND AND BRAIN: NEURAL PRINCIPALS OF LEARNING, PERCEPTION, DEVELOPMENT, COGNITION AND MOTOR CONTROL, Reidel Press, Boston, MA.

5. McClelland, J., and Rumelhart, D., PARALLEL DISTRIBUTED PROCESSING: EXPLORATIONS IN THE MICROSTRUCTURE OF COGNITION, MIT Press, Cambridge, MA, 1986.

Interchange of Electronic Design Through VHDL and EIS

Richard M. Wallace
Computer Scientist
Air Force
Wright-Aeronautical Laboratories
Electronic Technology Division
Wright-Patterson, AFB, Ohio

Abstract

The need for both robust and unambiguous electronic designs is a direct requirement of the astonishing growth in design and manufacturing capability during recent years [1,2]. In order to manage the plethora of designs, and have the design data both interchangeable and interoperable, the Very High Speed Integrated Circuits (VHSIC) program is developing two major standards for the electronic design community. The VHSIC Hardware Description Language (VHDL) is designed to be the lingua franca for transmission of design data between designers and their environments. The Engineering Information System (EIS) is designed to ease the integration of data between diverse design automation systems. This paper describes the rational for the necessity of these two standards and how they provide a synergistic expressive capability across the macrocosm of design environments.

The Rational

The VHSIC Program has propelled forward the design density of electronic systems to a point where current computer aided design tools, design representations, and the corresponding data management systems begin to limit the designers' ability to design throughout the continuum of system levels to physical levels. In order to provide mechanisms for designers in the next decade, the VHSIC program has several design automation efforts under-way in its Technical Insertion and System Level Design tool subprograms. The dual focus of these subprograms is use of VHDL as the notation for design/description and the EIS for integration of design data. Initiation of the the VHDL program was motivated by the diversity of design notations that failed to encompass the broad range of descriptive capability required for advanced system documentation, and by the need of the DoD to provide a standard descriptive notation for systems that have life-spans upward of fifteen years. The VHSIC Hardware Description Language provides an economical method to decrease the system design time and cost of government re-procured ICs. Design costs for ICs are now in the range of $2 to $5 million and development costs must be reduced to meet future needs. Maintaining and upgrading electronic systems in inventory demands specific, current, and rigorous descriptions. As English can be vague, and fraught with idiomatic contextual references, the automation of design and design verification demands a technology independent, rigorous notation as is VHDL.

The EIS is envisioned to provide efficiency for the design process and to ease the insertion of VHSIC technology into electronic defense systems. To this end the development of an integrated design, documentation, and life-cycle maintenance system for complex electronic systems must support initial specification design data capture to fabrication and testing data in one continuum. An EIS system is not, and would not, be available from the commercial sector due to the high cost of development for a "turn-key" system being beyond most businesses. Therefore design automation tool users are forced to integrate an assortment of design tools from other vendors and those that are developed internally. The unique, proprietary, and internal design representations of each vendors' design automation tool complicates the integration task drastically. Integration has been a sever problem [3,4] while integration is known to be beneficial [5,6]; thus the EIS has as its main goal the reduction of the present difficulties involved with intgration of different vendors' design tools by developing a set of inter-oper-

ability standards and then demonstrating them. The VHDL in total is to be used in the EIS system as the design documentation formalism required of complex electronic systems.

VHDL -- The Lingua Franca

A standard hardware description language benefits all industries that depend on electronics. By its use the problem of a second source can be greatly reduced. But how is this accomplished? What makes the VHDL such a beneficial notation for electronic design? From the inception of a standard hardware description language [7] the focus of the language was to allow a hierarchical continuum of design notation from the system to gate level. A discussion of the language hierarchy must begin at its basic building block, the design entity; and then progress through its other features to show the capability of the language for electronic design and the the transfer of that design from designer to designer.

The design entity is the principal hardware abstraction in VHDL. A design entity provides the separation of interface and function to allow a hierarchical design decomposition. The crux of the design entity is the interface which allows the entity to be combined with other components. The interface is the abstraction's "pin-out" that describes the data paths and other factors that need to be known by component users. The secondary part to a design entity is its body which describes the organization and/or operation of a component. As an abstraction, the entity interface may possess multiple bodies, each representing a different implementation or emphasizing a different view of design. A design entity models electronics of any intricacy. Examples would be a logic gate, a flip-flop, a control unit, or a computer system. In fact, the range is only limited by the imagination of the designer as design entities can be used to describe any physical object having a bounded identity.

The design entity interface contains information that is common to the bodies that use the entity interface. This information includes data that is visible and is not visible externally. Of the visible data there are two types ports and generics. The non-visible information may define types, constants, and attributes that are used by the alternate bodies of the entity. Inclusive of the object information, the interface can contain assertions that specify operating properties and operational circumstances of the entity. Operating properties specify desired timing or functional relationships demonstrated by the entity. Operating circumstances specify external conditions that must be stated in order for the entity to correctly model its component.

To define communication channels among design entities and the outside world, the port data describes the mode and type of that information. To pass data that is not part of an entity's port interface, but is important to the operating circumstances, the design entity interface may have generics. For example, a generic value would be passed to the entity to specify a particular technology that the design entity is representing. Generics may represent instantiations of preconditions for execution.

Given the interface of a design entity, the designer provides a body that will describe the function of the entity. In VHDL there are two major divisions of entity bodies; the architectural body that expresses the data transformations that occur within the entity and the configuration body that controls the choice of design entities that are used to model sub-components and the distribution of signal definitions.

In an architectural body the description styles that designers use roughly fall into three categories: structural, data-flow, and behavioral. As is implied, structural description is approximately equivalent to the schematic connection of electronic components. The data-flow description method consists of register transfer level data transforms. The behavioral method of description allows the designer to specify transforms in wholly algorithmic terms. Any given architectural body may use these three general forms of description interchangeably.

With the capability of developing a library of similar component designs, it is desirable to make use of existing entities even if names or ports are not exactly what are required, but a subset interface will suffice. Additionally, a design series can have multiple configurations, each using slightly different design entities to implement the given component's behavior. The configuration body, which contains the configuraton specification, provides the ability to respecify the default association rules so that an architectural body's compo-

nents may be bound to corresponding but not identical design entities. The architectural bodies must preceed the configuration bodies which use them. In this way a configuration body can add or modify the enity configuration post-design without altering its basic architecture.

With the basic structural elements of the VHDL identified, the data of such a block structured language must follow in rigor, and scope. The VHDL is a strongly typed language based on the syntax and semantics of Ada. With this being known, the VHDL supports descriptions of objects from the typical bit values of '0' and '1' to higher levels of abstraction such as "integer," "message packet," and "instruction." With the range of data that can be described, VHDL avoids the pit-fall of predefining data types available to the designer. This gives the designer the ability to completely describe new data types as they are needed. The set of types available to the designer include predefined types such as BIT, BOOLEAN, REAL, INTEGER, CHARACTER, and TIME. Additionally all scalar and composite types are allowed. These types would include enumeration types, physical types (allowing expression of measurement defined in a base unit), records, and multidimensional arrays. VHDL has the ability to create functions and procedures and place these in packages to enable the designer to encapsulate algorithmic behavior.

The two most salient features of VHDL for future application to artificial intelligence are the ability to create and attach attributes to objects and have liturgical assertions that have global scope for a design entity. As new technologies emerge for design and construction of electronic circuits, VHDL provides an attribute mechanism that allows designers to associate extra information with descriptions of components or parts of components. As attributes can be referenced in VHDL this allows entities and data-objects to have LISP-like atom properties. This capability is useful in intelligent silicon compilation [8,9,10]. In order to produce designs that are both efficient, and more importantly correct, the VHDL has the assertion ability required in many verification systems [11]. Assertion statements check static or dynamic conditions that are either checked prior to simulation or during simulation as signal values change. Assertions may occur at any point in a VHDL description and are user controllable in order to report the condition of the entity.

EIS -- The Pax Romana

In 1984 the disparity due to the diversity of design formats and languages prompted outcries from industry where the future was seen as,

> "A nightmare of incompatible formats and a babel of different languages."[12]

The rhetorical question would be, "Has it gotten any better since 1984?" From the surveys of design systems available in the trade press, the answer is no; although efforts by IC designers and fabricators have produced draft interchange formats (e.g. EDIF). In order to couple the large amount of distributed database designs many individual translators have been written. Such one-on-one translation does not provide the integration necessary for automated design and fabrication. Without data integration, no amount of automation will overcome the data interchange problem.

A series of workshops were held to form a base-line for what would constitute the requirements for an EIS. More than 150 people representing near as many organizations attended the workshops. The result was the creation of the DoD Requirements for Engineering Information Systems [13]. Five key areas for an EIS were identified by the participants. An EIS must support:

- the reuse of design information from all forms of input,
- an information repository and data caputre designed for a multi-base, heterogeneous environment,
- an interface to its information model such that it economically supports integration of existing CAE software,
- a system that is not monolithic in use so that installations may tailor the system for current and future needs, and
- the efficiency to support the above functionality in its operation.

The architecture of the EIS is rooted in its information model; which when used, provides a Pax Romana (enforced peace) on the conflict of data representation and data usage. This Information Model is the focus of the EIS effort that will allow the identified key area to be achieved. The requirements are that,

> "The EIS must provide a model of the classes of engineering information that are needed to accurately describe the sem-

> antics of the information in the *engineering environment* in which the EIS operates. The EIS Engineering Information Model (EIM) need not be used to actually represent engineering data; this is the purpose of the common exchange format. Rather, it must provide a definition of all information classes and modeling rules needed as the basis for formulating a conceptual framework for information exhange."[14]

It is this semantic description of engineering information that provides the knowledge-based technology that distinguishes the EIS effort from other data-dictionary based, multi-view databases. It is the goal of the EIM to have the specification of semantics in a precise and understandable form. The information classes and prescribed modeling rules will ensure that the allowable combinations of the data can be modeled in exactly one way; the are no redundant EIM models of the same data within the system.

A goal of the EIS is to develop an accepted *Common Exchange Format* (CEF) in order to promote the exchange of data between design systems, repositories and organizations. From the experience gained in the development of the VHDL, the important factor in data exchange is the information model. Once the model is developed, the development of the exchange format is one of representation notation design. The Object-Oriented Data Language will be used in the EIS for defining the syntax for manipulating objects maintained within an EIS. The PROBE Data Model, an object-oriented extension of DAPLEX, developed by Computer Corporation of America. DAPLEX is a semantic data model and query language that will provide the necessary features for an object-oriented information model; such as

- the concept of an entity or object that has existence independent of its properties or relationships,
- support for relationships between objects and for set-valued properties, and
- types and generalization hierarchies with inheritance.

For access to repositories through the EIS the Object-Oriented Data Language will be used as the CEF between EIS installations. In addition, data exchange adapters will be used to transport design data via VHDL and EDIF. Alternate exchange formats, such as a substantial portion of SQL will be used for non-EIS installations as the program develops thus allowing an interface to foreign information models.

The Object Manager of the EIS is the responsible "agent" for managing objects and functions. It registers new objects, deletes objects that are unneeded, locates and retrieves objects, and provides access to objects. The Object Manager provides services for resolving object references in bindings with application to 1) persistent and temporary objects (data and events), 2) stored and derived objects (database and computed), and 3) passive and active objects (data and processes). Implementation of the Object Manager is based on the design and facilities of the ENCORE system by Brown University.

With the EIS Information Model key-stone set within the EIS, the representation of data is best controlled through rule-processing and control-point activation of data management functions. The *short-term requirements for* rule-based processing of EIM data are that,

> "Rule processing must be supported by programs that implement all required management and control and other rule-based capabilities. There must be an interface specification for every situation in which rule processing is necessary that allows programs to invoke appropriate rule processing programs and pass parameters to them. Rule processing may be implemented via object programs in the short term...." "The EIS must be able to invoke the rule processing services in a heterogeneous, distributed environment. The services must fulfill tool availability requirements..."[15]

In the extended short-term requirements, the general rule-based system which allowed object programs to have static knowledge-bases is modified so that,

> "All rule-based capabilities required by the EIS must be provided by a rule processor, which can be invoked through programs that use the specified" [note: CAIS] "standard interfaces. The rule processor must support the execution of rules specified by a rule spec-

ification language. The EIS must support facilities for adding, deleting, and modifying rules. The rule specification language must support the concept of system supplied variables..." "and must support evaluation of expressions, condition testing and the triggering of actions. The rule specification language must allow for the specification of actions, including sending messages, changing global and object-related management and control information, and invoking programs. The rule specification language must support the concept of variables and parameters. The rule specification language must permit use of any type of object as a variable or parameter and must allow for the specification of parameterized queries containing update operations against EIS-managed data. The EIS, in combination with the rule processor, must be able to support the concept of parameterized messages and programs, and must be able to supply the parameter instantiations automatically." [16]

Thus the EIS Information Model is based on processing information using a multi-rule knowledge base in a multi-base environment. From this foundation the exchange of information among diverse environments is no longer a matter of format, but is one of semantics.

Summary

This paper has covered the descriptive capability and control mechanisms of the VHDL and the Information Model structure of the EIS. It is the purpose of both of these standards efforts to promote the interchange of electronic design data through the semantic content of the data rather than in its physical-/logical format. It is the intent that both of these "tools," a language and an environment, will be platforms from which knowledge based electronics design may continue forward. Internal to the VHDL there exist the necessary control structures and proof mechanisms for the language to be the input to a formal proof of correctness system as done by Dr. Luckham at Stanford University. As has been described above, the EIS Information Model is to be based on known knowledge-base requirements and techniques.

References.

1. Cammarata, Stephanie and Melkanoff, M., "An Interactive Data Dictionary Facility for CAD/CAM Data Bases," Expert Database Systems, Benjamin/Cummings Publishing Co., Menlo Park, CA, 1986, pp. 423-440.

2. King, Roger, "A Database Management System Based on an Object-Oriented Model," Ibid pp. 443-468.

3. Katz, Randal, "Managing the Chip Design Database," IEEE Computer Magazine, Vol.16, No.12, December 1983.

4. Kalay, Y., "A Database Management Approach to CAD/CAM Systems Integration," Proceedings 22nd ACM/IEEE Design Automation Conference, June 1985.

5. Brown, H., C. Tong, Foyster, G., "Palladio: An Exploratory Environment for Circuit Design," IEEE Computer Magazine, Vol.16, No.12, December 1983.

6. Elias, N., Byrne, R., et.al., "The ITT VLSI Design System: CAD Integration in a Multinational Environment," Proceedings 22nd ACM/IEEE Design Automation Conference, June 1985.

7. Preston, G., "Report of IDA Summer Study on Hardware Description Language," HQ 81-23681, Institute for Defense Analyses Science and Technology Division, Arlington, VA, October, 1981.

8. Johannsen, D., McElvain, K., Tsubota, K., "Intelligent Compilation," VLSI Systems Design, April 1987.

9. Janac, George, Carlos, G., Davis, R., "A Knowledge-Based GaAs Design System," VLSI Systems Design, April 1987.

10. Goering, Richard, "Intelligent Silicon Compiler Optimizes ASIC Design," Computer Design, April, 15, 1987.

11. Kemmerer, Richard, "Verification Assessment Study Final Report," C3-CR01-86, Office of Research and Development National Computer Security Center, March 27, 1986.

12. Patton, C. "Languages and Data Formats Vie As Potential Standards in the CAE Design Loop," Electronic Design, Vol.32, No.26, December 27, 1984.

13. Linn, Joseph, Winner, R. editors, "The Department of Defense Requirements for Engineering Information Systems," P-1953, Institute for Defense Analyses, Alexandria, VA, July, 1986.

14. Ibid paragraph 3.24.

15. Ibid paragraph 3.10.

16. Ibid paragraph 4.10.

DEVELOPMENT OF A COUPLED EXPERT SYSTEM FOR THE SPACECRAFT ATTITUDE CONTROL PROBLEM

K. Kawamura, G. Beale, J. Schaffer, B.-J. Hsieh, S. Padalkar
and J. Rodriguez-Moscoso
Center for Intelligent Systems
Vanderbilt University
Nashville, TN 37235

F. Vinz and K. Fernandez
National Aeronautics and Space Administration
Huntsville, Alabama 35812

Abstract

A majority of the current expert systems focus on the symbolic-oriented logic and inference mechanisms of artificial intelligence (AI). Common rule-based systems employ empirical associations and are not well suited to deal with problems often arising in engineering. This paper describes a prototype expert system which combines both symbolic and numeric computing. The expert system's configuration is described and its application to a space craft attitude control problem is presented.

Introduction

Current NASA planning to develop a low earth-orbit Space Station poses a unique opportunity for the development of an expert system for coupling symbolic processing and numerical computations.

Computer simulations are used extensively by NASA to verify system design. These simulations are developed by highly skilled simulation specialists and the complexity of these simulations require that the specialist be involved in the operational phase as well as in development. This results in a poor utilization of personnel. An expert system coupling symbolic processing and numerical computations may solve this problem by permitting detailed experiments and studies to be performed without the investigator's need to have a detailed knowledge of the model implementation.

Development of the Space Station will also require close coordination among system designers from NASA, the aerospace industry and other participants. An intelligent system with enough knowledge of system design may be able to assist in this coordination. Such a system could interact with each system designer in an intelligent way, allowing for the exploration of alternative designs, pointing out potential problems, catching forgotten details, etc. (De Jong (1983)). This system could also inform the other members of the design team of critical decisions made.

Most current expert systems focus on symbolic reasoning and inference mechanisms and traditionally have not been concerned with the numerical processes frequently used on engineering problems (e.g., the simulation of dynamic systems) (Kawamura (1985a)). However, the intelligent use of these numerical methods involves the kinds of expertise with which AI has dealt, and which is frequently in short supply.

Recognizing such a need, NASA's George C. Marshall Space Flight Center awarded a contract to the Vanderbilt University Center for Intelligent Systems to develop an expert system to run a class of spacecraft simulation programs. This contract had the following long-range objectives:

1.) To create an expert system that can assist the user in running a variety of simulation programs employed in the development of the Space Station.

2.) To create an expert system that understands the usage of a NASA-supplied simulation and that can assist the user in the operation of various features of this simulation.

As an initial step toward development of such an intelligent system, an expert system called NESS (NASA Expert Simulation System) was developed, which understands the usage of a class of spacecraft attitude control simulation software and can assist the user in running the software. NESS was build using a knowledge-engineering tool called GENIE (GENeric Inference Engine) (Sandell (1984)), developed at Vanderbilt University. The simulation software

represents a simplified model of a typical spacecraft. It has many of the same functions which appear in the simulation software of an actual spacecraft. The purpose of the generic simulation model is to serve as a test-bed simulation during the development of NESS. Since it was developed at Vanderbilt, the generic simulation model is well understood and is easily modified. Its use made the understanding of how to interface expert systems to simulation programs much easier than if an actual simulation program had been used.

COUPLED EXPERT SYSTEM

Design Principle

One of the major design decisions of this project was to maintain a clear separation between the generic simulation model, which performs the numeric computations, and the expert system, which performs symbolic processing. This parallels the situation in which a human expert sets out to perform a numerical simulation experiment. The human expert, using his or her knowledge of the system to be modeled and the characteristics of the simulation software, makes decisions about how to run the experiment. These decisions are then frequently implemented by creating an input file to be read by the general purpose simulation software. This file contains parameters describing the simulation model and switches which inform the program of the options selected by the user. The user then issues a command to the operating system to run the simulation program. If it runs without error, the user then examines the output files and interprets the results.

Following this approach, the expert system (NESS) was designed as a software system separate from the generic simulation model. Figure 1 shows the interaction of these two systems schematically. Each system was written in the language most natural to it. The generic simulation model was written in FORTRAN following years of traditional engineering practice, and NESS was build using a general inference engine (GENIE) written in FRANZ LISP, following current AI practice.

The knowledge-base of NESS contains three types of knowledge: general knowledge about spacecraft attitude control simulation experiments; specific knowledge of the input parameters and their formats required by the generic simulation program; and self-knowledge which is used to prevent foolish behavior, such as attempting to examine results before a simulation run has been executed.

The user-interaction scenario is envisioned as follows. The users invokes NESS which queries him or her about the system to be modeled and the experiment to be performed. This interaction should avoid requiring the user to specify all the low-level parameters. Rather, it should concentrate on the major engineering decisions required to get an answer to your questions. The setting of the low-level parameters should be inferred and performed by NESS, using its knowledge. After gaining sufficient information to specify a complete experiment, NESS runs the simulation and checks for run-time error messages from the operating system.

NESS should then examines the output files created by the simulation model and interpret them for the user in light of his or her major questions. This involves exhibiting plots of model responses and comments on the stability of the proposed system design.

System Architecture

NESS was designed using frames, agendas, menu-inputs and rule-bases, all of which ar facilities provided by GENIE. Frames are one of the basic data structures currently used in AI. Agendas provide the control information necessary for running an expert system. A menu-input stage is used to gather information from the user. Rule-bases containing individual rules store knowledge obtained from a domain expert. The architecture of NESS is illustrated in Figure 2.

As can be seen from Figure 2, NESS consists of five specific functional modules controlled by a top-level Manager. The Model Instantiator obtains initial parameter values from the user, the Simulation Executor runs the simulation model, the Librarian stores and retrieves parameter values from disk files, the Parameter Editor allows the user to edit parameter values, and the Graphics module displays the simulation results. The top-level Manager controls the firing of each of the five modules by means of a forward-chained rule-base. This architecture along with the rule-base control results in a modular, flexible and expandable expert system.

System Implementation

FRANZ LISP provides a number of ways in which a LISP process such as NESS can effect operating system calls. These calls allow NESS to do things like write a disk file containing the parameters that the simulation program needs, cause its execution and read the output files it creates (as illustrated in Figure 1).

The most straightforward utilization of a system call is simply to include a FRANZ LISP function "exec" (Foderaro (1983)) to cause the execution of a standard UNIX commands, directly in a rule clause. For example, output_display_rb_rule6 checks the precondition that insures that the simulation program has run (self knowledge) and that the user wants to see a plot of theta, which the simulation program would have deposited in a disk file called "theta0plt.stp." The 'then' side of the rule looks as follows:

($then (exec cat theta0plt.stp)).

This causes the UNIX "cat" command to execute, which simply copies the named file (theta0plt.stp) to the user's terminal.

A slightly more involved method is to write a demon (i.e., a special purpose LISP function) to perform some specific operation which may involve one or more calls to UNIX system functions. For example, a demon named "setup_init_val_in_simula.inp" calls the system function "fileopen," "close," and "cprintf," which performs formatted file write operations. This demon is called by run_rb_rule1.

This rule also calls the demon "start_sim," which uses the FRANZ LISP function "process" to fork a child process, which is the actual execution of the simulation.

Currently NESS and the generic simulation model reside on a VAX 11/785 running under the VMS/EUNICE operating system.

GENERIC SIMULATION MODEL

Spacecraft Attitude Control Problem

The function of a spacecraft attitude control system is to maneuver a space vehicle into a certain orientation defined by a reference vector, and to maintain that orientation over an extended period of time. As an example of attitude control, consider the pointing control system for the Space Telescope (Dougherty (1982)). The control system must maneuver the telescope through a 90 degree arc in less than 20 minutes, and then maintain a stable line-of-site to within 0.007 arc-seconds for 24 hours. Thus, the control system must be designed to maneuver through a large change in direction and then track the vehicles's position about a constant direction. The vehicle's dynamics could be represented by nonlinear differential equation during the maneuvering mode; in the tracking mode, the equation could be linearized about the desired operating point. In the initial phase of our project, only the simulation of the vehicle and control system during the tracking mode was considered.

The commanded inputs to the control system would generally be angular position. Both angular position and angular rate would be measured by star trackers and rate gyros, and these measurements would be available to the control system. The torque required to accomplish the maneuvers would be provided by a set of control moment gyros (CMGs). Generally, redundancy in sensors and actuators could be a design feature of the control system. For example, four sensors could be positioned to measure the variable in three-dimensional space such that any three of the sensors would provide linearly independent measurement. With this type of configuration, all four sensors could be used and consistency checks made on the measurements. If any one sensor failed, the remaining three could provide complete coverage of the desired variable. Figure 3. is a simplified illustration of the pointing control system for the Space Telescope. The controller, reaction wheel assemblies, and rate and position sensors mentioned above can easily be identified in the figure. The Fine Guidance Sensor block is used in different ways for the different modes of searching for a new target, course tracking of the target, and finally maintaining an attitude locked onto the target.

One factor which makes the control of a space vehicle more involved than the traditional position control problem is the need to use several coordinate frames in defining the vehicles's location and orientation. It is common practice for the vehicles's attitude to be specified by a series of transformations from an inertial frame to frames that are geocentric, defined in the orbital plane, and dependent on orbital shape. In addition, the vehicle's orientation is defined relative to a local vertical frame defined at the vehicle's center of mass and oriented with respect to the orbit normal and local vertical directions. Other reference frames are defined fixed within the vehicle at the location of sensors, actuators and bending modes; these internal frames relate sensor data, generalized forces, and bending deformations, respectively, to the vehicle's dynamic equations.

Transformations are possible between those various coordinate frames (Brady (1982) and Paul (1981)). A matrix can be

defined which can multiply a vector in one coordinate frame to convert it into the equivalent vector in a second coordinate frame. Transformation matrices can be defined in terms of roll-pitch-yaw angles between the coordinate frames or in terms applied to spacecraft attitude control problems through the concept of Quaternions (Ickes (1970) and Grubin (1970)). A Quaternion is a four parameter system composed of a vector about which the rotation is to be made and a scalar which is a measure of the angle of rotation.

Model Overview

The generic simulation model shown in Figure 4 represents the simulation of the spacecraft and control system during the tracking mode.

A simple spacecraft attitude control system would have a minimum number of three (3) coordinate frames. These coordinate frames would represent the inertial coordinate system, the actual vehicle orientation, and the target reference direction. Zero position error is achieved when the reference and vehicle coordinate frames are identical to each other. These three frames are used in the generic simulation for this research project. The function of the attitude control spacecraft until its coordinate frame becomes identical to the reverence coordinate frame, and then to maintain that orientation until new reference direction commands are given. The differences between actual and commanded angular positions and actual and commanded angular rates would be used by the control system as the error signals used to compute command signals for the actuators. These signals would command torque from the actuators about an axis defined by the Quaternion Transformation to force the errors to zero. This amounts to determining the transformation matrix between the current vehicle orientation and that of the reference vector, and determining the control signals necessary to physically implement that transformation matrix.

Implementation Status

Currently we are running NESS with a simplified simulation model, i.e., there are no bending modes, the inertia matrix is diagonal, and the controller is of the PID (proportional-integral-differential) type. The input command is angular attitude. Runge-Kutta and linear multistep integration algorithms are available for performing the numerical integration of the equations of motion. The simplified model (Prototype I) is shown in Figure 5.

SAMPLE SESSION WITH NESS

The primary knowledge in NESS is concerned with gathering input data for the simulation experiment. NESS asks the user to provide initial values of some parameters and obtains other by asking questions from which it can infer them. This is done in a systematic manner as follows:

a) NESS gathers all the data required to define the system to be simulated. This includes getting values for the inertial and controller matrices, initializing the Quaternion module and selecting a method of integration.

b) NESS asks for the type of response to be obtained from the system. A choice of STEP or FREQUENCY response is offered.

c) NESS then completes the set of parameters required to run the simulation experiment.

CONCLUSIONS AND FURTHER WORK

This paper has illustrated an expert system that can assist the user in running a class of spacecraft attitude control simulations. Although the knowledge-base and the simulation model are relatively simple and limited, we have demonstrated the coupling of symbolic processing and numerical computation. That was the purpose of Phase I of this research (Kawamura (1985b)).

In the subsequent phase, the capabilities of both the simulation model and the expert system will be extended. The simulation model will be extended to include actuator and steering distribution equations. Bending modes are being added in the body dynamic equations since they represented a significant concern to the control system designer. The expert system is being extended to assist the user in running a wide variety of simulation models. It will interpret the output data to determine system characteristics such as percent overshoot, settling time, gain margin and phase margin. Ness will also be extended to recommend a suitable series compensator to be added to the simulation model that is required to achieve the desired frequency or time response e.g., achieve a specified overshoot or phase and gain margins.

REFERENCES

[1] De Jong, K., Intelligent Control:

Integrating AI and Control Theory, Proc. Trends and Applications 1983, National Bureau of Standards (1983) 158-161

[2] Kawamura, K., Coupling Symbolic and Numerical Computations, Proc. 1985 IEEE International Conference on Systems, Man and Cybernetics (1985a) 507-510.

[3] Sandell, H., Bourne, J. and Shiavi, R., GENIE: A Generic Inference Engine for Medical Applications, Proc. Sixth Annl. Conf. IEEE Engr. Med. Biol. (1984) 66-69.

[4] DeCeraro, J.K. (ed.), et al., The FRANZ LISP Manual (University of California, Berkely, 1983).

[5] Dougherty, H., et al., Space Telescope Pointing Control System, Journal of Guidance, Control, and Dynamics 5(4) (1982) 403-409.

[6] Brady, M. (ed.), et al., Robot Motion: Planning and Control (MIT Press, Cambridge, 1982).

[7] Paul, R.P., Robot Manipulators: Mathematics, Programming and Control (MIT Press, 1981).

[8] Ickes, B.P., A New Method for Performing Digital Control System Attitude Computation using Quaternions, AIAA Journal 8(1) (1970) 13-17.

[9] Grubin, C., Derivation of the Quaternion Scheme via the Euler Axis and Angle, Journal of Spacecraft 7(10) (1970) 1261-1263.

[10] Kawamura, K., Beale, G., Schaffer, J., Hsieh, B.-J., Padalkar, S., and Rodriguez-Moscoso, J., Research on an Expert System for Database Operation of Simulation/Emulation Math Models, NASA Phase I Final Report Vol. I and II Contract #NAS8-36285, Center for Intelligent Systems, Vanderbilt University. (August, 1985b).

NOTE:

This edited paper describing NESS Phase I appears in its complete form as a chapter in Coupling Symbolic and Numerical Computing in Expert Systems edited by Janusz Kowalik and published by the North-Holland Company in 1986. Phase II of NESS was completed in May of 1986, and it resulted in the development of more general interface specifications allowing NESS to interact with a wider range of digital simulations. Emphasis was also placed on incorporating knowledge specific to the design of series control system compensation yielding a system that assists the control system designer in achieving desired system performance. The results of Phase II are documented in the Final Report to MSFC on Contract #NAS8-36285, Center for Intelligent Systems, Vanderbilt University (May 1986).

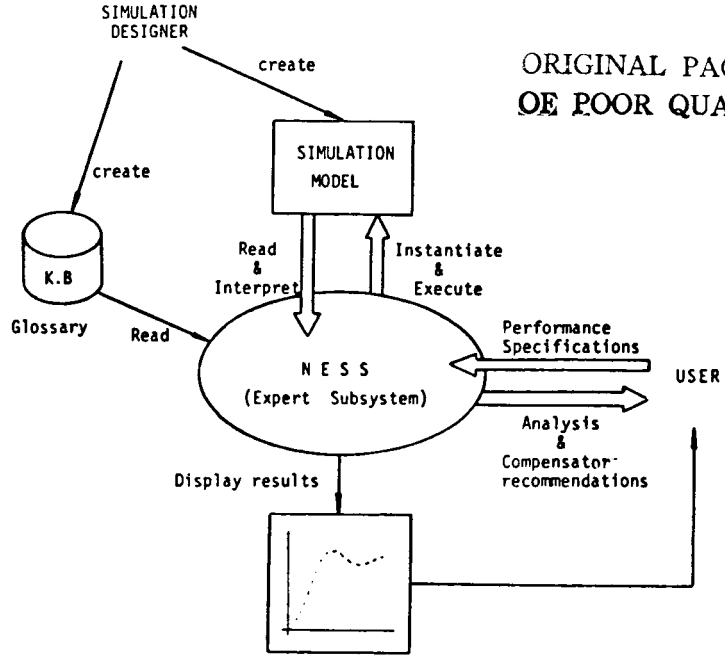


Figure 1 User Interaction with the Coupled Expert System

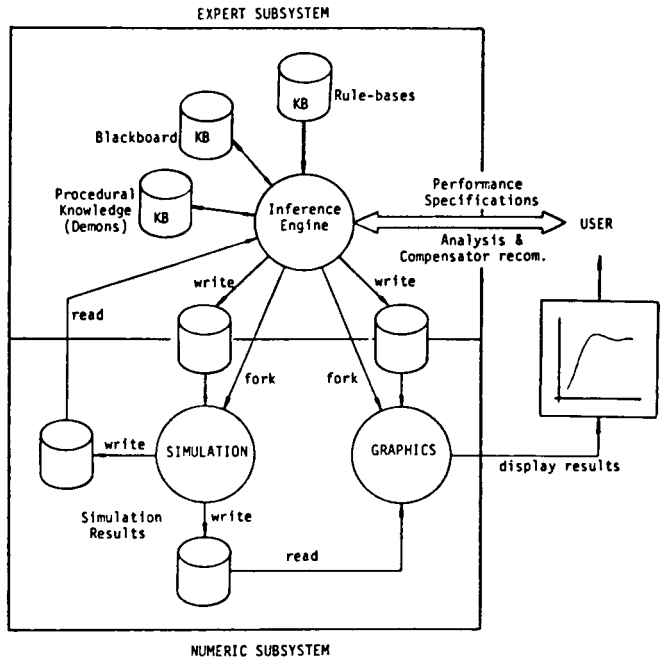


Figure 2 NESS Architecture

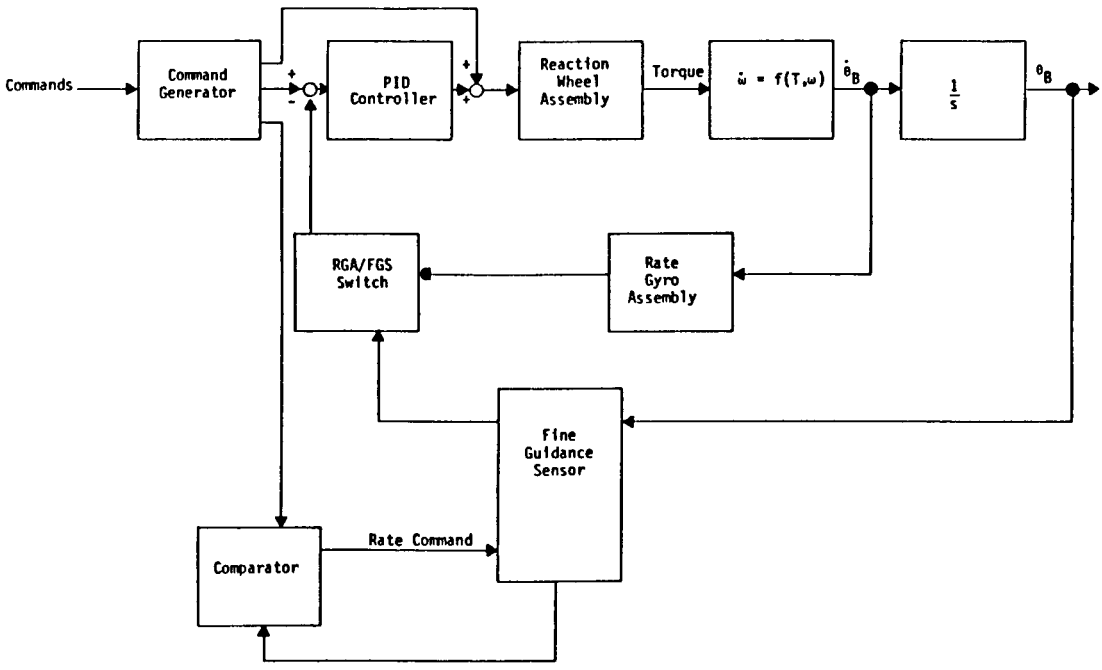


Figure 3 Simplified Attitude Control System

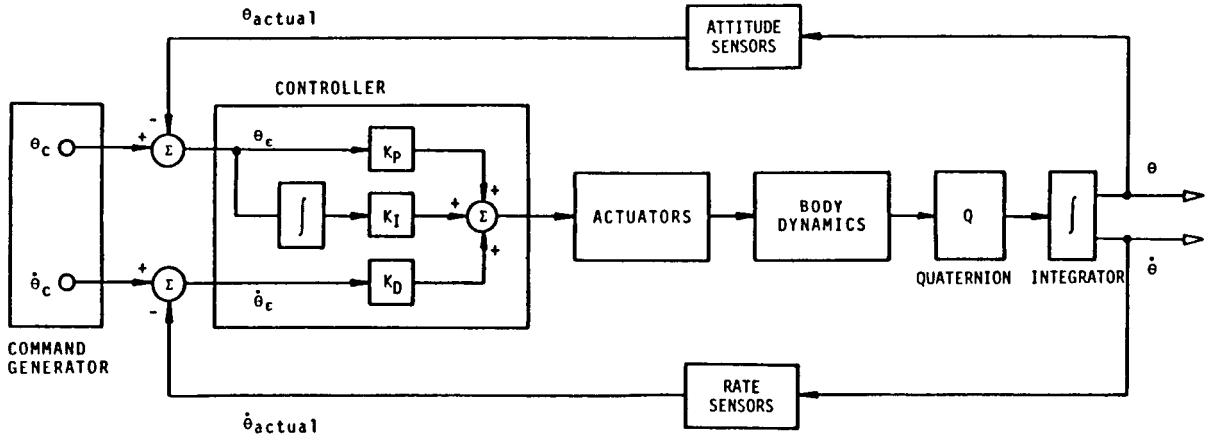


Figure 4 Block Diagram of the Generic Simulation Model

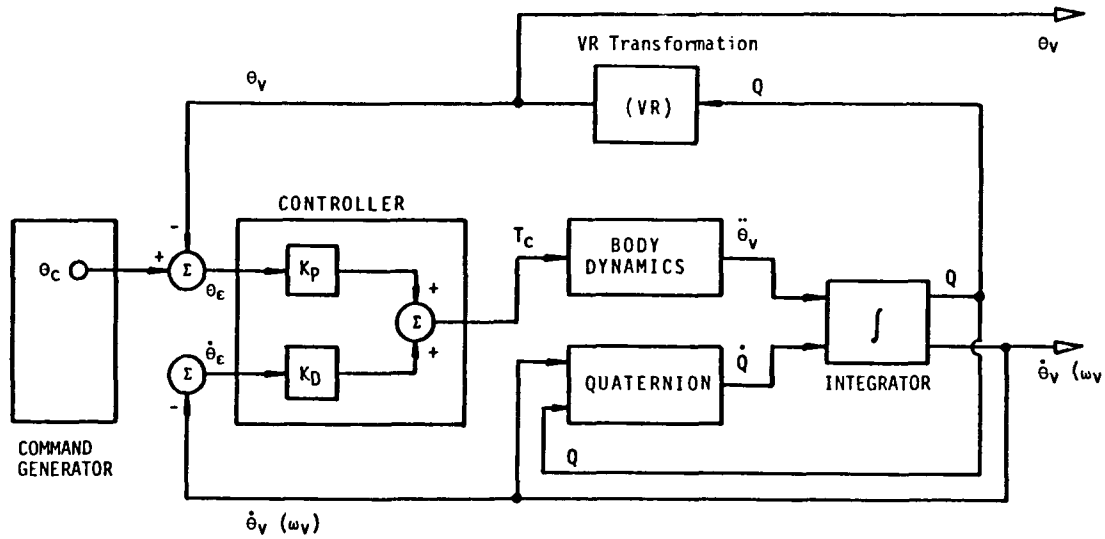


Figure 5 Simplified Diagram of the Generic Simulation Model

USER INTERFACE DEVICES FOR MISSION CONTROL

Wayne Boatman
JSC/FS93
Johnson Space Center, Texas 77058

ABSTRACT

The Mission Control Center (MCC) at Johnson Space Center (JSC) in Houston, Texas is being upgraded with new technology engineering/scientific workstations. These workstations will replace the existing consoles and will emulate the present hardware input and display media. The workstations will be using new and different input devices for the flight controller to interact with the workstation and mainframes. This paper presents the results of the USER INTERFACE survey conducted by the Workstation Prototype Lab (WPL). The WPL offered the opportunity for users to do "hands-on" evaluation of a number of user interface options that Lab personnel had prototyped.

INTRODUCTION

The System Development Division's Workstation Prototype Lab (WPL) demonstrated several interactive computer input devices to space shuttle flight controllers. The input devices that were presented as possible candidates for the future Mission Control Center (MCC) included the following:

a) Touch Screen - Comprised of two sheets of plastic film placed over the terminal display whose inner sides are coated with a resistive substance.

When the screen is touched, an analog signal corresponding to that point is generated and sent to the associated controller box. The control box then converts the analog signal to a digital value which is used by the host workstation to determine which point on the screen was touched.

b) Mouse - Used to control the terminal screen's cursor

The optical mouse is moved over a special tablet to initiate movement. The mechanical mouse is similar to an "upside-down" track ball and is moved on a smooth surface to initiate cursor movement. Once the cursor has been moved to the desired screen position, a button or a series of button would be pressed to initiate a particular function.

c) Joystick - Used to control the terminal screen's cursor.

The cursor is moved in the same direction as the joystick lever. Once the cursor is placed on the desired screen position, a button or a series of buttons would be pressed to initiate a particular function.

d) Keyboard/Key Pad - Uses the standard keyboard and special keypads to manipulate a cursor position on a screen or to invoke a user selection.

This input device also includes all function keys, directional arrows, numeric keypads, etc.

e) Continuous Voice Recognition System - Accepts a speech input that is transmitted electronically to the voice recognition system via microphone.

The system takes the analog voice signal, digitizes the pattern, and stores this pattern in its memory. These sets of patterns can be stored on small floppy disks or on a host workstation. Then, each time a waveform is received be the system, it performs a pattern match to the digitized "vocabulary" previously stored in memory. If a good match is found the work is said to be "recognized" and its predefined function is send to the workstation to execute a command. The voice system also has the capability to playback recorded messages and response messages from the host computer.

USER INTERFACE SURVEY

To provide the benefit of hands-on experience, a program was created which guided the users through a series of demonstrations designed to show how these devices could be applied to practical problems facing future users of the next generation MCC software. This program provided the user with a means of calling up various displays and simulating the initiation of Orbiter commands.

Over 150 people participated in the original demonstrations and each was asked to make inputs to a relational database with their rating on each device as applied to a specific application. Since the original demo, another 400 people have seen the workstation input devices. In the table below, results from the original evaluation is shown (see Table I). Note that 9.0 is the highest possible rating and 1.0 is the lowest possible rating.

CONCLUSIONS

The results from these demonstrations shows that the Mouse and Keypad were the preferred input devices for the flight controllers. Additionally, there was a limited number of votes for the touch screen. It was determined the mouse was very good for grabbing and dragging an object on the screen. The mouse will be used for moving and resizing windows and for building displays with a graphics editor. The keypad software is application specific, but with compiler libraries provided by the workstation vendors, software with keypad input can easily be written. The touch screen input will be offered only as an option to the MCC upgrade program.

TABLE I

Command Demo	Device Type	Display Demo
7.7	Touch Screen	7.9
5.4	Mouse	5.7
3.0	Joystick	3.3
4.7	Keypad	7.0
*	Voice	5.5

* Voice Input was not applicable for the Command Demo

THE DESKTOP INTERFACE IN INTELLIGENT TUTORING SYSTEMS

Stephen Baudendistel Grace Hua

COMPUTER SCIENCES CORPORATION
Applied Technology Division
16511 Space Center Blvd.
Houston, TX 77058
(713) 280-2430

Abstract

The interface between an Intelligent Tutoring System (ITS) and the person being tutored is critical to the success of the learning process. If the interface to the ITS is confusing or non-supportive of the tutored domain, the effectiveness of the instruction will be diminished or lost entirely. Consequently, the interface to an ITS should be highly integrated with the domain to provide a robust and semantically rich learning environment. In building an ITS for ZetaLISP on a LISP Machine, a Desktop Interface was designed to support a programming learning environment. Using the bitmapped display, windows, and mouse, three desktops were designed to support self-study and tutoring of ZetaLISP. Through organization, well-defined boundaries, and domain support facilities, the desktops provide substantial flexibility and power for the student and facilitate learning ZetaLISP programming while screening the student from the complex LISP Machine environment. The student can concentrate on learning ZetaLISP programming and not on how to operate the interface or a LISP Machine.

Introduction

Artificial Intelligence techniques are now beginning to be applied to the area of education, in particular to the development of Intelligent Computer Assisted Instruction (ICAI). Frequently, the ICAI is in the form of Intelligent Tutoring Systems. Figure 1 depicts a typical ICAI architecture [9]. The area of the ITS most frequently addressed to date has been the student model. By contrast, the interface has been minimally addressed. Yet the interface is the student's contact with every component of the tutor. If the student cannot get past the interface, the quality of the student model or of any other component of the ITS will not matter. Consequently, the interface must be a high priority in the development of any ICAI [17].

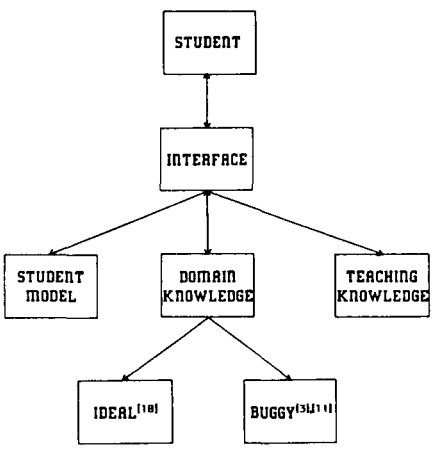


Figure 1. A typical Intelligent Computer Assisted Instruction (ICAI) Architecture.

This paper will describe the implementation of an ICAI interface, referred to as the Desktop Interface, for a ZetaLISP Intelligent Tutoring Assistant (ZITA). To date the ZITA student model has been only minimally implemented, while the emphasis has been on developing an interface which would support and encourage learning to program in ZetaLISP on a LISP Machine. In fact, the Desktop Interface is intended to provide much more than a typical user interface; it is to provide a Programming Learning Environment (PLE) [19]. Moreover, the Desktop Interface is presented as an authoring vehicle for developing programming language tutors for languages in addition to ZetaLISP.

In promoting the ITS interface, we are not advocating a position of ignoring components of ICAI other than the interface or of producing a glittering interface with no underlying substance. Ideally, all the components would be highly integrated. However, up to this point, more attention has been devoted to the more glamorous components: the Student Model and the Domain Knowledge. We do not want the gains made in these latter components diminished or lost because the learning environment does not foster and facilitate learning. Unpleasant experiences with frustrating, difficult interfaces will not advance ICAI, but rather retard it. Our ideal tutoring environment is one which seems invisible to the student but which supports the intuitive operational expectations of the student relative to the domain being tutored.

Background

In the past five years important advances in graphical presentation capability have made possible a new, powerful method of communication. Bitmapped, graphical windows and the mouse have resulted in proven techniques for reliable, high-bandwidth information exchange between people and computers [21] which more closely model human cognitive processes, especially with the use of metaphor and frames [5]. With these capabilities we can move far beyond the limitations imposed by static CRT screens with 25 lines of 80 characters. Previously such capabilities have required expensive, multi-MIPS computers. But the decreasing cost and increasing power of microcomputers now make such capabilities readily available for ICAI. Indeed, we should demand windows and mice, and refuse to consider systems limited to complicated keystroke patterns and displaying a few lines of text.

Criteria for Developing Tutoring Environments

While the tutoring environment must be designed with the specific domain in mind, some general criteria for developing tutoring environments have begun to emerge [24]. Environments should be intuitive, obvious and fun. The use of metaphor, icons, and the mouse should take advantage of student intelligence, experience and resourcefulness. Environments should provide high-bandwidth communication between the student and the tutor. Designers should be motivated by teaching and cognitive knowledge about how experts perform tasks in the subject domain. Environments should isolate key tools for attaining expertise in the domain. Environments should maintain fidelity with the real world (in learning programming, the student should be able to run both examples and problem solutions). Environments should be responsive, permissive, and consistent based on skills students already have rather than forcing them to learn new skills. Finally, all tools should be based on similar interface devices such as menus, mouse clicks, etc.

A ZetaLISP Tutor

We currently have a task with the Artificial Intelligence Section of the Mission Planning and Analysis Division (MPAD) of NASA's Johnson Space Center (JSC) to provide training in AI topics (Common LISP, ZetaLISP, LISP Machines, CLIPS, ART). The ZetaLISP tutor has been developed on an as-time-permits basis to complement our ZetaLISP class. In designing the ZetaLISP tutor, two goals were established. First, we wanted an effective environment for tutoring ZetaLISP on a LISP Machine. Secondly, we wanted to develop a general programming learning environment for computer applications languages. In particular, we wanted a PLE which could be duplicated on workstations and the upcoming, more powerful personal computers.

One must make a number of assumptions when implementing a tutor. Ours were as follows: the student would be a technical professional employed by NASA or its contractors; the student would have the equivalent of 40 hours of Common LISP training and 8 hours of hands-on training in the use of a LISP Machine; the tutor would supplement our classroom ZetaLISP training; the tutor could evolve to be used by persons who had completed the ZetaLISP training (about 45 hours) and were interested in obtaining more experience or were seeking examples to help in their current tasks.

The coaching system of ZITA evaluates the student's performance through a differential modeling technique, comparing the student's progress to an ideal solution step-by-step, intervening immediately when it perceives the student has made a mistake [4], [18]. At this stage of development, the immediate intervention issued by the tutor primarily points out syntactic errors and noise level errors made by the student presumably due to negligence and fatigue. Based on the previous assumption of the student's background, these errors are not considered to have resulted from misconceptions in learning.

Learning to Program and the PLE

How could an appropriately structured environment facilitate the acquisition of programming skills [16]? In order to answer this question, we first investigated some of the aspects of learning to program. Three aspects of learning to program were to be supported by our PLE [1]. First, the PLE was to help the student organize and compile problem-solving operators for programming. Learning to program involves recognizing appropriate goals and decomposing the goals into subgoals until goals are reached which correspond to code. Secondly, the PLE was to represent the relevant knowledge, both declarative and procedural, in ways which correspond to the cognitive representations of programmers, because one's representation of a problem has strong impact on one's problem-solving ability. Thirdly, the PLE was to act as an external memory device for programmers to reduce the impact of human memory limitations. Approximately 50 percent of LISP novices' time is spent recovering from errors of memory [1]. By reducing student working memory load, the PLE will minimize student errors due to memory limitations.

Good programmers are made, not born [23]. B.S. Bloom found that 98 percent of the students with private tutors performed better than the average classroom student. He also found that the greatest learning gains were for the poorest students [2]. The average college graduate is not prepared to perform professional programming tasks without additional training when he or she first arrives on the job in industry. Large sums of money are spent training and retraining programmers with widely varying results. We can improve this process greatly by developing intelligent tutors for learning programming which will provide consistent, cognitively modeled [12] tutoring when and where needed, and at significant cost savings.

The PLE of our ZetaLISP tutor addresses the three aspects of learning programming described above in four ways:

a) Learning by example [20], [10], [4];
b) Facilitating knowledge representation;
c) Reducing student working memory requirements;
d) Unleashing the power of the computer on the ICAI interface.

The PLE is based on learning by example. Examples are critical to learning and to the structure of knowledge and memory. Learning by example provides the student with early, positive experiences and lays down a solid foundation on which to build. Examples help the student organize and compile the use of appropriate operators for programming. Examples illustrate goals and subgoals appropriate to a particular language but which may not transfer to or from other languages. Techniques recalled from examples help reduce the number of steps to produce a solution in similar problems. Novices use examples to generalize solutions, set limits to those generalizations, make recipes for standard tasks, and as a basis for retrieval and modification approach to generating other examples.

Adult students only acquire effective use of problem-solving knowledge by practicing with a series of examples and problems [19]. Adults prefer learning by doing rather than watching because it makes the subject immediately useful and meaningful [22]. Studies by the Xerox Corporation confirm that learning occurs 50 percent faster with active, hands-on training than when the learning is passive [13]. Adult learners seek a focused, applicable treatment of the subject so they can transfer the concept to their work. Generalities are acceptable only when they lead to specific information and ideas. Adults are highly motivated to apply their learning to their work and are willing to assume responsibility for learning. Adult learning uses experience as a resource. Adults feel rewarded when the learning enriches their experience. Material that provides options is more appealing to adults than material that locks in one approach. Examples reinforce and strengthen the link between the concept and application transfer, rewarding the learning experience and disposing the student toward further knowledge.

The PLE facilitates programming knowledge representation as used by the expert. Not only is syntactic knowledge represented, but more importantly, much implicit semantic knowledge, acquired over many years of experience, is presented to the student. Techniques illustrating when, what, and how to extend specific knowledge in the examples to solve new problems (extrapolate) [15] must be taught. Human learning occurs as a search in a problem space [12] and the desktop interface of the PLE helps constrain and focus the search. Each learning state and operators are well defined for each desktop in the PLE. Chunking is well suited to learning because it is a recorder of goal-based experience; it caches the processing of a subgoal in such a way that a chunk can substitute for the normal, possibly complex,

processing of the subgoal the next time the same or a similar subgoal is generated [11]. Each exercise is a chunking process of storing both knowledge and links to appropriate, related knowledge.

Memory load is minimized by the PLE. Each desktop of the PLE organizes information by chunking into easily recognized areas, minimizing student memory requirements. Each desktop is self-contained; the information necessary to perform required actions on the desktop is present in a window. Transitions from one desktop to another are accomplished with a simple mouse click on a clearly marked box. By using direct manipulation techniques with the mouse and menus, options are clearly delineated and selected in obvious, foolproof ways. Examples and problems help clearly separate details from general principles and establish limitations when extending operators. Finally, each student can use as much or as little of the instructions and explanations as desired, thus both avoiding information overload and frustration from too little information.

Students fail to learn from ICAI only when there are negative forces set up against learning [23] such as unfriendly, difficult interfaces. By unleashing the power of the computer in creating a seemingly invisible desktop tutorial interface, we provide an ideal programming learning environment. The format of the PLE defines boundaries unobtrusively while leaving the horizons of the domain open for the student to acquire the desired knowledge. Bitmapped windows, the mouse, and high-powered (MIPS, memory, windowing operating systems), low-cost, microprocessor-based computers have made possible high-bandwidth, self-evident ICAI interfaces.

The Desktop Interface Implementation of the PLE

The Desktop Interface implemented for the ZetaLISP PLE resembles a desk with relevant documents spread out neatly on it; because there are several discrete stages in the PLE, there is a separate desktop for each stage. Each desktop is divided into four or five parts (windows) with each part representing one document; if a document cannot be seen completely in its window, the window scrolls (using the mouse) to permit unseen sections to be read. People can deal with from four to seven chunks of data at one time [8]. The division of the desktop into less than seven chunks is designed to fit this cognitive model and thereby to limit the student working memory load. Desktops and windows are consistent in format and function. Each desktop must be self-contained so that the student can concentrate on learning the desired knowledge of the domain and not on operating the interface or searching books for additional information. All options are selected with the mouse. Code for examples and problem solutions can be executed by clicking the mouse on an appropriate menu item. The student can hardcopy the window contents for easier reading, making notes, or for future reference [23].

Four desktops comprise the Desktop Interface for this PLE. The first three have been implemented; the fourth has not been designed. The first desktop is the Selections Desktop (Figure 2). In the Selections Desktop the student selects, with the tutor's assistance (based on past performance), the topic of study by selecting an example topic with the mouse. This desktop also contains a LISP listener where the student can enter and execute LISP code if desired for any reason. When an example topic is agreed on between the tutor and the student, the student is taken to the second desktop, the Study Desktop (Figure 3).

In the Study Desktop, the student is presented with instructions for the desktop, the code for the selected example topic, explanations for the topic, and a LISP listener. Because so much information about programming is conveyed only by executing programs, the student can execute the code for the example being studied by selecting a box with the mouse at the bottom of the LISP listener (Figure 4). When the student has finished studying the example, he or she can work problems posed by the tutor which are variations of the code of the example studied by selecting a box with the mouse at the bottom of the LISP listener. In this case, the student is taken to the third desktop, the Tutorial Desktop. As before, there are instructions for this desktop and the code of the example from the Study Desktop.

In the Tutorial Desktop, the student clicks the mouse on the menu item "Show Variation Choices" and is then presented with a list of available problems. Once the student selects a problem to work, the code of the problem, which is a variation of the example studied, is loaded in a window (code which the student is to supply is missing, from a few lines to whole functions). Guidelines for working the problems appear in reverse video and a reverse video window appears over the example code window for the student to enter the missing code according to the guidelines (Figure 5). The student enters ZetaLISP code and the tutor

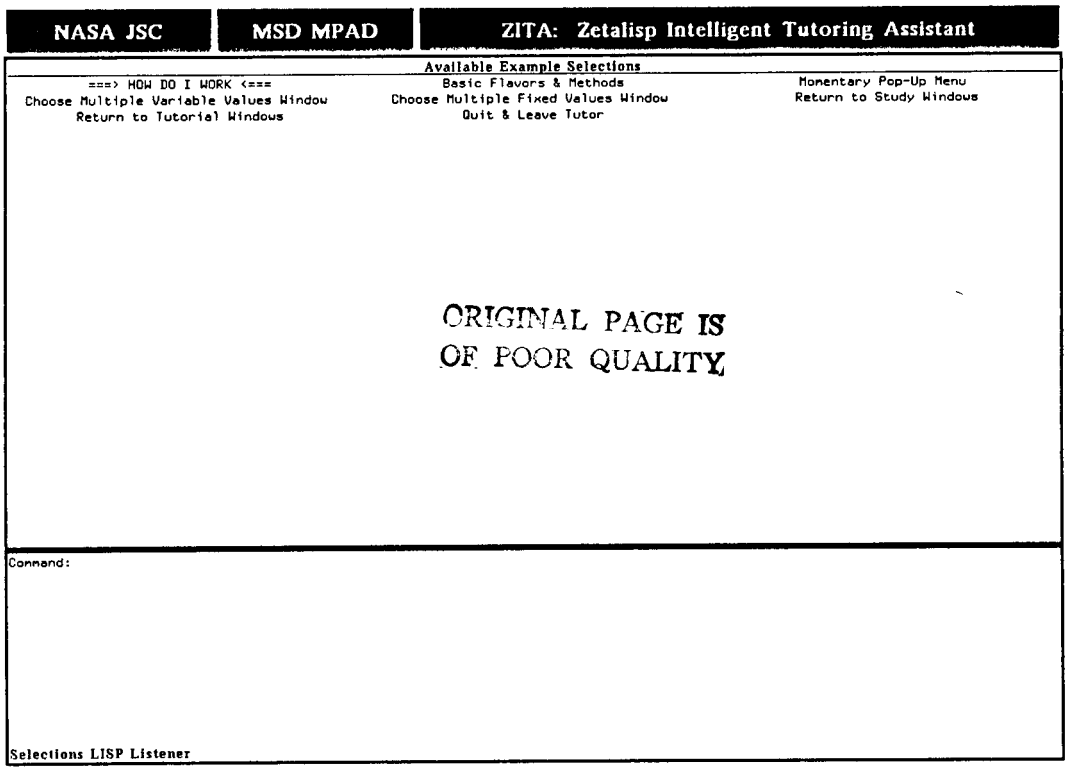


Figure 2. The Selections Desktop of the Desktop Interface.

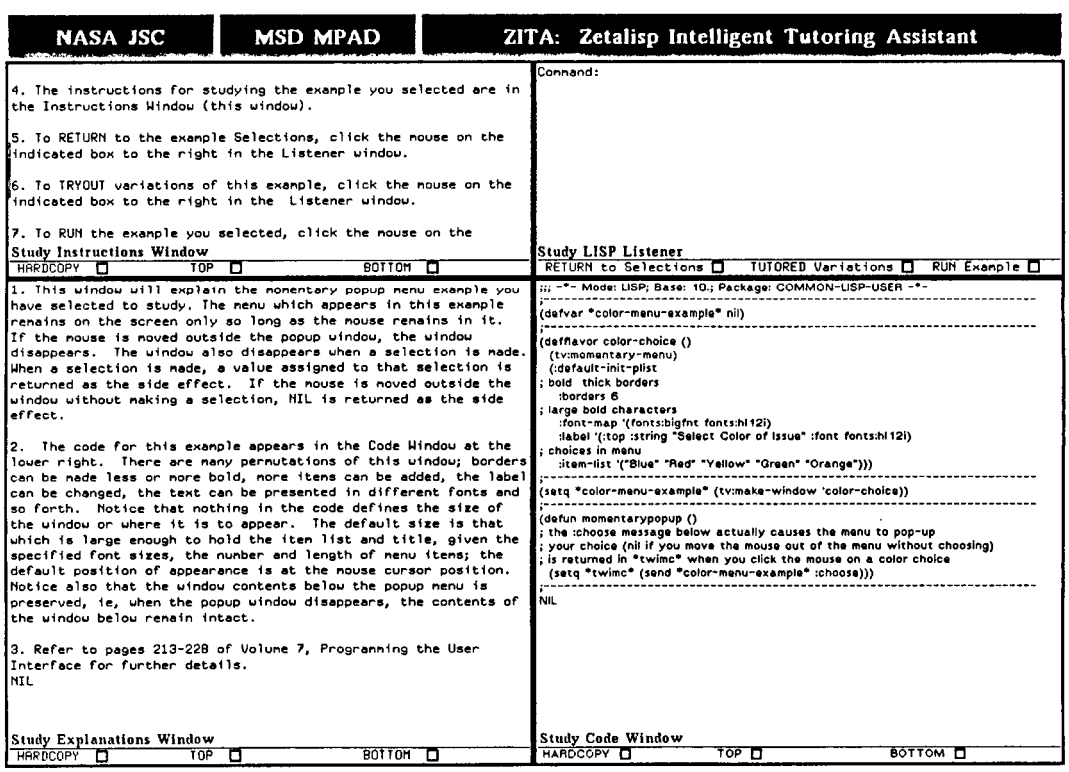


Figure 3. The Study Desktop of the Desktop Interface.

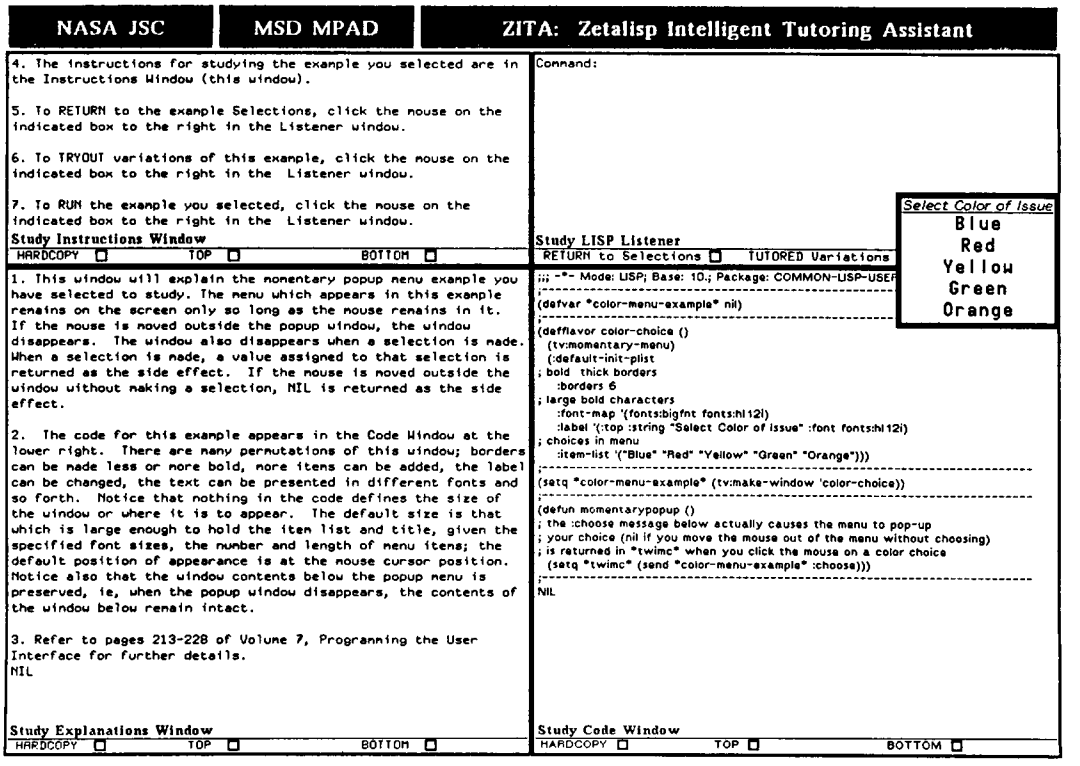


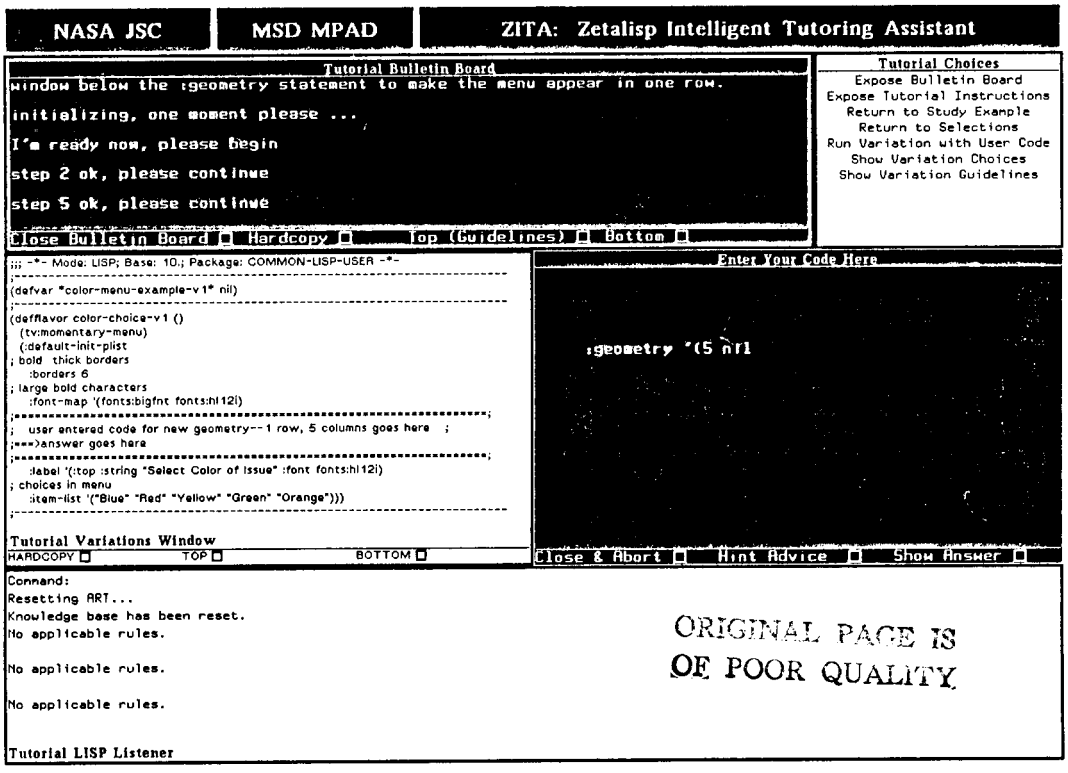
Figure 4. Student executing code for the example being studied on the Study Desktop.

attempts to diagnose bugs and offer corrective dialogue. When the student successfully completes the problem, the tutor inserts the code into the variation code window and the student can execute the problem solution (Figure 6) by clicking the mouse on the menu item "Run Variation with User Code". The student may then select another problem on the current topic or return to the Selections Desktop to choose another topic.

The fourth desktop, the Planning and Goals Help Desktop, has not been implemented yet. Because successful programming requires knowledge of how to both recognize recurring operations and make goals and plans to perform those operations, unsuccessful programmers will exhibit a lack of such abilities. Consequently, the tutor will have to help not only with syntax but also with establishing programming goals and plans. Overcoming this inability is critical if the student is to learn programming [18], [14], [6], [7]. Thus, when the student demonstrates an inability to form correct programming goals and plans, he or she will be transferred to this desktop and will be assisted by the tutor in devising successful goals and plans for the selected problem before being returned to the Tutorial Desktop. Once back in the Tutorial Desktop, the tutor will assist the student in writing code based on the goals and plans developed in the Planning and Goals Help Desktop.

Expectations for an ICAI PLE

We expect the PLE to satisfy a number of sound cognitive principles. The actual layout of the PLE is not important so long as the underlying structure makes the semantics of the domain evident, that is, makes it easy to carry out actions in the domain, and to see and understand the results and implications of those actions. It must support students as they acquire an understanding of the complex semantic domain of programming, minimizing the gap between expectations and actions supported. Certainly it is specialized, highly integrated with the domain and semantically rich with high-bandwidth information transfer between interface and student. It avoids low-bandwidth, semantically weak interfaces which greatly complicate the diagnosis problem. By offering a good match to goals and plans of the student as they



ORIGINAL PAGE IS
OF POOR QUALITY

Figure 5. Student entering code to solve the problem posed by the ZetaLISP tutor on the Tutorial Desktop.

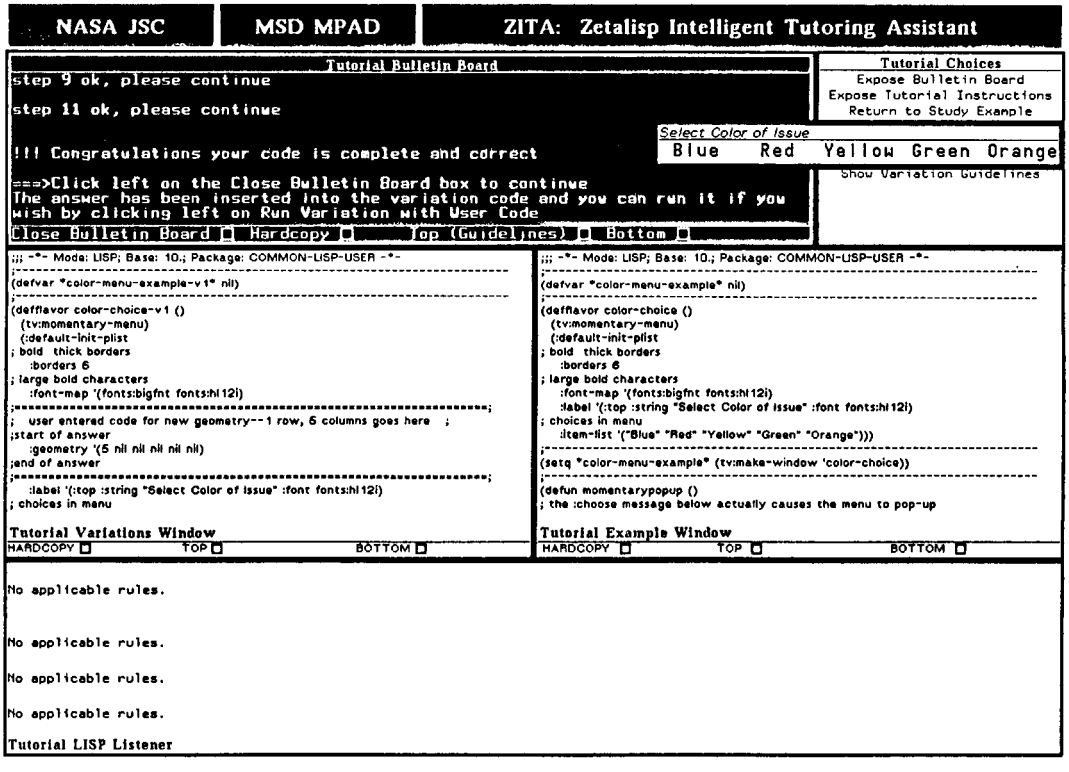


Figure 6. Student executing code for their solution to the problem posed by the ZetaLISP tutor.

learn to program, it accommodates stages of student conceptualization of the domain and how movement from one stage to another takes place. It reflects the task of learning programming, the information that must be presented, and ways in which students may interact with the information, that is, how good programmers organize knowledge and use operators. Serving as an external memory system, the PLE uses the desktop metaphor to organize, standardize, define boundaries, reduce memory requirements, obviate actions/results, and convey a feeling of control.

Conclusions

We now need, and will continue to need, many well-trained programmers. The current method of training programmers is expensive, haphazard, and not founded on an understanding of how to learn programming. Over the past five years we have obtained much knowledge of how to learn programming and, at the same time, computers and software have advanced dramatically in capability while their cost has declined substantially. At this point we have the knowledge and tools available to develop an ICAI Programming Learning Environment and deliver uniform, semantically rich, and cognitively based tutors to train the necessary programmers. The Desktop Interface is a candidate authoring vehicle for such an ICAI PLE. We are continuing, as time permits, to develop and test the Desktop Interface and the Student Model in the ZetaLISP tutor.

References

1. Anderson, J., "Learning to Program," *Proceedings Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, August 8-12, 1983, pp. 57-62.

2. Anderson, J., Boyle, F., Yost, G., "The Geometry Tutor," *Proceedings Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA., Aug 17-23, 1985, pp. 1-7.

3. Burton, R., "Diagnosing bugs in a simple procedural skill," in Intelligent Tutoring Systems, D. Sleeman and J.S. Brown, eds., Academic Press, New York, NY, 1982.

4. Burton, R., Brown, J., "An investigation of computer coaching for informal learning activities" in Intelligent Tutoring Systems, D. Sleeman and J.S. Brown, eds., Academic Press, New York, NY, 1982.

5. Dear, B., "AI and the Authoring Process," *IEEE Expert*, Summer 1987, pp. 17-24.

6. Farrell, R., Anderson, J., Reiser, B., "An Interactive Computer-based Tutor for LISP," *Proceedings Third National Conference on Artificial Intelligence*, Austin, Tx, Aug 6-10, 1984, pp. 106-109.

7. Genesereth, M., "The role of plans in intelligent teaching systems," in Intelligent Tutoring Systems, D. Sleeman and J.S. Brown, eds., Academic Press, New York, NY, 1982.

8. Harmon, P., King, D., *Expert Systems, Artificial Intelligence in Business*, John Wiley & Sons, New York, NY, 1985.

9. Kearsley, G., ed., *Artificial Intelligence & Instruction*, Addison-Wesley, Reading, Mass., 1987.

10. Kolodner, J., Simpson, R. Jr., Sycara-Cyranski, K., "A Process Model of Cased-Based Reasoning in Problem Solving," *Proceedings Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA., Aug 17-23, 1985, pp. 284-290.

11. Laird, J., Rosenbloom, P., Newell, A., "Towards Chunking as a General Learning Mechanism," *Proceedings Third National Conference on Artificial Intelligence*, Austin, Tx, Aug 6-10, 1984, pp. 188-192.

12. Langley, P., Ohlsson, S., "Automated Cognitive Modeling," *Proceedings Third National Conference on Artificial Intelligence*, Austin, Tx, Aug 6-10, 1984, pp. 193-197.

13. Lichtman, D., Watt, P., "Bottom Line Training - Getting Results, Not Classes," *Manage*, Third Quarter 1986, pp. 22-23, 35.

14. Littman, D., Pinto, J., Soloway, E., "An Analysis of Tutorial Reasoning About Programming Bugs," *Proceedings Fifth National Conference on Artificial Intelligence*, Philadelphia, Pa., August 11-15, 1986, pp. 320-326.

15. Matz, M., "Towards a process model for high school algebra errors," in Intelligent Tutoring Systems, D. Sleeman and J.S. Brown, eds., Academic Press, New York, NY, 1982.

16. Miller, M., "A Structured Planning and Debugging Environment for Elementary Programming," in Intelligent Tutoring Systems, D. Sleeman and J.S. Brown, eds., Academic Press, New York, NY, 1982.

17. Miller, J., "Human-computer Interaction and Intelligent tutoring systems," MCC Technical Report Number HI-294-86, April 1987.

18. Orlikowski, W., Vasant, D., "Imposing Structure on Linear Programming Problems: An Empirical Analysis of Expert and Novice Models," in *Proceedings Fifth National Conference on Artificial Intelligence*, Philadelphia, Pa., August 11-15, 1986, pp. 308-312.

19. Reiser, B., Anderson, J., Farrell, R., "Dynamic Student Modelling in an Intelligent Tutor for LISP Programming," *Proceedings Ninth International Joint Conference on Artificial Intelligence*, Los Angeles, CA., Aug 17-23, 1985, pp. 8-14.

20. Rissland, E., Valcarce, E., Ashley, K., "Explaining and Arguing with Examples," *Proceedings Third National Conference on Artificial Intelligence*, Austin, Tx, Aug 6-10, 1984, pp.288-294.

21. Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, Addison-Wesley, Reading, Mass., 1987.

22. Von der Embse, T., "Course Leadership," *Manage*, Volume 39, Number 2, July 1987, pp. 7-8, 33.

23. Weinberg, G., *The Psychology of Computer Programming*, Van Nostrand Reinhold Company, New York, NY, 1971.

24. Woolf, B. and Cunningham, P., "Multiple Knowledge Sources in Intelligent Teaching Systems," *IEEE Expert*, Summer 1987, pp. 41-54.

Dynamic Human Dexterity and Control System (DEXDROID)

(Paper not provided by publication date)

SYSTEMS INTEGRATED HUMAN ENGINEERING ON THE NAVY'S RAPID ACQUISITION OF MANUFACTURED PARTS/TEST AND INTEGRATION FACILITY

Glen R. Gallaway
Battelle Columbus Division
Columbus, Ohio 43201

ABSTRACT

The Human Engineering function in many projects is at best a limited support function. In this NAVY project the Human Engineering function is an integral component of the systems design and development process. Human Engineering is a member of the systems design organization. This ensures that people considerations are: 1) Identified early in the project; 2) Accounted for in the specifications; 3) Incorporated into the design; and 4) The tested product meets the needs and expectations of the people while meeting the overall systems requirements. The project exemplifies achievements that can be made by the symbiosis between systems designers, engineers, and Human Engineering. This approach increases Human Engineering's effectiveness and value to a project because it becomes an accepted, contributing team member. It is an approach to doing Human Engineering that should be considered for most projects. The functional and organizational issues that give this approach strength are described in the paper.

PROJECT BACKGROUND

The purpose of the Rapid Acquisition of Manufactured Parts Test and Integration Facility (RAMP/RTIF) project is to quickly produce quality replacement and spare parts for the Navy which are unavailable when needed. The objective is to make a substantial reduction in the total time required to produce parts to thirty working days after notification of award. Figure 1 shows present supply system responsiveness and the performance requirements to be accomplished in the RAMP/RTIF project.

The RAMP/RTIF Manufacturing System will initially include manufacturing and process planning systems, engineering, production control, and order entry for the production of Small Mechanical Parts (SMP) and for the production of Printed Wiring Assemblies (PWA).

PROJECT HUMAN ENGINEERING ISSUES

The NAVY, from the beginning of the project, has emphasized that people issues require significant and appropriate consideration in system design work. The RAMP/RTIF Human Engineering Program is a response to those concerns by integrating human engineering considerations into the design and development of the RAMP SMP/PWA workcell hardware, software, procedures and facilities. Special emphasis is being placed on human engineering concerns associated with the introduction of automation into manufacturing, administration, fabrication, and maintenance of the SMPs and PWAs. Some of these concerns include ensuring that:

1. Task complexity is not increased by automation, but rather simplified and made more efficient.
2. Newly created user interfaces are designed to be user-friendly, easy to learn, and easy to use.
3. Potential safety hazards are examined and eliminated from the design of user workstations.
4. The user population can effectively, efficiently and safely operate and maintain all equipment/software in the RAMP/RTIF (i.e., users can see, reach, and operate equipment, and can understand and easily use commands and menus in the software).
5. The data and information that people must deal with will be appropriate for their tasks, in a form that will make the task doable within their skill level, and/or appropriate training is provided.
6. The RAMP/RTIF system is being designed to ensure that maintenance personnel can effectively, efficiently, and safely perform maintenance functions.

THE STRENGTH OF THIS PROJECT APPROACH

The resolution to the NAVY's request for quality Human Engineering has been a commitment by the prime contractor (South Carolina Research Authority) to include Human Engineering in all stages of the RAMP/RTIF project design and development. Although the individual tasks that the human engineer will do on the project are not uncommon, the manner that they are tied together, the organization position, and the assignment of responsibility/accountability are among the factors that make this Human Engineering function an effective system engineering team member. The benefits of this position are that Human Engineering will contribute to:

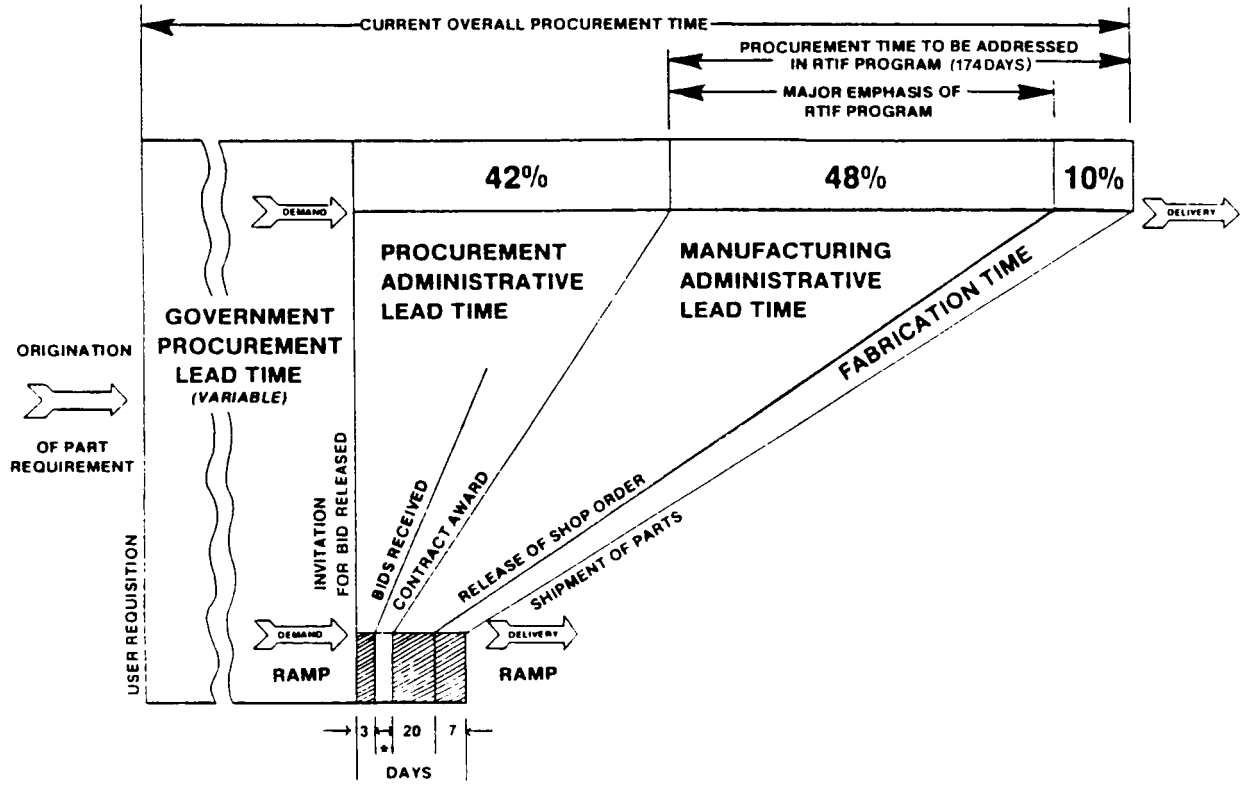


Figure 1 RAMP System Requirements Vs Present Average Supply System Responsiveness

1. Likelihood that the WHOLE PROJECT will succeed because people were properly accounted for in the design.
2. A practical approach to design where human needs are identified early and incorporated (integrated) into the systems requirements specs.
3. A realistic accounting for people issues in terms of accomplishing a balance between people needs and overall system needs.
3. Providing an effective symbiosis between people and the system that meets the performance, accuracy, and acceptability required.

This paper is to show the attributes of the RAMP/RTIF Human Engineering function that takes Human Engineering out of a support function (with little impact on overall system design) and makes it a system design team member (with appropriate impact on system design). Each of the following topics contributes to making Human Engineering a valuable team member. They are discussed here to encourage other projects to place Human Engineering into a similar role where they can greatly improve their contribution to a project.

Strength: SYSTEMS DESIGN CONCEPT

In the RAMP/RTIF project the human engineer must be concerned with all the various tasks, interfaces, and specific involvements that people will have with the RAMP. This means that the human engineer must work with each of the following functional areas in order to deal adequately with the people issues across the project:

- o Systems design, simulation, and integration
- o Hardware/software/process design
- o Safety/hazard analysis
- o Manpower and training
- o Test and evaluation
- o Logistics engineering
- o Operations and maintenance doctrinal development
- o Configuration control and management (software/hardware)
- o Program review/approval processes
- o Vendor evaluation and selection
- o Architectural facility design, development, review and approval
- o Engineering change review/implementation processes

As one looks at all these views that effect design and development, it is obvious that there must be synthesis of information across the technical areas in order to develop a workable design and development solution. The synthesis of information across these areas means that important aspects of a "Systems Design" process are being used. This concept is being emphasized here because the human engineer is a RAMP/RTIF team member in this process which gives great value to the Human Engineering function. By taking a system view in RAMP/RTIF, the people issues are dealt with wherever people have to perform a task, process information, or will physically come into contact with equipment and materials. As a result of the systems design approach, a consistent standard of interaction with people can be maintained across the system so that people will always know what to expect and be able to properly interact.

By taking a systems view in RAMP/RTIF, the people issues are dealt with wherever people have to perform a task, process information, or will physically come into contact with equipment and materials. As a result of the systems design approach a consistent standard of interaction with people can be maintained across the system so that people will always know what to expect and be able to properly interact.

Strength: STAGES OF SYSTEM DEVELOPMENT

The stages of system development presented below are not specifically identified in the same form in the project but the content is normally covered in the military system design process. The stages are mentioned here to clarify and emphasize the scope of the "System Design" work that the human engineer has to deal with. The human engineer will play a role in each of these stages throughout the project. Collectively the roles describe a methodology designed to maximize the effectiveness of the Human Engineering contributions to the project while minimizing the Human Engineering resources needed.

- o Conceptualization -- The initial identification and description of the people parameters must be made here.
- o Specification -- With the people issues identified in the conceptualization stage, conversion of those issues into integrated requirements will be more effective. Performance and acceptance test methods should be defined here. The tests must specify that the people who will be using the system will be a part of the test.
- o Design -- Human engineering will develop specific solutions to meet system specifications. The person works with the systems, software, hardware, and other engineers to generate design solutions that appropriately take care of and implements a design that meets needs.
- o Development -- As the design is being implemented the human engineer works with the engineering to formulate and ensure that design solutions are appropriate for the people who will be involved with the system. Implementation solutions will be user tested throughout the development stage to evaluate effectiveness of design and aid in making modifications in implementation where necessary.
- o Test -- Participate in evaluation of the system. Evaluate the effects of the system on people and the effectiveness of people to use the system. Show that system performance and acceptance meets requirements with people using the system.

Strength: HUMAN ENGINEERING APPROACH TO WORK

Support Function, No!

Traditionally people outside the Human Engineering discipline have viewed Human Engineering as a support function that is limited to concern for computer screen design, and controls and displays work. Human Engineering therefore has often been relegated to do specific, and very limited tasks that are considered to be within the domain of Human Engineering. The resulting piecemeal, out of context approach usually minimizes the likelihood that many/most of the important people issues across a project(i.e. uniform useability, system performance, acceptance) are addressed. An analogy would be like asking an architect to specify windows and doors for a building but not telling the architect for what type of building (critical information needed for the architect to determine the correct doors and windows to specify). The people who would use the building might find, for example, that they needed a garage door but none was included.

This approach usually results, at best, in a small amount of improvement for people who will come into contact with the system, but will generally result in forcing a poor match between the system and the people who must use it (See Figure 2, the square peg in the round hole syndrome). Since this is not an effective approach to deal with people issues, it is not being used.

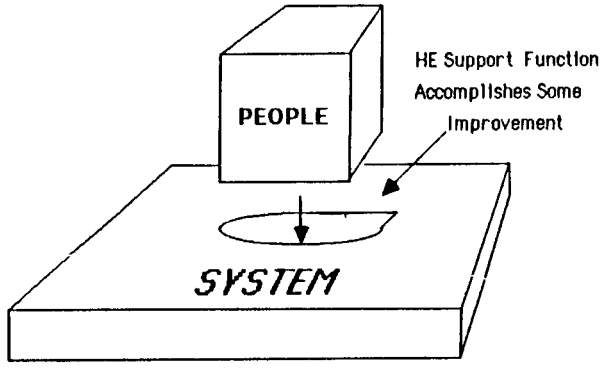


Figure 2 People Forced to Fit the Design of the "System" (The Square peg in the round hole approach)

Design System From Human Engineering Perspective, No!

A reverse approach to designing a system might be to place primary emphasis on designing it from the human engineering perspective. This would optimize the system for the people who will use it and/or come into contact with it (See Figure 3, Fitting the System to the People).. This approach might make the system easy to use, easy to understand, easy to learn, etc. but probably not meet overall system objectives, costs, performance, and other system requirements. Again this is due to taking a single, narrow, non-systems view of the project which decreases the likelihood of project succeed. Since this also is not an effective approach to deal with people issues, it is not being used.

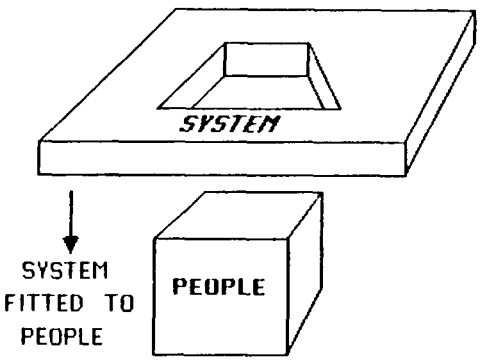


Figure 3 People Considerations are Prime Design Consideration. The System is Fitted to the People and Tasks (Overall system requirements may not be met because of lack of integration of people and system)

Integrated Engineering Team Member, Yes!

A very effective approach to integrating the system design with people's needs is to have Human Engineering play an active design team member role throughout the project. Through the application of the system design, development, and integration process to all Human Engineering issues, people's needs will be addressed and dealt with.

This ensures that all people interfaces with the system (hardware, software, other people, documentation, the environment) will be consistent, complete, and effective. This approach ensures that all Human Engineering considerations are considered in context with other engineering discipline considerations. Thus Human Engineering concerns are put into the perspective of being one of many concerns to be considered in the product being developed. The considerations are integrated into the design solution so that it provides the best product to meet the design criteria (See Figure 4, Integrating the System With the People). This integrated team design approach is being used successfully on the RAMP/RTIF project.

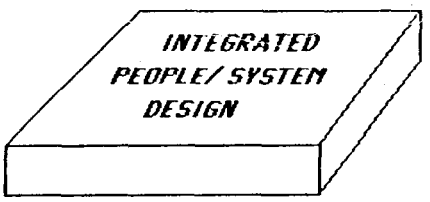


Figure 4 All Concerns Integrated Into a Total System Design (appropriately accounting for people issues)

This plan set the stage for Human Engineering to be a equal partner in the design of the system along with all the other disciplines. As the A and B specifications are being developed Human Engineering has a specific Human Engineering Section. But more importantly Human Engineering will work with all functions described in the spec to ensure that human engineering concerns are addressed. This means that Human Engineering inputs will be incorporated into those sections and not identified as specifically a "Human Engineering Concern" which is a concept that increases the effectiveness of the product.

Strength: HUMAN PERFORMANCE SYSTEM MODELING AND SIMULATION

To determine or predict the effectiveness of the RAMP/RTIF system human interfaces and task requirements, Human Engineering incorporates human tasks, human information processing, and performance data into the system design modeling and simulation. Examples of specific areas to be accounted for are:

1. Human information handling and processing requirements.
2. Tasks that must be performed in conjunction with the RAMP/RTIF hardware and software.
3. Manual vs automation task trade-off comparison for performance, quality, cost, and safety.
4. Maintenance performance as related to system availability.

By including this data in the system model and simulation, realistic system performance can be predicted.

Strength: ADMINISTRATION/MANAGEMENT

Responsibility

The Human Engineering function has primary organizational responsibility for the appropriate and timely decisions regarding integration of human engineering concerns throughout the RAMP/RTIF development, design, test, and integration process. Human Engineering participates in design, technical interchange meetings, program and critical design reviews, vendor evaluation and selection, and test and evaluation activities. The Human Engineering function must monitor, review, analyze, and respond

to human engineering related issues generated by all design development groups, and attend and participate in all significant design/development review processes. All Human Engineering tasks are scheduled to coincide with the master schedules, such as Preliminary, System, and Critical Design reviews and tasks that support those schedules.

Assigning the Human Engineering responsibility to the people that have the expertise ensures that the people concerns are appropriately addressed and resolved in the systems context.
Organizational Position is Important

The Human Engineering function is organizationally a part of the Systems engineering functional element in the RAMP/RTIF Systems Engineering Organization. This relationship will optimize the effective integration of Human Engineering concerns throughout the system development, integration, and test process.

Strength: DEFINE AND ALLOCATE SYSTEM FUNCTIONS

An operator versus machine functional task allocation describes those characteristics of the system which are strictly hardware and software functions and those that are functions of the operator. This important allocation function in RAMP/RTIF is performed by Human Engineering in conjunction with design engineers throughout the system specification process. An example of the process that is used to perform this task is:

1. Determine task requirements -- Once requirements have been identified for each task and cluster of tasks, the requirements are be analyzed in terms of the capabilities and limitations of people and/or machines that could perform the tasks. Humans and machines both have capabilities that they are very good at and limitations to what they can do. Performance, safety, acceptance, strength, information processing requirements, decision making capability, and cost of performance are some of the issues that can be used to determine approach.
2. Compare people and machine capabilities and weaknesses with requirements -- When there is a significant cost or performance differential between the two approaches, further evaluation (a trade study) is proposed. To aid the selection process, the system simulation model is used to look at the trade-offs between manual and automated task approaches.
3. Allocate people and/or machine resources to tasks -- Based on the analysis, the allocation of resources (people and/or machinery) to tasks are made and included in the B Specs. In cases where system functional requirements are outside the capabilities of the people who will use the system, an "expert system" AI package may be considered to perform the function.
4. Design system to efficiently and effectively use these resources.
5. Test against requirements.

These operator-machine interfaces will be described as an operational baseline to identify and standardize the functions to be performed or supported by the various segments of the RAMP/RTIF system. This analysis also serves to specify critical people, procedures, equipment, facility, environmental, and software performance/interface requirements for system operations, maintenance, and control functions. The choice of functions that are allocated to the system hardware and software and those allocated to the operator are assessed and validated.

HUMAN ENGINEERING PLAN DESCRIPTION

The scope of the Human Engineering activity for the RAMP/RTIF project was formalized early in the project through the Human Engineering Program Plan which is a part of the System Engineering Plan. The Human Engineering Plan identified the approach to be taken by Human Engineering, the responsibilities, and the working relationship with other functions in the project. This plan explicitly defined the issues that have been defined as strengths. The table of contents for the plan is listed here to show the scope of the definition:

- o Human Engineering Program Tasks
- o Establish RAMP/RTIF Human Engineering focus
  - o Drawing Approval
  - o Traceability and Access to Human Engineering System Data
- o Define and Allocate System Functions
- o Determine Potential Operator/Maintainer Processing Capabilities
- o Evaluate RAMP/RTIF System-Human Effectiveness
  - o Human Performance System Modeling & Simulation
  - o Decision/Action Flow Diagrams
  - o Timelines
  - o Workload Analysis
  - o Link Analysis
  - o Human Information Processing Analysis
- o Support the Selection/Design of RAMP/RTIF Equipment
- o Support the Selection/Design of RAMP/RTIF Software
- o Perform Human Engineering Task Analysis
- o Conduct Studies, Experiments and Laboratory Tests
- o Support RAMP PWA/SMP Workcell Environment and RTIF Facility Design
- o Support Test and Evaluation of Human Performance in the RAMP/RTIF
- o Site Specific Analysis
- o Human Engineering Deliverables

VALUE OF INTEGRATION APPROACH

By definition all COMPONENTS of a SYSTEM have impact on a SYSTEM. In the RAMP it has been realized that people issues can have a major impact on the SYSTEM. The degree to which the system meets the physical needs, mental needs, and expectations of those people can determine the level of system performance and acceptance that is

achieved. An important function of the system such as delivering the number and quality of parts within the cost requirements will depend as much on the people that use and run the system as to the technological aspects of the system. Therefore it is important to gain the best people performance in the system by ensuring that design solutions incorporate/integrate people needs.

CONCLUSION

In this NAVY program the Human Engineering function is an integral team member that contributes value to the systems engineering organization. Human Engineering works with people issues throughout the design and development of the system in such areas as: 1) Identifying people issues early in the project; 2) Accounting for those issues in the specification; 3) Appropriately resolved the issues in design; and 4) Testing the product to ensure it meets the needs and expectations of the people while meeting the overall systems requirements.

The Human Engineering approach being taken in this project facilitates the symbiosis between systems designers, engineers, and Human Engineering. This approach increases Human Engineering's effectiveness and value to a project because it becomes an accepted, contributing team member. It is an approach to doing Human Engineering that should be considered for most projects.

Acknowledgement -- This leading edge NAVY CIM contract is managed by the South Carolina Research Authority and design/development is being performed by a consortium of contractors (Battelle, Grumman, Ingersol, SRI International, and A.D. Little).

THE IDEAL MODELING METHODOLOGY: CAPABILITIES AND APPLICATIONS

Ken H. Evers
SofTech, Inc.
3100 Presidential Dr.
Fairborn, Ohio 45324-2039

Robert F. Bachert
Armstrong Aerospace Medical Research Laboratory
Wright-Patterson Air Force Base, Ohio 45433

The IDEAL (Integrated Design and Engineering Analysis Languages) modeling methodology has been formulated and applied over a five year period. It has proven to be a unique, integrated approach that utilizes a top-down, structured technique to define and document the system of interest; a knowledge engineering technique to collect and organize system descriptive information; a rapid prototyping technique to perform preliminary system performance analysis; and a sophisticated simulation technique to perform in-depth system performance analysis.

The use of modeling and simulation as a tool within the system analysis process is becoming an ever increasingly important capability. As part of this trend, the IDEAL methodology is making modeling and IDEAL provides these improvements in two ways. First, IDEAL permits the user to concentrate primarily on the nature of the system being anayzed rather than the simulation effort itself. Second, IDEAL permits more accurate models to be developed resulting in a better understanding and a more thorough anaysis of the modeled system. IDEAL is the unique anaysis capability it is because, above all else, it provides a communication and documentation vehicle. The design and analysis of any system (program scheduling and management; software and/or hardware systems; man-machine systems; large or small systems) requires, to one degree or another, an integrated team effort. For example, the development of a man-machine system could reasonably involve hardware and software engineers, human factors engineers, psychologists, design engineers, and program managers. IDEAL provides the features needed to integrate or blend these varied backgrounds into an effective working team. It is through this team effort that the best possible system is developed in the most efficient manner.

IDEAL has been effectively applied to a wide range of system types. Examples of these applications are a man-in-the-loop Surface-to-Air Missile (SAM) system, data base development programs, software development programs, aircraft radar systems, automated communications systems, and computer bus architectures.

This paper will discuss the capabilities of IDEAL, the types of system insights received while developing an IDEAL model, and the types of system analysis that have been performed using IDEAL.

INTRODUCTION

Simulation techniques are used to model or duplicate the behavior aspects of real-world or conceptual systems. These techniques are applicable to all phases of a systems life cycle. Simulation is a powerful method of analysis or evaluating a system without requiring the time and cost of building or modifying the actual system. The development of a simulation model requires a system description in the form of a functional model which is combined with timing and precedent relationships to form a dynamic computer simulation.

Current simulation techniques are highly dependent upon the experience and skill of those applying them. However, the techniques do not integrate all the personal critical to the success of the systems analysis and design. The lack of personnel integration results in the simulation expert having to define the system as he understands it, to define the analysis goals, to solve the problem as he understands it, and to exercise the simulation and interpret the results as he understands them. This usually results in a biased and limited view that is inadequately, verfied, validated, and communicated. Poor and imcomplete problem definition and system description results in an incomplete and inaccurate performance analysis of the system.

The IDEAL (Integrated Design, Engineering, and Analysis Languages) technique was developed to provide the features necessary to define or bound the problem, to develop a validated functional model of the system, to build a simulation model based on the functional model, and to communicate and discuss the analysis approach and results via the funtional model. It is through team efforts that the best possible system simulation is developed and the most complete system analysis performed.

SIMULATION PROCESS

The development of a simulation for analyzing a system requires a team effort. The larger and more complex the system is, the more difficult is the team effort. The team is normally composed of analysts, system experts, and project managers. The system analyst is the focal point for the entire effort. It is the analyst's responsibility to identify the purpose for the proposed simulation from the viewpoint of both the system experts and the project management.

Guided by the viewpoint and agreed upon purpose, the analyst, working as the team's focal point, steps through a systematic approach to identify and collect the information necessary for a system description and operation scenarios, to develop the system simulation from the system description, to exercise the simulation in a manner controlled by the operational scenario, and to recommend a proposed solution for satisfying the simulation purpose.

This general view of the simulation process is illustrated in more detail by the model in figure 1. The simulation process, upon which IDEAL is based, is divided into six major activities. The Describe System activity uses the system description portion of IDEAL to construct a validated static system model. The Generate Performance Data Base activity uses the static system model as a guide to identify and collect the appropriate dynamic characteristics of the system. The Determine Simulation Objectives acitivty identifies the objectives, or goals, of building and exercising the simulation and establishes the criteria which will be used to evaluate system performance. The Construct System Simulation Model activity develops a dynamic model of simulation of the system. This activity is accomplished by translating the system information described in the static model and the dynamic characteristics into the simulation using the structures defined within the IDEAL simulation technique. This model defines the way in which the elements of the system interact to cause changes in the state of the system over time. The Exercise Model activity verifies and validates the dynamic model. Verification determines that

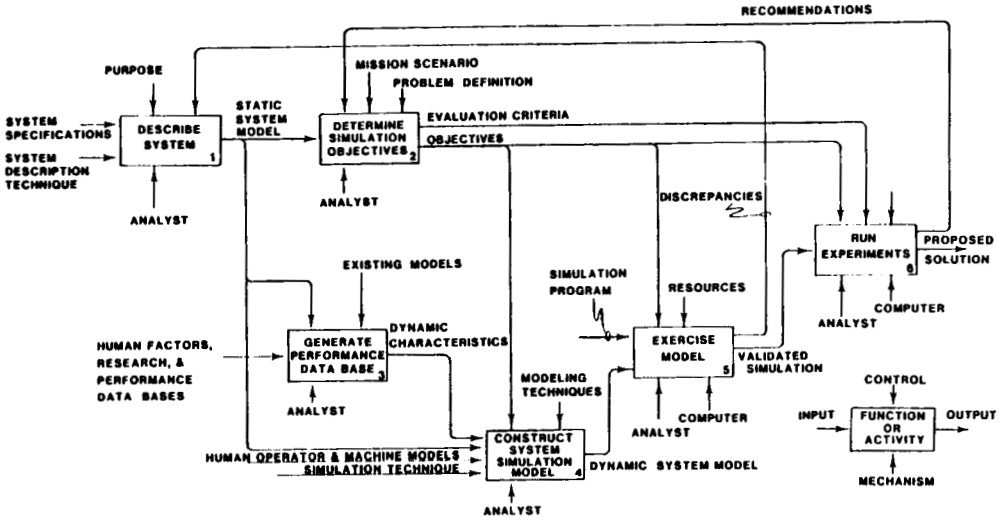


Figure 1. The Simulation Process

the model executes on the computer as the modeler intended. Validation determines that the model is a reasonable representation of the system. The Run Experiments activity exercises the model on the computer and interprets the results. The results of this activity are either recommendations for additional analysis or a proposed solution to the problem definition.

IDEAL is a step-wise technique in which each step maps into either the Describe System, the Generate Performance Data Base, or the Construct Systme Simulation Model activities. IDEAL uses a hierarchical, functional decomposition approach to describes the system in terms of a static system model. The approach provides the means for bounding the system of interest in terms of identifying what overall function the system is to perform, inputs to the system, outputs produced by the system, what controls and relationships are used to transform input into output, and what equipment, software, and humans are required to perform the transformation. Once the system has been bounded, the decomposition activity of IDEAL provides the structures necessary to gain a gradual, controlled graphic representation and understanding of the system (figure 2).

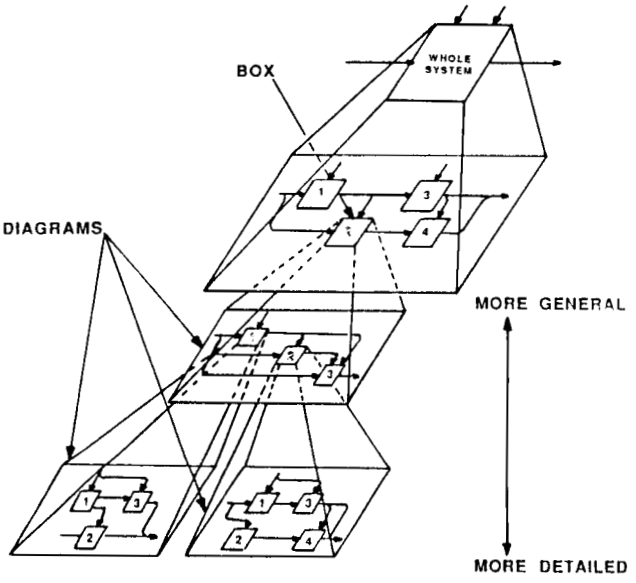


Figure 2. Functional Decomposition Concept

IDEAL uses a data base approach in which each function identified in the functional decomposition has an associated template for storing the dynamics data needed to provide a

complete operational description of that function. To develop the final simulation, IDEAL uses a network simulation approach which links the functions together and integrates dynamic data to form the operational representation of the system.

The one feature which provides IDEAL with its power as an analysis tool is its utilization of a team effort. IDEAL accomplishes this integration through its knowledge engineering capability. The process of developing the functional model and the sytem performance characteristics is accomplished through an interviewing procedure during which the analyst collects information from system experts. The analyst then integrates the information and represents it in terms of the functional model and the system performance description. This integrated process helps to identify discrepencies among information sources as well as missing information about the system. This integrated information is then reviewed by the system experts in order to verify the analyst's understanding of the system, to add additional information, or to clarify conflicts among the experts.

This knowledge engineering approach is an interactive process which requires communication among all members of the team which and causes IDEAL to be both a management tool and a technical tool (figure 3). The management members of the team are provided the vehicle through which they can understand the simulation being developed as well as being able to plan, organize, control, and coordinate the resources required for the effort. The technical members of the team are provided the vehicle to have their information and analysis desires integrated into the simulation and to monitor and understand the simulation as it is being developed. The ultimate effects of developing a model using IDEAL are that the model will be understood and accepted by the team, will be a fully documented model, will be much easier to maintain and engance, and can be used as a training tool with respect to the operation of the system.

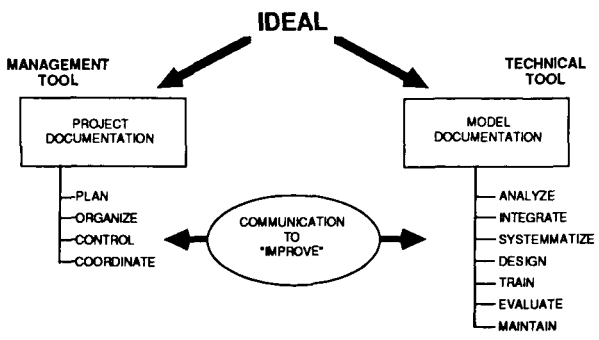


Figure 3. Project Integration Using IDEAL

IDEAL is a general purpose system analysis/design tool. It is applicable to any type of system and is beneficial even if only portions of the methodology are applied. There are natural breakpoints after the functional description and the Performance Data Base (PDB). At each breakpoint, a significant amount of new information will be available about the system. This may be enough information to answer the specified question. As more system insight is desired, the additional steps can be exercised or the current steps and be iterated.

The methodology has been applied to a number of system types. Examples of these applications have included man-machine systems, automated message processing systems, data base design, manufacturing analysis, software design, and the integratin of existing simulations.

MAN-MACHINE SYSTEMS

IDEAL has been used to develop a simulation of a Surface-to-Air Missile (SAM) system [1]. This effort required the integration of knowledge from three experts types. From the system operators, an expert system data base was developed by identifying tasks each operator performed, what information they used and how they used it, and how well they were able to perform their performance. From the hardware engineers, an understanding of the hardware capabilities was established. From other modeling and simulation experts, submodels of the same system were identified, evaluated, and assessed for integration into the IDEAL simulation.

This simulation was used to analyze the ability of the SAM crew to intercept threat aircraft that were operating under a variety of flight profiles and using a combination of countermeasure techniques.

AUTOMATED MESSAGE PROCESSING SYSTEMS

IDEAL has been used on a number of message processing systems. Two of the most in depth analysis were performed on the AE-9B High Speed Ring Bus (HSRB) Protocol [2] and the Central Processing Communication (CPCOM) system for the Precision Location and Strike System [3]. In both cases, the performance information was obtained from appropriate documents for the preliminary model. The system designers and the software developers were then incorporated into the IDEAL process to expand system understanding and perform model validation.

The results of these analysis provided very useful. For CPCOM, it was found that major interface problems existed between the communications system and subcomponents that was linked to CPCOM. Also identified were some minor process timing problems that when changed caused substantial improvements in the processing capability.

For the HSRB, most of the design parameters were verfified but one design parameter was determined to be less than optimal and significant recommendations were made as to the message from that would be best transmitted on the bus.

DATA BASE DESIGN

IDEAL was applied to the specification of a large data base design and for proving that the design would operate within the specified requirements [4]. This model was used to define, document, and communicate the original design and continue to be used to test additions before they are implemented.

MANUFACTURING

IDEAL has been applied to a variety of manufacturing capabilities. One model involved the analysis of an existing manufacturing facility and was aimed at identifying what processes were being accomplished, what information was common among the processes, what equipment and personnel were required to perform the functions, and how well each operator was being performed [5]. The goal of this effort was to develop a plan for upgrading the facility.

A model was developed to analyze the process flow of an oil refinery. This model development involved the analysis of the flow rates, storage capabilities, and process monitoring requirements. The goal of the model was to recommend an approach for optimizing the process monitoring capabilities of the refinery.

SOFTWARE DESIGN

IDEAL has been used on numerous program to either aid in the design of large, complex software systems or to analyze an existing system. One significant model involved the model ing and analysis of the software for the radar mode interleaving process of the B1-B aircraft [6]. The results of this simulation were used to identify a number of operational restrictions as well as some design errors. The results of the analysis were that problems were identified during the design stages and corrections were made without any major impact on the development schedule.

CONCLUSIONS

IDEAL is a proven, general purpose methodology for modeling a wide variety of system types. IDEAL's power lies in its capability to utilize knowledge engineering techniques to collect, integrate, and verify system information from a team of people. IDEAL provides the capabilities to build a system simulation in a top-down, structured manner, in a notation that communicates and documents the system, and in a form

executable on a computer. The graphical notation of the technique aids in the debugging, testing, modifications, and communication of the system. It allows the various system subfunctions to be modeled at different levels of detail in order to meet the needs of the problem statement and has the framework necessary to effectively integrate existing subsystem models.

REFERENCES

1. SofTech, Inc., "SAINT Performance Assessment Model of a SAM System (SPAMSS); Analyst Manual", August 1984.

2. Cooper, W.M., "Using SAINT in Performance Analysis of Complex Hardware/Software Systems", Proceedings of the IEEE 1985 National Aerospace and Electronics Conference (NAECON), Dayton, Ohio.

3. SofTech, Inc., "High Speed Ring Bus (HSRB) Protocol Analysis", June 1987.

4. SofTech, Inc., This information is contained in a SofTech proprietary document.

5. SofTech, Inc., This information is contained in a SofTech proprietary document.

6. SofTech, Inc., This information is contained in a SofTech proprietary document.

7. Evers, K.H., Bachert, R.F., "SADT: An Effective Tool For Knowledge Acquisition", Human Factors in Organizational Design and Management - II, August, 1986, Vancouver, B.C., Canada.

8. Bachert, R.F., Evers, K.H., Hoyland, C.M., & Rolek, E.P., "IDEF/SAINT SAM Simulation: Hardware/Human submodels", Proceedings of the IEEE 1983 National Aerospace and Electronics Conference (NAECON), Dayton, Ohio.

9. Bachert, R.F., Evers, K.H., Santucci, P.R., "SADT/SAINT: Large Scale Analysis Simulation Methodology", Proceedings of the 1981 Winter Simulation Conference.

The Rapid Intelligent Prototyping Laboratory: A Direct Manipulation Tool for Prototyping and Evaluating User-System Interfaces

(Paper not provided by publication date)

COOPERATIVE ANALYSIS EXPERT SITUATION ASSESSMENT RESEARCH

Michael G. McCown, 1Lt, USAF
Rome Air Development Center

For the past few decades, Rome Air Development Center has been conducting research in the area termed Artificial Intelligence. When the recent advances in hardware technology made many Artificial Intelligence techniques practical, the Intelligence and Reconnaissance Directorate of RADC initiated an applications program entitled Knowledge Based Intelligence Systems (KBIS). The goal of the program is the development of a generic Intelligent Analyst System, an open machine with the framework for intelligence analysis, natural language processing, and man-machine interface techniques, only needing the specific problem domain knowledge to be operationally useful.[1]

On the path towards such an architecture, RADC first implemented a number of domain specific expert systems. The evolutionary design of these individual expert systems has rapidly become more modular and distributed in itself (Figure 1). The next step of the generic design requires that the modular and distributed natures of these systems be carried to their necessary conclusion, that of loosely coupled, distributed, cooperating knowledge based system architecture. The effort to explore this step is entitled Cooperative Analysis Expert Situation Assessment Research (CAESAR).

In order to understand the evolution of the current architecture used in the KBIS program, it is necessary to understand a few fundamentals of intelligence analysis and the intelligence community. Much day to day intelligence analysis is done by assessing the status of indicators. Indicators are sets of actions that an enemy would be expected to take in preparation for an aggressive act. An indicator list is a list of activities that an enemy might be expected to engage in if they intended to initiate hostilities. The activities are logical/plausible moves or acts based on: a) Operational procedures, b) Observed activities during past conflicts and crises, and c) Results of intelligence assessments of enemy strategic offensive military doctrine.

The monitoring of indicators pertaining to strategic warning is an extremely time-critical process since this warning impacts the degree of readiness of our strategic offensive and defensive forces. Given the increasing threat from a number of sources, the task of providing an accurate and timely assessment of enemy intent is limited by the availability of qualified intelligence analysts and their ability to process increasingly large amounts of information in a crisis. While some tools exist to assist the intelligence analysis function (the World Wide Indicator Management System, for example), further research is needed to increase the accuracy and timeliness of strategic warning.

Today's automated support for the intelligence analyst is limited primarily to Intelligence Data Handling Systems, which often times can be as difficult to use as the paper pushing methods they replace. Yet automated data handling support is dictated by the massive volumes of traffic being passed through all Indications and Warning Centers. In addition to the data flow and management problem, the analysis and fusion of that data is becoming increasingly complex and subtle as foreign technology and doctrine improves. Today's intelligence analyst requires phenomenal memory, computer skills, and analytical capability in order to be effective. To relieve some of the burden, automated support must take the role of an informed colleague to the analyst, extending his memory, performing data retrievals, and aiding him in analysis. In addition, this automated system must require little to no time and capability to learn and to use. The KBIS program is designed to help alleviate this problem.

Recent efforts have concentrated on the automation of indicator assessment, particularly for space situation assessment.[2] These have included sophisticated data base managers as well as single domain expert systems designed to maintain indicator lists. However, as the Intelligence Community is currently structured, different centers have control over different data sources, and each exploit these data sources in slightly different ways. This necessitates the sharing of relevant information between centers and creates two important problems. It is difficult to tell exactly what information may be relevant so it is often necessary to overcompensate by sending too much information,

KBIS ARCHITECTURE

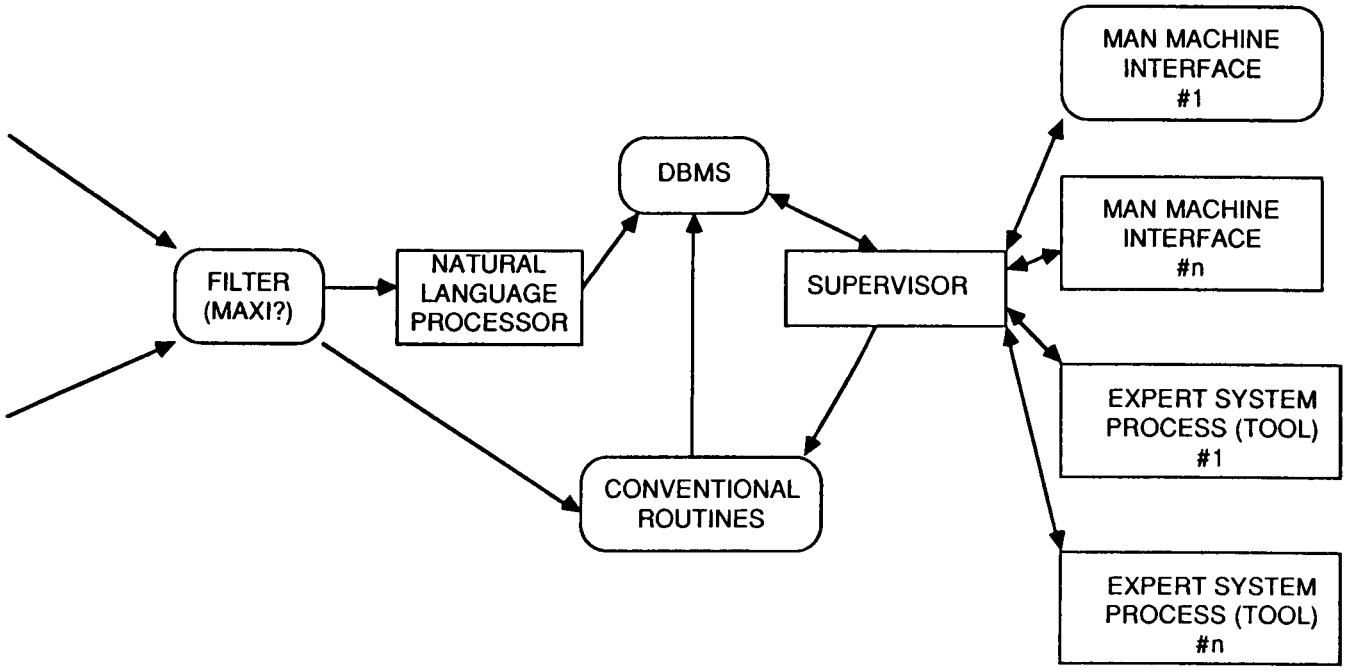


FIGURE 1

and it is this overcompensation which causes such a data flow problem.

It is the hypothesis of the CAESAR extension of the KBIS program that increases in accuracy and timeliness of assessments can be made by developing a cooperative framework for a number of single domain expert systems, and this may allow for a possible reduction in data flow over communication lines. The architecture will make use of a common information exchange language that will enable the domain expert systems to request and receive information from other KBS's on a shared communication network, such as AUTODIN. As a result, this system will provide the intelligence analyst with an expanded yet focused interpretation of strategic indicators developed via cross-domain synergism, as well as relieve the data distribution and flow problems which currently exist.

As can be seen in Figure 1, the architecture has levels of components: the message processing, data base, and active agent. This is the architecture currently under implementation in the KBIS program. The data base level is separated from the other two primarily so as to allow integration of advances in the other two areas with existing data bases. This is crucial, as current knowledge based systems have been shown to be most successful when dealing with a deep, narrowly scoped domain. As such, they need to be incrementally added to an existing system in order to be operationally practical. In addition, the intelligence analysis process itself is often broken down in this same manner into subanalysis tasks so as to be handled by individual analysts, and the assessments from these subtasks are combined at a higher hierarchical level. This real world process then serves as our model for the cooperative expert system design.

The active agent concept is an important one to the architecture. Active agent components are so called as it should not matter to the other two components (message processing and data base) whether the communications from that level are from a human being or a knowledge based (expert) system. Since both theoretically will request and pass similar information, that facet should be transparent to the architecture. This concept allows for multiple types of man-machine interfaces, from existing (dumb terminal) types to future sophisticated (smart terminal, intelligent MMI) types. Also, this allows for multiple sub-domain expert systems working on the same problem in a blackboard manner, shown to be a powerful method for problem solving. The key here is that these active agents are themselves independent, and cooperation takes place only at the level of using the same communication language.[3] This communication language within the architecture has a highly formatted syntax, but is free as to content of fields.

The message processing level is a complex and critical aspect to this architecture. Since a large number of messages are currently passed between intelligence centers in natural language, the message processing level requires a sophisticated natural language processing capability in order to make the other two levels effective. It is primarily at this level that changes will need to be made in order to have a viable distributed network architecture. However, changes at this level are strongly linked to the methods used at the active agent level. It is the hope of this work that the internal communication language currently embodied in the Supervisor module which handles the control and information transfer between the knowledge based systems, man-machine interfaces, and DBMS can serve as the baseline for the more general case.

The general case, however, is quite broad. From raw data, to multiple levels of fusion and assessment, to hypothesis and warning generation, the types of information passed vary widely. In addition to specific information requests, broadcasting of desired goals and achieved states, and passing of reasoning chains, the general case must also include control of information transfer, including network protocols, multi-level security, communication gateways (network to network links), message routing, and routing header formats.

Since the capacity of this internal communication language is crucial to the practical success of this architecture, it bears futher discussion. While numerous attempts have been made in the past to rigidly format messages so as to allow for automated data processing ease (JINTACCS, for example), these attempts always have met with limited success. Limited, in the sense that all message formats allow for at least one free text area to explain and present information which doesn't fit well in the rigid fields of the message type.

This effort makes two assumptions as to the feasibility of its internal communication language. One, that the intelligence domain has a finite number of information sources which can be categorized into types (IMINT, HUMINT, SIGINT, etc.), and these types can be rigidly mapped into a rigid syntax grammar (or vector with fixed fields). Two, these vectors can be directly converted and matched to the range of knowledge representations currently employed in knowledge based systems. Each field in a given message type is itself of fixed type but free content. Should the content of a field not match a particular bit of knowledge in a KBS, the KBS can recognize that the content is outside its current scope and either initiate some learning algorithm based upon the known type and field, or flag the input for a knowledge base maintenance function.

The scope of the CAESAR effort is the development of an experimental version of a technological capability to perform distributed time critical event assessment of foreign military activity. The experimental version will demonstrate cooperating knowledge-based systems technology for indications and warning that performs both domain and cross-domain indicator analysis through bi-directional communications. The effort will require the design and implementation of experimental software for handling communications between separate knowledge-based systems. The effort will also require the development of a structured evaluation scheme (to include test procedures, evaluation criteria, etc.) to assess the system effectiveness of the experimental version.

Much of the past research in cooperating expert systems has dealt with a tightly coupled cooperation, claiming that better performance can be achieved. While blackboard concepts are still popular, the tendency has been to sacrifice true modularity for performance in such systems.[4] [5] In addition, much of the current literature claims that in order to be truly effective, KBSs must become much more tightly integrated with data bases, and many expert system tools are evolving in this direction.[6] The author supports this evolution, and agrees that such architectures enhance performance and effectiveness. However, practical considerations have forced the evolution of a loosely coupled, distributed Intelligence Community, and an architecture which models itself on this has the greatest chance of simplified integration and success. The applicability of such an architecture, though, would seem to extend beyond the needs of the Intelligence Community to any environment where a loosely coupled architecture is advantageous, such as the space station. There is a supporting body of research supporting this concept as well.[7]

The KBIS program has developed a number of successful stand alone systems which tackle real world problems, most notably for space situation assessment, launch prediction, and space object identification. The success of these systems and the lessons learned as to individual architectures have provided the baseline for the CAESAR effort. As it is necessary for the persons using these systems to share information, it is the obvious next step in the evolutionary development of an overall architecture for these systems. While success seems promising based upon the current research, it cannot be overly stressed that the real world is always more complex than the most ingenuous laboratory environment. Until enough individual systems are in place in operational settings to make a test of the CAESAR architecture valid, the success of the program can only be measured against other academic and laboratory research. This evaluation will take place in the 1990 time frame.

---

[1] Final Technical Report, "The Intelligent Analyst System", RADC F30602-83-C-0105, June 1985.

[2] Final Technical Report, "Space Foreign Launch Assessment", RADC-TR-86-157 (S), September 1986.

[3] Agha, G., ACTORS, A Model of Concurrent Computation in Distributed Systems, The MIT Press, 1986.

[4] Nii, H., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures", The AI Magazine, Summer 1986.

[5] Hewitt, C. and Lieberman, H., "Design Issues in Parallel Architectures for Artificial Intelligence", IEEE COMPCON 84, February 1984.

[6] Howard, H., et. al., "Knowledge-Based Database Management for Expert Systems", SIGART Newletter, No. 92, April 1985.

[7] Genesereth, M., et. al, "Cooperation Without Communication", AAAI-86 Proceedings, Vol. 1, 1986.

N88-17229

PERFORMANCE ANALYSIS OF PARALLEL BRANCH AND BOUND SEARCH WITH THE HYPERCUBE ARCHITECTURE

Captain Richard T. Mraz, USAF
1MSCS/DOXC
Johnson Space Center, Texas 77058

ABSTRACT

With the availability of commercial parallel computers, researchers are examining new classes of problems for benefits from parallel processing. This paper presents results of an investigation of the class of search intensive problems. The specific problem discussed in this paper is the 'Least-Cost' Branch and Bound search method of deadline job scheduling. The object-oriented design methodology was used to map the problem into a parallel solution. While the initial design was good for a prototype, the best performance resulted from fine-tuning the algorithm for a specific computer. The experiments analyze the computation time, the speed up over a VAX 11/785, and the load balance of the problem when using a loosely coupled multiprocessor system based on the hypercube architecture.

INTRODUCTION

Within the past decade, parallel computer architectures have been a subject of significant research efforts. Integrated circuit technology, high speed communications, along with hardware and software designs have made parallel computers much easier to build and much more reliable (6,10,14,15). Parallel processing has also proven to be an effective solution to certain classes of problems. Probably the most notable class is array or vector problems that run order-of-magnitudes faster on parallel architectures such as the Cray. Because of the recent proliferation of parallel computers, researchers are investigating other classes of problems for potential benefits from parallel architectures. Search intensive problems are one such class. Figure 1 illustrates, research in the area of parallel computers has been highly successful in producing several general purpose hardware designs. Clearly, this list indicates the availability of parallel processing system hardware; however, the application and software support systems are not as prevalent. Stankovic points out that "much of the distributed system software research is experimental work" (17: 17). He further emphasizes that "work needs to be done in the evaluation of these systems in terms of the problem domains they are suited for and their performance" (17: 17).

Yet, another area of interest in parallel processing is the mapping of a problem to a parallel solution. Probably, the largest problem researchers face today in parallel computer systems is the inability of humans to decipher the inherent parallelism of problems that are traditionally solved using sequential algorithms. Patton identified a possible cause of this human shortcoming when he said, "While the world around us works in parallel, our perception of it has been filtered through 300 years of sequential mathematics, 50 years of the theory of algorithms, and 28 years of Fortran programming" (11: 34). Basically, humans have not trained their thought processes to accommodate the concepts of solving problems in parallel. Because of this, without new parallel computing algorithms, parallel software development tools, and performance measuring techniques, parallel computing may never be fully exploited.

Company	Product
Alliant Computer Systems Corporati on	FX/Series
Bolt, Beranek, and Newman	Butterfly
Control Data Corporation	Cyber 205 Series 600
Cray Research Inc.	Cray-2 and X-MP
Digital Equipment Corporation	VAX 11/782 and 784
ELXSI (a subsidiary of Trilogy Inc .)	System 6400
Encore Computer Corporation	Multimax
ETA Systems Inc. (a spin-off of Control Data Co rporation)	GF-10
Floating Point Systems Inc.	T Series
Goodyear Aerospace Corporation	MMP
IBM Corporation	RP3
Intel Scientific Computers	iPSC
Schlumberger Ltd.	FAIM-1
Sequent Computer Systems Inc.	Balance 21000
Thinking Machines Corporation	Connection Machine

Figure 1: U.S. Companies Offering or Building Parallel Processors (6:753)

Problem

Because of the proliferation of parallel computers and because a large class of problems that may benefit from parallel processing are search intensive, this research investigated the actual performance of a class of search problems on the Intel iPSC Hypercube computer.

Two examples of the need for this research into parallel search algorithms and performance evaluations are elements of the Strategic Defense Initiative (SDI) and the Pilot's Associate (PA). The SDI Organization is investigating defensive weapon systems and battle management systems for a strategic defense. While researchers for the PA program are investigating flight domain systems that provide expert advice in critical mission functions, such as aircraft systems monitoring, situation assessment, mission planning, and tactics advising (5,12:102,16). The general approach to solve some of the battle management and PA problems uses traditional operations research (OR) and artificial intelligence

(AI) programming techniques. These techniques are based on a systematic search of the solution space of the problem. Hence, this research focuses on parallel search methods. And without loosing generality, the specific technique is parallel branch and bound. For example, the SDI battle management system must resolve the resource allocation of sensor and tracking satellites to defensive weapon systems (16: 4-5). Answers to such a problem involves a complex solution space with exponential computation time to find the optimal solution. Researchers plan to reduce the run time complexity using parallel computers. The ultimate goal is to find the proper combination of parallel computer architecture and parallel algorithm such that results can be calculated in "real-time", where "real-time" is that time interval in which an answer must be delivered(10:8).

While a general search definition is useful during the parallel design phases of research, a specific problem must be solved for an actual performance evaluation. To this end, the specific class of 'Least-Cost' Branch and Bound search is used, where the basis of the hypercube performance evaluation is the Deadline Job Scheduling (DJS) problem. In a DJS problem, a set of jobs or tasks are defined by a 3-tuple (p_i,d_i,t_i), where

p_i = Penalty for not scheduling job i
d_i = Deadline by which job i must be completed
t_i = Time to run the job i

The goal is to find the largest subset of jobs that can run by their deadline while minimizing the total penalty incurred. This search uses both a ranking function to identify potentially good solution paths and two bound functions to eliminate needless searching in parts of the solution space. The DJS problem is characterized by exponential time complexity to find the optimal subset of jobs (worse case).

The goals of this research can now be summarized as follows,

1- Explore a design methodology to map a problem into a parallel computer.

Because of the difficulties of mapping a problem to a parallel computer, a formal design approach is needed to help the programmer identify the parallel activity within a problem. Since the development and proof of a new design methodology is beyond the scope of this research, only traditional design approaches will be examined.

2- Measure the performance of parallel branch and bound search on a parallel computer.

Since some researchers with search intensive problems, such as the SDIO and Pilot's Associate, have requirements for 'real-time' processing, experiments must be run to examine the possibilities for speed up. The results of a parallel branch and bound test can be used as a benchmark for further research as well.

3- Evaluate the hypercube as a suitable architecture for search algorithms.

In conjunction with the development of a good parallel algorithm, the speed up of a problem is also a function of the parallel computer architecture. Therefore, as Stankovic pointed out, the parallel architecture must be evaluated to identify their suitable problem domains.

Parallel Processing Issues

Two fundamental issues of parallel processing form a basic set of constraints for parallel problem solving. Simply stated, the first concept of maximum parallelism places a restriction on a parallel solution. This constraint may take several forms. First, the problem may inherently have limitations and dependencies that cannot be overcome. Second, a poor algorithm may inhibit parallel activity. Finally, parallel computer architectures have been targeted to solve specific classes of problems.

The second parallel processing issue deals with the mapping of a problem into a parallel solution. For humans, thinking in parallel does not come naturally. Therefore, a design methodology is needed to describe a problem such that parallel activity can be identified.

Overview of the Paper

In the introduction, a look at the need for this research, the definition of the problem, and the description of two parallel processing issues identified fundamental concepts used throughout this research. In the next section, a description of the hypercube computer presents the parallel environment for this research. Then, the definition of search and the parallel branch and bound design is reviewed. Following the design, the experimental results and conclusions of this research complete the paper.

HYPERCUBE ARCHITECTURE

The parallel environment for this research is the Intel iPSC Hypercube computer. Initial research on the hypercube, known as the Cosmic Cube, was conducted by Professor Charles L. Seitz at the California Institute of Technology (8,14). The basis of the hypercube computer can be described by the process model of computation (14). Simply stated, the process model describes the interaction of processes using message passing instead of shared variables(14:22). Using such a model, "a programmer can formulate problems in terms of processes and 'virtual' communication channels between processes" (14:23). The Intel iPSC hypercube used in this research adheres to the process model of computation in two ways. First, programmers define and encapsulate processes on any iPSC node. In fact, several processes can be placed on each iPSC node. Second, the iPSC operating system provides a set of message passing primitives for interprocess communication. The processor interconnection strategy that provides good message passing properties to support this model of computation is called the binary-n-cube or hypercube (see Figure 2) (14,15,18). As described by Wu, the binary n-cube is a network of 2^n processors where each node has n neighbors (18:239). The number n also describes the dimension of the cube. For example, a 3-dimension cube has 2^3 nodes and each node has 3 neighbors. Node identification consists of a binary number of length n (see Figure 2).

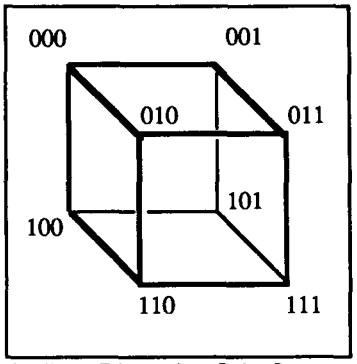


Figure 2: Three-Dimension Cube Structure, with vertices labeled from 0 to 7 in binary (15:66).

In addition to the message passing architecture, a programmer can configure the hypercube into several logical structures, such as ring, tree, grids, torus, and bus using specific message passing schemes (8,14). Using these structures, efficient nearest neighbor communications is maintained and the structure of the parallel solution can be designed to match the structure of the problem.

FUNDAMENTALS OF SEARCH

Search is a basic Operations Research (OR) and Artificial Intelligence (AI) programming technique. Such a strategy is used when problems cannot be solved using direct methods (i.e. formulas, algorithms, etc.) (13:55). Several specific search strategies have been developed (2,7,13). Each strategy varies the way the solution space of the problem is examined for answers. Sometimes the entire solution space is blindly searched for an answer. While other search techniques use heuristics or rules to guide through the solution space. The solution space for a search is typically represented using a tree organization (7:325). Horowitz and Sahni describe the search tree as follows (7:325-329). The root of the tree represents the *initial state* of the problem (see Figure 3). Each nonterminal node in the tree represents a *problem state* in the search. The *state space* of a search is defined as the collection of all paths from the root node to any node in the tree.

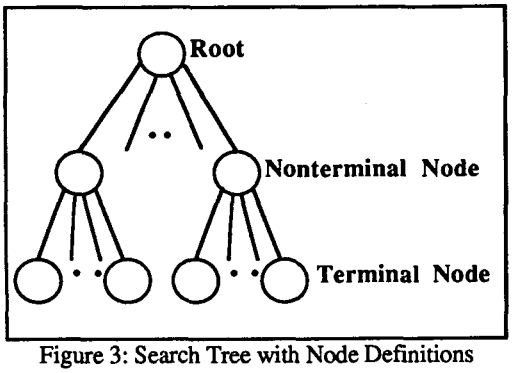


Figure 3: Search Tree with Node Definitions

Even though trees are used to represent the solution space of a search, the tree is usually not stored explicitly in the computer. Because search problems have the additional overhead of combinatorial explosion due to the branching factor or the depth of the tree, only portions of the tree needed to solve the problem are kept in storage. For this research, branch and bound, the general form of a state space search, is used. By manipulating two functions, a ranking function and a bound function, branch and bound can be used to model 'blind' as well as intelligent search. While 'blind' search techniques, such as Depth-First and Breadth-First search, do not use knowledge of the problem domain to control the search process, other search methods, called intelligent search, try to narrow the search space, shorten the search time, and reduce the storage needed by applying knowledge of the problem domain to control the search. The following actions are used to meet the three goals of 'intelligent' search (2:59),

1- Decide which node to expand next.
2- Select the most promising successors when expanding a node.
3- Eliminating or pruning the search tree.

To represent a node in the branch and bound search space, a solution vector $(x_1,x_2,\ldots,x_n)$, is used (7:323). Each x_i is constrained by explicit and implicit constraints. The explicit constraints define the range of values that each x_i can be assigned. For example, the solution vector for a 4-Task Deadline Job Scheduling problem is (x_1,x_2,x_3,x_4). The explicit constraints for this problem are simply $x_i \in \{0,1\}$, where 1 denotes that task i is included in the schedule and 0 denotes that job i is not included in the schedule. For instance, a valid solution vector for the 4-Task problem would be (1,1,1,•). This vector represents the search state where jobs 1, 2, and 3 have been scheduled and job 4 has not been scheduled. To help the reader in understanding the DJS constraints,, the following job set will be used in examples throughout this section (16:384),

Job	p_i	d_i	t_i
1	5	1	1
2	10	3	2
3	6	2	1
4	3	1	1

Using the definition of explicit constraints, the solution space of the example job set is depicted in Figure 4. The grey node identifies the example solution vector (1,1,1,•). The second set of constraints, implicit constraints, define relationships among the various x_i's. The nodes in the solution space that meet both the explicit and the implicit constraints define answer nodes. The first implicit constraint for the DJS problem is called the Deadline/Total Time Bound. This constraints requires a job to be scheduled such that the total run time for all jobs included in the schedule does not exceed the maximum deadline. Referring to the example 4-Job problem above, the solution vector (1,1,•,•) passes the Deadline/Total Time implicit constraint because the maximum deadline of jobs 1 and 2 is 3 and the total run time of jobs 1 and 2 equals 3. However, the solution vector (1,1,1,•) does not pass the Deadline/Total Time Bound because the maximum deadline of jobs 1, 2, and 3 equals 3 and the total run time of those same jobs equals 4.

The second implicit constraint, Cost/Upper Bound, for the DJS problem is based on the cost of the node and a global upper bound. The cost function is calculated in two steps (16:386). First, find **m** where,

$$\mathbf{m} = \max\{i \mid i \in S_X\}$$

S_X = the subset of jobs examined a node X

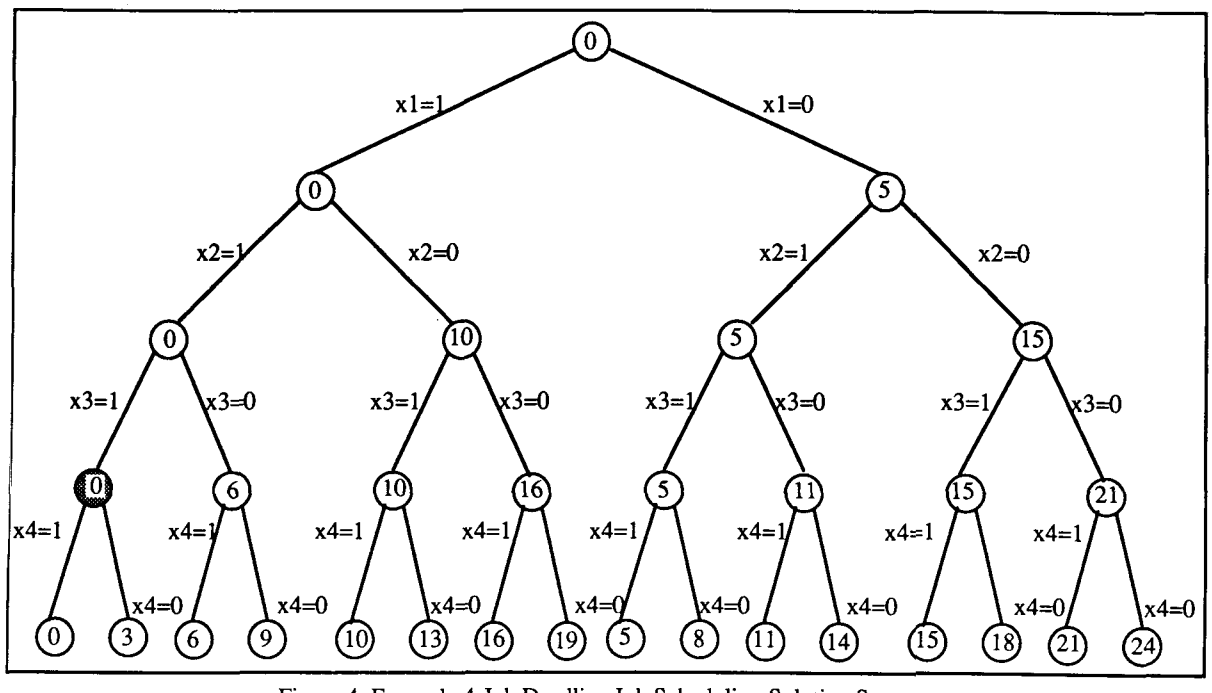


Figure 4: Example 4-Job Deadline Job Scheduling Solution Space

Next, compute the cost of node X using the following equation,

$$c'(X) = \sum_{\substack{i \le m \\ i \notin J}} p_i$$

where J = the set of jobs included in the schedule at node X.

The cost of a node translates to the total penalty incurred of all jobs that have not been scheduled so far. The cost of each node of the example job set is shown inside the circles of Figure 4. For example, the cost of solution vector (1,0,•,•) equal 10 because,

$S_X = \{1,2\}$

$m = \max\{i \mid i \in S_X\} = 2$

$J = \{1\}$

$$c'(\ (1,0,\bullet,\bullet)\ ) = \sum_{\substack{i \le 2 \\ i \in \{2,3,4\}}} p_i = 10$$

The second part of the Cost/Upper Bound constraint involves calculating the upper bound of node X using the following function,

$$U(x) = \sum_{i \notin J} p_i$$

The value of the upper bound identifies the maximum cost solution node inthe subtree rooted at node X. For example, vector(1,0,•,•) of the tree in Figure 4 has an upper bound of 14 since the cost of solution node (0,1,0,0) equal 14 and solution node (0,1,0,0) is the highest cost node in the subtree. During the seach, the lowest upper bound is maintained as a global bound. The gobal upper bound is defined by the following function,

global upper bound = min{U(x),current upper bound}

A child of the current node being expanded is added to the list of 'live nodes' (nodes that will be expanded later) if the cost of the child is less than the global upper bound. (Note: the list of 'live nodes' is maintained in Least-Cost order; hence, the name Least-Cost Branch and Bound).

PARALLEL DESIGN

One goal of this research was the investigation of a parallel design methodology. After reviewing several common design strategies, the object-oriented design methodology was selected for the research (3,4). The basic concept of an object design is the decomposition of the problem into objects, operations, and communications among the objects. Because the hypercube architecture is defined by the process model of computation, where 'processes' communicate using 'messages', parallel solutions defined by an object design map naturally into the hypercube. The programmer can describe the problem as fine-grained objects using the object model. These objects can be mapped directly to hypercube processes, or for efficiency and reduced communications, a collection of objects can be implemented as a hypercube process. Figure 5 shows the configuration of the parallel branch and bound search objects and the communications dependencies among those objects. The first process is called the Control Process. It is defined by the objects in the top processor box (see Figure 5). The meta-controller, terminate check, and bound check serve as global control throughout the parallel search. The Control Process resides in Node 0 of the hypercube. The remaining nodes contain the Worker Process. The task of each Worker Process is to find the best answer to a subproblem. A subproblem for a branch and bound search is equivalent to searching a subtree of the solution space for the best answer in that subtree.

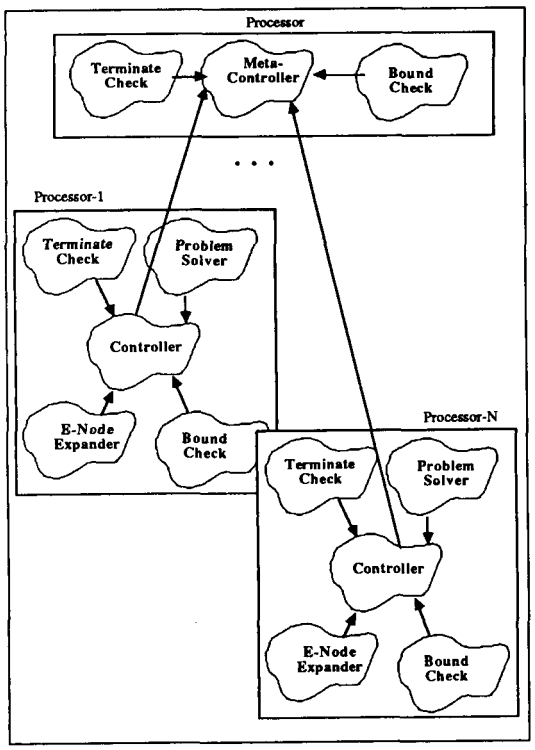


Figure 5: Object Visibility Diagram

During the search, the Control Process monitors the progress of the search by creating an initial set of subproblems to solve, sending those subproblems to Worker Processes, and terminating the search. Upon receiving a problem, the Worker Process finds the best answer (in that subtree). Once the entire subtree has been examined, the Worker Process posts a 'work request' to the Control Process and waits for additional work. The entire problem is finished when the Control Process does not have any problems to solve and all Worker Processes have posted 'work requests'. This translates to a machine state where no more work is available and all workers need a problem to solve.

EXPERIMENTAL RESULTS

Three measures, Computation Time, Speed Up, and Load Balance, were used to categorize the performance of the parallel search problem on the iPSC Computer. First, Computation Time measures the run time of a problem. Because the parallel computational environment involves additional processes, one representation of the total computation time of an algorithm is defined by the following formula (1:95),

$$T_N = T_S + T_C + T_W \qquad (1)$$

where

T_N = Computation Time for N processors
T_S = Start Up Time
T_C = Processor Computation Time
T_W = Wind Down Time

Start Up Time, T_S, measures the time to initialize the parallel processor before any parallel computation begin. Start up may include such things as initial parsing of the job, initial message transfers, or down load time of the programs to the parallel machine itself. The second term, processor computation time, measures the time the computer spends actually solving the problem. This term is common in sequential processor run time analysis. The final term in equation 1 is wind down time. This time accounts for the gathering of results from the various processors in the computer and analyzing or tallying those final results.

The second measure for the performance evaluation, Speed Up, compares the time to compute a solution using one processor and the time to compute a solution using N processors. It is defined as follows (14:28),

$$S = T_1 \div T_N \qquad (2)$$

where

T_1 = Time to computer a result with one processor
T_N = Time to computer a result with N processors (Eqn 1)

The speed up of a problem run in parallel is intuitively easy to understand. If a problem can be parsed into N sub-problems, with each subproblem taking 1/N of the total computation, then the maximum speed up of N is achieved. The perfect speed up, N, is highly unlikely because of the overhead of start up and the wind down time. Communications among processing elements also induce limitations on this measure.

Finally, the third measure, load balance, may be helpful in identifying computation bottlenecks. Because of the nature of the design and the branch and bound problem, the 'load' is defined to be the number of nodes expanded by a Worker Process. When plotted against the average load performed across all Worker Processes, balanced and unbalanced work loads can be identified. Single-Instruction-Multiple-Data (SIMD) problems that partition data to promote parallel activity tend to have regular communication and computation cycles. These classes of problems show the best performance under balanced work loads (9). Since parallel search is a Multiple-Instruction Multiple-Data (MIMD) problem, the communication and computation cycles cannot be guaranteed to be regular. Hence, the load balance measure must be evaluated along with the other performance measures before drawing conclusions.

Baseline Performance

The baseline of performance for this research is a Digital Equipment Corporation VAX 11/785 running the 4.2 BSD (Berkeley Software Distribution) UNIX operating system. The configuration of the machine used for this research has 8 Megabytes of main memory and 1800 Megabytes of disk storage. The sequential versions of the Deadline Job Scheduling problems were programmed in C Language, and the time information was obtained using the UNIX "times" function. Of the four parameters measured by the "times" function, this research focused on user_time. The user_time of a process is that time devoted to computation. The overhead associated with system calls, page swaps, etc. was not used for two reasons, (1) this research is actually interested in compute time of the algorithm and not operating system overhead; and (2) the VAX is under various system loads during the course of the experiments which would influence system time and the overall timing data.

Parallel Performance

First, the DJS problem was tested on the Intel iPSC simulator running on the VAX. While the simulator creates a good environment to learn how to program the iPSC, it does not show true parallel activity. Hence, it should not be used to fine-tune a problem. After porting the code from the

simulator to the actual iPSC, the original design was modified to achieve the best computation times. It should be noted that the object design worked well for an initial implementation, but the best performance results were attributed to fine tuning on the actual hardware. The only part of the parallel DJS used for fine-tuning was the iPSC Control Process. This process has the responsibility to create the initial set of problems to solve. At some point, it becomes beneficial to stop creating problems and to start handing them out to worker nodes. It should be noted the results of these experiments have fine-tuned to large problem sizes. For the parallel experiments, job sets from 4-Jobs to 25-Jobs were tested on six cube sizes, d-1, d-2, d-3, d-4, and d-5, where d = dimension). The d-0 cube could not create a data structure large enough to solve large DJS problems. Timing results for all runs was calculated using the iPSC Clock function on each node of the hypercube. Since the resolution of the iPSC Node Clock function is 1/60th of a second, some of the computation times were unmeasurable. The data in this section has been plotted for comparison and to show trends.

Before analyzing the results of the job scheduling experiments, a description of the test data is necessary. Since the deadline job scheduling solution uses least-cost branch and bound, them time to schedule a set of **n+1** jobs may take less time than scheduling **n** jobs. Therefore, two pseudo-equivalent classes of problems were devised such that the larger the job set created a more difficult problem to solve. Two reasons for creating pseudo-equivalent classes are, (1) the proof of equivalent classes of jobs is beyond the scope of this research; and (2) job set with these characteristics make the analysis a bit easier.

As described in a previous section, each job is defined by a 3-tuple (p_i,d_i,t_i), where p_i is the penalty incurred if the job is not scheduled, d_i is the deadline when the job must be finished running, and t_i is the time to run job i. With this information, the first set of problems (see below) guarantees that all jobs can be scheduled. The VAX solves this problem in O(n) time.

$$p_i = 1, \forall\ i$$

$$\sum_{i=1}^{n} t_i \leq \min(d_i)$$

The second pseudo-equivalent class is solved in exponential time by the VAX and it is described with the following values for the job 3-tuples,

$$t_i = i$$

$$p_i = 2^{*}t_i$$

$$d_i = \left\lfloor \frac{\sum_{i=1}^{n} t_i}{2} \right\rfloor$$

First, an analysis of the O(n) job set. Parallel processing appears to show no reductions in the time order complexity of O(n) problems (see Figure 6). The best performance was attributed to the iPSC d-1 and the best speed up was approximately 0.33 over VAX. Since this search problem degenerates to an examination of the left-most branch of the search tree, the problem does not map well to a parallel processor. The Load Balance analysis shows this result (see Figure 7). Basically, this problem cannot run in parallel. For small problem sizes, (scheduling 15 jobs or less) only one processor solves the problem while for large problem sizes, two iPSC worker nodes are used. This problem re-enforces the concept of maximum parallel activity because of limitations inherent to the problem.

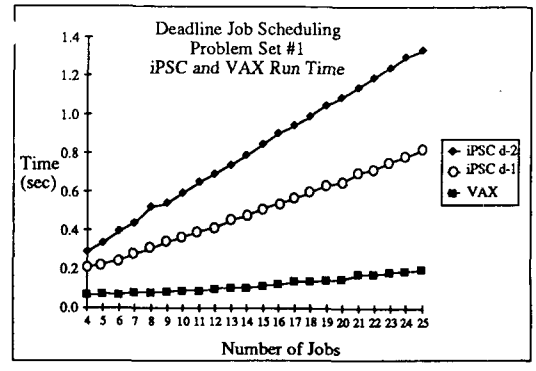


Figure 6: Deadline Job Scheduling- Problem Set #1
Computation Time

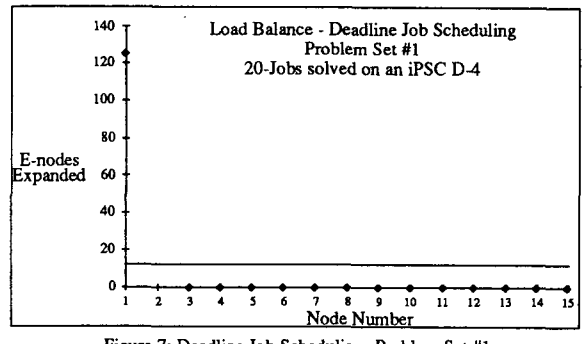


Figure 7: Deadline Job Scheduling- Problem Set #1
Load Balance of scheudling 20-Jobs on an iPSC D-4

Next, the second job set must be examined. The run time analysis shows that significant reductions in the time order complexity can be achieved. In Figure 8, the semi-log plot of the VAX computation time shows the inherent exponential nature of the problem. The iPSC d-4 and d-5 curves demonstrate the power of the parallel computation with problem sizes of 11 or greater. It should be noted the best speed up achieved was 58 times over VAX with a d-5 hypercube solving a 15-Job problem. The d-4 reached a speed up of 43 times over VAX (see Figure 8). A dimension-3 cube attained a speed up 33 times over VAX. Finally, d-2 and d-1 hypercubes solved the problem approximately 3 times faster than VAX. These results can be explained while examining the global upper bound during the parallel computation. Figure 9 shows the load balance of scheduling 15 jobs on a d-4 cube. Even though the load is unbalanced, the iPSC solved this problem 43 times faster than the VAX. Solving this same problem on a d-5 cube, all Worker Processes have node expansion counts (loads) equal to zero. This anomaly is attributed to the global upper bound. In a d-4 cube, the Control Process generates 64 initial problems. As the workers solve these problems

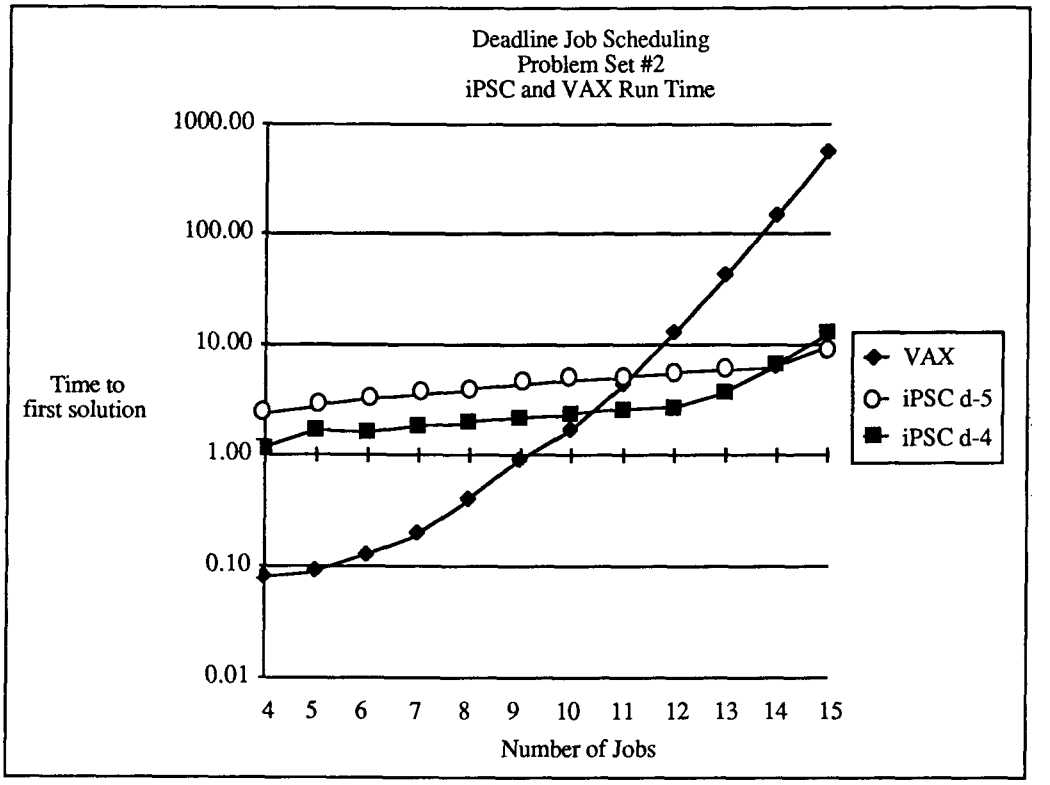


Figure 8: Deadline Job Scheduling- Problem Set #2
Computation Time

concurrently, the global upper bound converges quickly to the best upper bound of the entire search space. Once the upper bound converges, the workers no longer search the subtrees. They only prune the remaining search space. In the case of the d-5 hypercube, the Control Process generates 128 initial problems to solve. At this point the upper bound has all ready converged, and the search quickly ends with the workers just pruning the search space and never actually searching the subtrees.

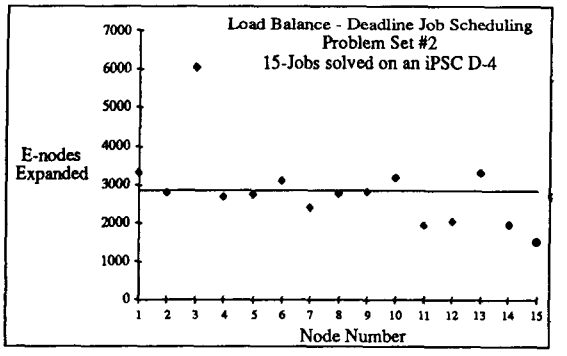


Figure 9: Deadline Job Scheduling- Problem Set #2
Load Balance of scheduling 15-Jobs on an iPSC D-4

CONCLUSIONS

The main objective of this research was the performance evaluation of search problems on the hypercube architecture. First, conclusions from the other research goals. The results of the first goal identified the object-oriented design methodology as a good design approach to map a problem into a parallel solution. The object model worked well for this research. The results of the object design resulted in a fine grained mapping of the problem space, and the implementation of the design focused on collecting several objects into coarse grained iPSC processes. During the design, details of the branch and bound problem were not overlooked and during the implementation, inefficiencies of communications were reduced. Even though the initial design needed fine tuning to achieve the best performance, the implementation of the initial design created a good prototype. As a recommendation for future research, the object design methodology should be extended to other parallel processors such as shared memory machines or other hypercube architectures. As noted with this research, an object design worked well for the hypercube because of the similarity of object design and the process model of computation. Additional tests of object-oriented design will test the flexibility and suitability of the design methodology as a general approach to map a problem into a parallel architecture.

The second goal of this research was the performance evaluation of search problems on a parallel processor. As the results show, a sequential problem solving technique, like search, can be mapped to a parallel processor and speed ups over traditional sequential machines can be achieved. In fact, over a narrow range, the parallel solution reduced the time order complexity of the problem. But, the results of the O(n) job set also re-enforced the concept of maximum parallel activity due to limitations within the problem.

C-3

Finally, the third goal of this research was to examine the suitability of the hypercube architecture to solve search problems. Branch and bound is a sequential programming technique with centralized control. The parallel solution presented in this paper was mapped onto an extremely loosely coupled architecture. Even though this research successfully produced speed ups, the nature of the hypercube architecture and the nature of the problem are not similar. Therefore, parallel search should be examined on other, more tightly coupled architectures, such as shared memory machines. Yet another approach to speed up search problems is to design new algorithms instead of mapping sequential techniques to parallel processors.

BIBLIOGRAPHY

1. Akl, Selim G. et al. "Design, Analysis, and Implementation of a Parallel Tree Search Algorithm," IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-4: 192-203 (March 1982).

2. Barr, Avron, and Edward A. Feigenbaum. The Handbook of Artificial Intelligence, Vol 1. Standford, California: HeurisTech Press, 1981.

3. Booch, Grady. "Object-Oriented Development," IEEE Transactions on Software Engineering, SE-12: 211-221.

4. Booch, Grady. Software Engineering with Ada. Menlo Park, California: Benjamin/Cummings, 1983.

5. Bosma, John T. and Richard C. Wheelan. Guide to the Strategic Defense Initiative. Arlington, Virginia: Pasha Publications, 1985.

6. Fenkel, Karen A. "Evaluating Two Massively Parallel Machines," Communications of the ACM: 29 752-758 (August 1986).

7. Horowitz, Ellis, and Sartij Sahni. Fundamentals of Computer Algorithms. Rockville, Maryland: Computer Science Press, 1978.

8. Intel iPSC Concurrent Programming Workshop Notes, Intel Scientific Computers, Beaverton, Oregon. (16-20 June 1986).

9. Lee, Lieutenant Ronald. Performance Comparison and Analysis of State of the Art Machines. MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1986.

10. Norman, Captain Douglas O. Reasoning in Real-Time for the Pilot Associate: An Examination of a Model Based Approach to Reasoning for Artificial Intelligence Systems using a Distributed Architecture. MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1985.

11. Patton, Peter C. "Multiprocessors: Architectures and Applications," Computer: 29-40 (June 1985).

12. Retelle, LtCol John P. Jr. "The Pilot's Associate-Aerospace Application of Artificial Intelligence," Signal: 10-105 (June 1986).

13. Rich, Elaine. Artificial Intelligence. New York: McGraw-Hill, 1983.

14. Seitz, Charles L. "The Cosmic Cube," Communications of the ACM: 28 22-33 (January 1985).

15. Siegel, Howard Jay, and Robert J. McMillen. "The Multistage Cube: A Versatile Interconnection Network," Computer: 65-76 (December 1981).

16. Seward, Walter D., and Nathaniel J, Davis IV. "Opportunities and Issues for Parallel Processing in SDI Battle Management/C3." Presented at the AIAA Computers in Aerospace V Conference, October 1985.

17. Stankovic, John A. et al. "A Review of Current Research and Critical Issues in Distributed System Software," Distributed Processing Technical Committee Newsletter, 7: 14-47 (March 1, 1985).

TASK ALLOCATION IN A DISTRIBUTED COMPUTING SYSTEM

Walter D. Seward
Air Force Institute of Technology
Department of Electrical and Computer Engineering
Wright-Patterson AFB, OH 45433

ABSTRACT

Distributed computing offers the potential for improved system's performance for many applications. Critical to the realization of this performance improvement is a methodology for task allocation which considers both the application requirements and the system architecture. This paper examines a conceptual framework for task allocation in distributed systems and discusses application and computing system parameters critical to task allocation decision processes. The paper addresses task allocation techniques which focus on achieving a balance in the load distribution among the system's processors. That is, equalization of computing load among the processing elements. Examples of system performance are presented for specific applications. Both static and dynamic allocation of tasks are considered and system performance evaluated using different task allocation methodologies.

INTRODUCTION

Recent advances in the development of microcomputer systems has increased interest in the use of distributed computing systems. Certain applications such as the Battle Management and Command, Control, and Communications (BM/C3) requirements of a strategic defense system appear naturally as distributed computational systems.[1] In addition, despite the impressive speed of the current generation of computers, their architecture limits them to a mostly serial approach to computation, and limits their usefulness for problems that are computational intensive and which may require processing speeds upwards of 100 million operations per second. Physical limits suggest that these traditional, serial architectures offer little hope of large performance improvements. Distributed and parallel processing systems offer an opportunity for improved system performance, reliability and flexibility. Critical to the realization of increased system capabilities is an effective means of allocating the processing tasks among the system's computing resources. Without an effective scheme of task allocation, the performance of the distributed system can be degraded to something less than that of one of the system's single processors.

This paper presents preliminary results of research performed to analyze the performance of task allocation methodologies for a distributed computing system. For this paper, distributed processing is considered as a special case of parallel processing where the processing elements are loosely coupled and any exchange of information and control of the system must take place via an interconnection structure instead of by means of shared memory. However, the fundamental approaches used in this work do not preclude use with a tightly coupled parallel processing system. The appli-cation program used in this analysis was that of Integer Linear Programming (ILP). ILP is representative of a class of algorithms which is applicable to the solution of several problems within the BM/C3 environment of strategic defense. Results of this analysis show that dynamic task allocation can provide significant performance improvement. Also discussed are planned extensions to this research.

PERFORMANCE OF DISTRIBUTED SYSTEMS

Evaluation of alternative system implementation must be based a relevant metrics. Because increased speed of computation is one of the primary reasons for using parallel systems, the speed of the parallel algorithm is one of the most important parameters for system evaluation. The most frequently used measures of parallel and distributed system performance are

speed-up and efficiency. These measures are defined as follows: [2]

$$S = \frac{\text{worst-case running time of fastest known sequential algorithm for the problem}}{\text{worst-case running time of parallel algorithm}} \quad (1)$$

$$E = \frac{\text{worst-case running time of fastest known sequential algorithm for the problem}}{\text{cost of parallel algorithm}} \quad (2)$$

where,

S = Speed-up
E = Efficiency

and the cost of parallel algorithm is defined to equal the product of the parallel running time and the number of processors used.

The ratio of single processor time to parallel system time can be expressed as $S = T_1/T_N$ where T_1 is the

single processor time and the computation time using N processors, T_N, is determined by the following formula:

$$T_N = T_S + \text{Max}\ (T_C) + T_W \quad (3)$$

where

T_N = Computation Time for N processors

T_S = Start-up Time

Max (T_C) = Time required for last busy processor to complete its computation

T_W = Wind-Down Time.

Start-up time measures the time that is required to initialize the system before any computations can begin. Wind-Down time refers to the time required to collect from the various system processors and analyzing or tabulating into a final product. The time required for the last busy processor to complete its computations includes several elements. Included is all of the time during which this processor and all other processors in the system were actually performing useful computations. Also included is any idle time, time required to communicate with other processors, and time required to perform overhead functions built into the parallel algorithm.

Maximum speed-up, which is the overriding objective for most parallel processing applications, occurs when the parallel processing time is minimized. The elements of the equation for the parallel processor time are not independent. The algorithm, the architecture and their implementations determine the specifics of the relationships among these parameters. However for applications of interest the predominant factor, by orders of magnitude, is the computation time. Thus, speed-up is maximized when $\text{Max}(T_C)$ is minimized.

Balanced Computational Load

There are three rules of thumb for minimization of $\text{Max}(T_C)$ and, therefore, maximizing the performance of parallel processing systems: [3]

(1) Distribute the computation load evenly;

(2) Maximize the computation time to communication time ratio;

(3) Minimize communication distance.

These rules of thumb, unfortunately, may conflict. For example, in order to maintain a balanced computational load additional communications overhead may be required. Thus $\text{Max}(T_C)$, which includes both computation and required interprocessor communications, must be considered in its entirety. Just maintaining a balanced workload with out consideration of associated overhead and memory system costs may result in decreased system performance. This is the task allocation issue being addressed in this research.

TASK ALLOCATION

Concepts and techniques for task allocation or task scheduling have evolved from the considerable body of work on job-shop or assembly-line problems. [4,5] Work in this area has been has been extensively documented in management science, operations research, and computer science and engineering literature. The two fundamental approaches to task allocation are static and dynamic. Each of these techniques has advantages and disadvantages which are a function of the application and the system architecture.[6]

Static Allocation

Static allocation involves the a-priori determination of the allocation of tasks. This method is the least complex and requires the least overhead

of the system during its operation. In systems where complete knowledge of the computational requirements of the application tasks and the system's operational characteristics are known, static allocation of task can be the most efficient. Minimal resources are required during run-time while maintaining high levels of efficiency. But, solution of the static allocation problem in these cases generally requires solution by means of an off-line integer programming problem or other computational intensive combinatorial algorithm. Furthermore, when knowledge of the system operational characteristic while dedicated to a specific application, or if the exact computational requirements are not known, static allocation is not generally effective. In these cases, all advantages of parallelism in the problem structure and architecture may be lost and the total computational load assigned to a small subset of the available processors.

Dynamic Allocation

Dynamic allocation of tasks is an attempt at maintaining the most efficient use of the system's computational resources by adapting, at run-time, the load distribution to the demands of the computational tasks. It involves assessing the availability of each processing element to accept additional work and reallocation of the total system computational load so that a stated performance metric is optimized. The process of obtaining an optimal result is itself a large-scale combinatorial optimization problem, and therefore dynamic allocation techniques are most often based on heuristic procedures. Moreover, the difficulty of the task is increased by lack of precise information about the application program's requirements and the computational characteristics of the system. These factors must, in general, be estimated using some method of heuristic or statistical technique. The objective of any technique of dynamic allocation is to optimize the use of the system communications, computation, and memory resources, but this requires minimization of overhead for execution of the allocation routine, which requires specific knowledge of the distributed system architecture.

DISTRIBUTED SYSTEM ARCHITECTURE

Evaluation of the system's performance require consideration of both communications and computational dependent parameters specific to the system implementation. The performance of any dynamic task allocation method depends upon not only the application, but these characteristics of the computing system's architecture.

The distributed architecture system used for this study was the Intel iPSC system.[3] This system is based on a hypercube topology where each node of the topology is a processor with its own memory and communications capability. The system operation is based on a process model of computation supported by message passing. A hypercube or binary n-cube is a network of 2^n processors each with direct communications with its n neighbors. The number n defines the dimension of the cube. Figure 1 illustrates a cube of dimension 4.

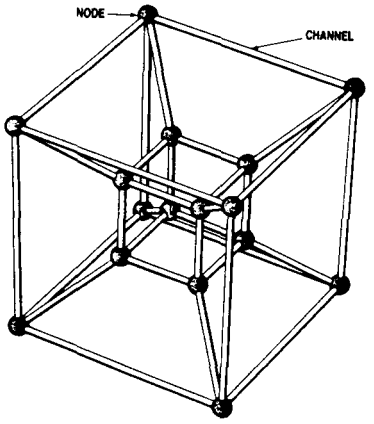


Figure 1 Dimension Four Hypercube.[3]

The iPSC can be configured with modules of 16, 32, 64 or 128 interconnected nodes. The processors which form the elements of the hypercube are called nodes. Each node is numbered by an n-bit binary number between 0 and 2^n-1. Node 0 is connected to a separate processor called the "cube manager" which provides the input and output link between the nodal processors and the outside world.

Communications within the cube is based on a static routing algorithm which routes messages by a minimum number of hops along the edges of the cube. Although there is support for communications processing on each node, the nodal processor, an Intel 80286 chip, must perform some of the tasks associated with forwarding a message. Hence the pattern of communications within the application's implementation may have a significant impact on the overall performance.

The iPSC system provides support for software developed in 80286 assembly language, C, and Fortran 77. The algorithms developed for this paper were implemented in Fortran 77. This language was chosen because compiled Fortran 77 is faster than C on the iPSC.

INTEGER LINEAR PROGRAMMING

The specific problem analyzed in this research is a distributed computer algorithm for the solution of Integer Linear Programming problems. In matrix representation, Integer Linear Programming (ILP), is an optimization problem which consists of finding a vector $\underline{x}$ which maximizes (minimizes) a linear objective function $\underline{C}*\underline{x}$ subject to a set of linear constraints $\underline{A}*\underline{x} = \underline{b}$, $\underline{x} \geq 0$, and $\underline{x}$ integer. $\underline{A}$ is an m by n matrix, $\underline{C}$ is a 1 by n row vector, and $\underline{b}$ and $\underline{x}$ are n by 1 column vectors all of which are integer.[7] Such problems are frequently posed as problems with inequality constraints which must be transformed to the required format by adding "slack" and "surplus" variables.[7] Although the problem is easily expressed it is extremely difficult to solve in practice. It has been shown, in fact, that ILP is NP-complete.[8]

Among the techniques frequently used to solve the ILP is an enumerative technique called the branch-and-bound.[5] A naive approach to enumerative solution of an ILP involves the explicit examination of each possible integer vector to determine the one which results in maximization (minimization) of the objective function. However, even for relatively small ILP's the number of possibilities, while finite, becomes excessively large. Consider for example an ILP with ten unknown variables each of which has an integer range of zero to ten. There are more than ten billion possible solutions to this problem. Explicit enumeration of the possible solutions quickly becomes overwhelming.

The branch and bound method of solution of the ILP uses a combination of explicit and implicit enumeration to reduce the total number of permutations which must be examined. The branch and bound technique can best be described in terms of a search tree. The height of the tree is determined by the number of integer variables, n, in the ILP. A path from the root node to a leaf node which satisfies all problem constraints corresponds to a feasible integer solution. At a given level of the tree the nodes correspond to specific integer values for one of the problem variables. Figure 2 illustrates a search tree for a three variable ILP where the x_1 has a domain of {0,1,2} and x_2 and x_3 each have a domain of {0,1}.

Initially the ILP algorithm performs an unconstrained solution to the linear programming problem which results if no integer constraints are applied. If no feasible solution exists to the unconstrained problem, then no solution exists for the ILP. The value of the objective function for the feasible solution to the unconstrained problem represents an upper (lower) bound on the ILP objective function for the maximization (minimization) problem.

The branch and bound algorithm used in this research performs a depth first search of the enumeration tree successively constraining the integer variables to specific values. As each variable is constrained, the value of the objective function for the feasible partial solution is compared with the existing lower (upper) bound to determine if further search along that path is justified. The lower (upper) bound is determined by the maximum (minimum) objective function value for feasible integer solution to the ILP. As a new larger (smaller) bound is determined it replaces the existing lower (upper) bound. The bound defined by the feasible integer solutions determines the criteria for eliminating nodes from the search tree before extending them. Any partial solution which falls below (above) the feasible solution bound cannot represent an improved solution to the maximization (minimization) problem.

Distributed Computation of ILP

The ILP was implemented on the Intel iPSC distributed computing system in the following fashion:

1. One node of the hypercube is designated the responsibility of coordinating the computation of the other processors in the system. This controller node partitions the problem space, assigns blocks of the partition to independent processing elements, collects intermediate results and shares these results with the other processors, and maintains the status of the progress of the solution.

2. The "worker" nodes of the system perform the ILP algorithm on assigned blocks of the problem space partition. These nodes inform the controller node of improved results, current status in solving the assigned partition, and if they are idle.

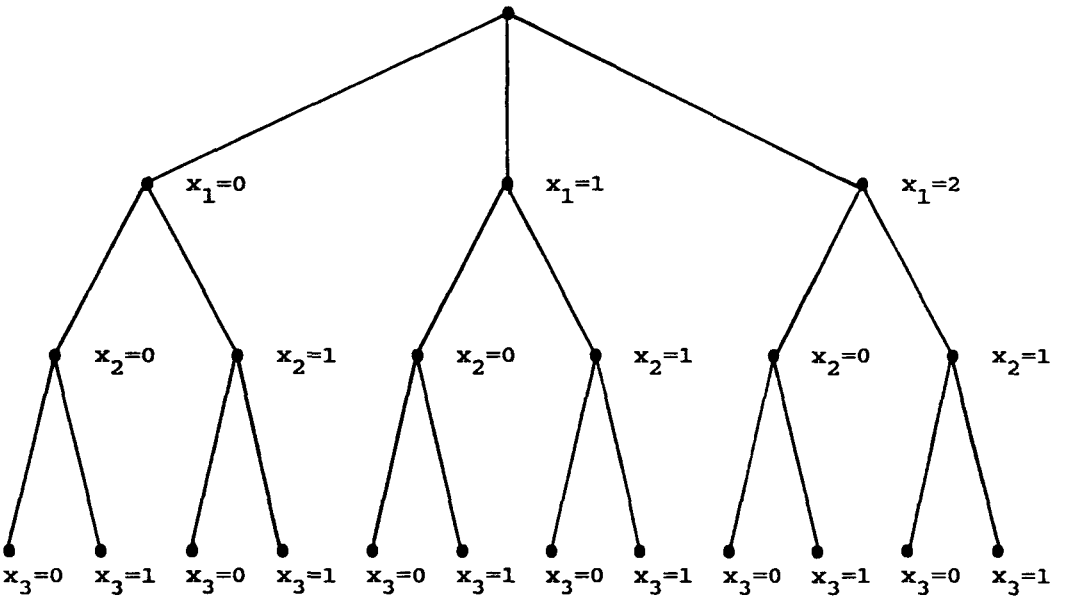


Figure 2 Example Enumeration (Search) Tree

Computation performed by the worker nodes includes the branch-and-bound algorithm. These nodes send to the controller node the objective function value for each improved integer feasible solution. This information is shared with all other nodes so that more rapid implicit enumeration is possible. These nodes also provide the the controller node with a status vector which represents their progress in solving assigned subproblems of the ILP problem.

RESULTS AND ANALYSIS

This section presents the results of experiments performed using the Intel iPSC to compute the ILP for a group of problems which range in complexity from easy to moderately difficult. Table I contains the descriptions of these problems. All of these problems are stated in the form of $\underline{A}*\underline{x} \leq \underline{b}$. The performance of these ILP problems using differing configurations of the iPSC and differing methods of task allocation are discussed in the following sections.

Baseline Performance

A baseline of system performance for the four problems is presented in Table II. The processor time required to perform a sequential version of the ILP computation using a VAX 11/785 and a single node of the iPSC are compared. The ratio of the VAX time to the node processing time is seen to remain relatively constant at approximately 0.285. As can be seen for Problem 4, the time required on either system is excessive for most environments. These results are based on the depth first search of the enumeration tree and are dependent upon the speed with which the lower bound converges to the optimal solution. The longer it takes to converge the larger the problem space which must be explicitly evaluated.

Table III presents the "best case" times for these problems using a single node of the iPSC. These results were obtained by setting the lower bound to the previously determined optimal value and allowing the search algorithm to perform maximal implicit enumeration. The results show a substantial speed-up for all but Problem 2. Problem 2 does not show the same improvement because its optimal solution is encountered early in a depth-first search. Hence implicit enumeration quickly dominates the enumeration scheme even for the sequential implementation.

The results obtained by this method are not representative of normal performance. They depend upon a-priori knowledge of the answer. These results are included so that the parallel system performance can be compared with the best possible sequential performance for the specific algorithm used.

Static Partition and Allocation

Except for those applications where knowledge of the problem space is complete enough for static partition and allocation to provide sufficient

Table I

Integer Linear Programming Problems Used for System Evaluation

Problem 1

Objective Function = $4x_1 + 8x_2 + 5x_3$

Variables

x1	x2	x3	b
1	2	3	18
1	4	1	6
2	6	4	15

Problem 2 [7:370]

Objective Function = $x_3 + x_4 + x_5$

Variables

x1	x2	x3	x4	x5	b
2	3	1	2	2	18
3	2	2	1	2	15
-6	0	1	0	0	0
0	-7	0	1	0	0

Problem 3 [7:371]

Objective Function = $x_7 + x_8 + x_9 + x_{10} + x_{11} + x_{12}$

Variables

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	b
9	7	16	8	24	5	3	7	8	4	6	5	110
12	6	6	2	20	8	4	6	3	1	5	8	95
15	5	12	4	4	5	5	5	6	2	1	5	80
18	4	4	18	28	1	6	4	2	9	7	1	100
-12	0	0	0	0	0	1	0	0	0	0	0	0
0	-15	0	0	0	0	0	1	0	0	0	0	0
0	0	-12	0	0	0	0	0	1	0	0	0	0
0	0	0	-10	0	0	0	0	0	1	0	0	0
0	0	0	0	-11	0	0	0	0	0	1	0	0
0	0	0	0	0	-11	0	0	0	0	0	1	0

Problem 4 [9:236]

Objective Function = $-10x_1 + 7x_2 - x_3 + 12x_4 - 2x_5 - 8x_6 + 3x_7 + x_8 - 5x_9 - 3x_{10}$

Variables

x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	b
-3	-12	8	1	0	0	0	0	8	-2	8
0	1	10	0	5	-1	7	1	0	0	13
-5	-3	1	0	0	0	0	-2	0	-1	-6
5	3	-1	0	0	0	0	2	0	1	6
0	0	4	-2	0	5	1	-9	2	0	8
0	9	0	-12	7	-6	0	2	15	3	12
8	5	-2	-7	1	0	-5	0	10	0	16

performance, static techniques tend to be unsuccessful. System performance using static partition and allocation of the ILP problem space depends on the application program size and the created subproblem (blocks of the partition) granularity.

The static algorithm partitions the problem space into one or more subproblems per processor. The subproblems are allocated first-in-first-out (FIFO) to any idle processor until the subproblems created in the original partition are depleted. Once the list is empty, processors remain idle after completing their assigned subproblems. Insufficient numbers of subproblems quickly results in system speed which approaches that of a single processor. The larger the problem space the more likely static techniques will not substantially improve performance. This occurs because prevention of single processor degeneration requires very large numbers of subproblems which in turn results in high communications to computation ratios.

For example, even when the solution of Problem 4 is tuned by providing the optimal solution the overall solution time (with 31 nodes and 2000 subproblems per node) is only 15 percent better than the single processor performance, and the computational load is distributed such that one processor is always busy while the other 30 processors are busy only 4.6 percent of the time.

Table IV shows the results of static partition and allocation for varying problems with the numbers of subproblems fixed at 250 per node. Table V illustrates the performance of static

methods using a range of partition sizes on one of the problem sets. Clearly general techniques must involve methods other than static allocation.

Dynamic Allocation Procedures

The dynamic allocation procedures implemented in the algorithms used in this research incorporate several heuristic techniques. Maximum parallelism of the computation results if all processors initial idle time is minimized. Toward this end, the initial static partition and allocation is performed as specified for the static algorithm. Once the initial set of subproblems is depleted, any processor that becomes idle notifies the control node of its status. The control node is responsible for finding additional work, if it exists, and interrupting a busy processor so that the remaining work can be partitioned and reallocated. It is the procedures used to determine which processor to interrupt and how to partition that processor's work load that incorporate heuristics.

The decision as to which processor to interrupt is based on which processor has been busy for the longest time. It is assumed that this processor has not been able to perform implicit enumerations on most of its subproblem, and therefore, contains many feasible integer solutions. Once the processor is interrupted, the number of subproblems to form must be decided. To form only one subproblem is counter productive. The time lost to overhead of interrupt and partition will be repeated each time a processor becomes idle. Instead, the partition creates many subproblems which are used to create a FIFO problem queue, as is done for the initial partition, therefore no other processors are interrupted until the queue is empty. The number of subproblems to create for each interrupt is bounded below by a user specified minimum and calculated using a multiplicative factor determined by the partial solution object function value and the current maximum feasible object function. The closer to the maximum the more subproblems created. Finally, as the solution is approached the possibility exists for thrashing to occur. To prevent repeated interrupts of processors, a minimum busy time can be specified by the user.

Performance of Dynamic Task Allocation

The results of experiments using the dynamic task allocation method described above are presented in Tables VI and VII and Figures 3 and 4.

The results shown in Table VI illustrate the performance of the dynamic partition and task allocation algorithm on the four test problems. The results cited are the best measured performance for each problem as determined by the algorithm parameters of number of initial blocks in the partition, minimum number of blocks in each partition of an interrupted subproblem, and the minimum busy time. There is a clear performance improvement over the static allocation results shown in Table IV. Moreover, the performance improvement increases as the problem complexity increases.

As shown in Table VI, the speed-up for Problem Four exceeds linear, with respect to the number of processors, when compared to the single processor time without a-priori knowledge of the solution. With a-priori knowledge of the solution, the speed-up is less than linear but substantial. Further analysis of the the super-linear performance is in order.

Based on the serial processing algorithm implemented in this study, super-linear speed-up is achievable. However, could the performance of this algorithm be improved so that it approaches the performance measured with a-priori knowledge as shown in the "best case" performance? If such an optimal algorithm can be determined, then it should be applied. Improvements may come from different search techniques or improved implementation techniques; however, it is questionable if such an algorithm can be developed for the general case.

The advantage of the parallel solution of the ILP using a branch and bound algorithm is that the knowledge essential to rapid solution of the problem, the bounds, are determined more quickly. These results are broadcast to all computing elements, thereby accelerating their performance. If a sequential algorithm can be optimized, then each of the nodes could also use the algorithm. The performance improvement using multiple processors would approach linear, and differ only by the overhead associated with communications, start-up, and wind-down.

Table VII illustrates the performance of the dynamic scheduling algorithm applied to Problem Four for varying numbers of nodes. The speed of performance improves approximately linearly with the number of processors. These results were based on the same number of blocks in the initial partitions, the same number of blocks in the subsequent partition of

subproblems, and the same minimum busy time.

Figures 3 and 4 illustrate the criticality of the dynamic task allocation parameters. There exists a clear indication that specific values provide significant performance improvements. Moreover, certain values for the parameters result in a significant degradation in the performance. How the "optimal" parameters are determined is the subject of on-going research.

SUMMARY

The results illustrate that reasonable performance from a distributed computing system requires some method of dynamic task allocation. Such an allocation scheme must consider not only the application but the characteristic of the system architecture used for the implementation. Specifically the processing speed, the communication subsystem characteristics, and the memory requirements.

The results of pervious experiments demonstrates the need for an adaptive algorithm to "tune" the application algorithm's performance to the system architecture. On-going research is examining the development of automated techniques for generating the critical task allocation parameters. Under consideration are methods of stochastic estimation and heuristics for determining the best combination of allocation algorithm parameters. In addition, further work is being performed to improve the serial algorithm by consideration of different search strategies.

TABLE II

Serial Processing Time
(in Seconds)

PROBLEM NUMBER	INTEL iPSC (1 NODE)	VAX 11/785
1	11.1	3.6
2	49.8	14.8
3	4067.3	1193.6
4	108335.9	32052.0

TABLE III

"Best Case" Serial Processing Time
(in Seconds)

Problem Number	INTEL iPSC (1 Node)
1	0.795
2	49.685
3	750.295
4	3222.355

Table IV

Static Partition and Allocation
(31 Nodes)

Problem Number	Solution Time (Seconds)	Speed Up
1	4.12	2.69
2	45.81	1.09
3	1689.31	2.41
4	108289.00	1.00

Table V

Problem 3
Static Partition and Allocation
(31 Nodes)

Number Blocks Per Node	Solution Time (Seconds)	Speed Up
100	1691.6	2.4
200	1689.5	2.4
500	1664.3	2.5
600	1736.0	2.3
800	1736.1	2.3
1000	1691.7	2.4
1500	1691.6	2.4
2000	1689.5	2.4

Table VI

Dynamic Partition and Allocation
(31 Nodes)

Prob. #	Solution Time (Sec.)	T_1/T_N	"Best T_1"/T_N
1	3.22	3.45	0.24
2	13.86	3.59	3.58
3	488.23	8.33	1.57
4	155.09	698.54	20.78

Table VII

Problem 4
Dynamic Partition and Allocation

Nodes	Solution Time (Sec.)	T_1/T_N	"Best T_1"/T_N
4	1233.9	87.8	2.61
8	664.7	163.0	4.85
16	373.7	289.9	8.62
31	155.0	698.94	20.78

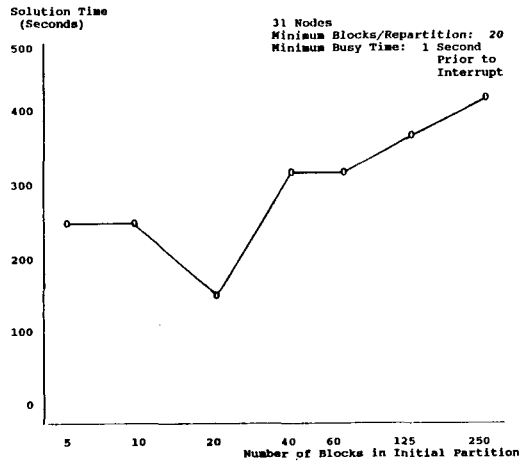


Figure 3 - Problem 4: Solution Time vs Initial Blocks

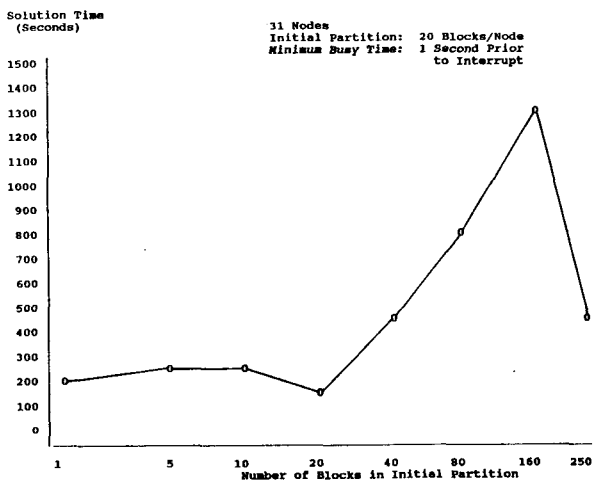


Figure 4 - Problem 4: Solution Time vs Initial Blocks

ACKNOWLEDGEMENTS

This research is supported in part by the Strategic Defense Initiative Organization. Also, I gratefully acknowledge the extensive software development and technical development of Captain Paul Bailor a PhD student at the Air Force Institute of Technology.

REFERENCES

1. Seward, Walter D. and Nathaniel J. Davis IV, "Opportunities and Issues for Parallel Processing in SDI Battle Management/C3," Unpublished report Presented at the AIAA Computers in Aerospace V Conf., Oct 1985.

2. Akl, Selim, PARALLEL SORTING ALGORITHMS, Academic Press, Orlando, FL, 1985, pg 8-9.

3. ..."Intel iPSC Concurrent Programming Workshop Notes," Intel Scientific Computers, Beaverton, OR, 16-20 June 1986.

4. Coffman, E. G. and Denning, P. J., OPERATING SYSTEMS THEORY, Prentice-Hall, Englewood Cliffs, NJ, 1973.

5. Kohler, W. H. and Steiglitz, K., "Enumerative and Iterative Computational Approaches," COMPUTER AND JOB/SHOP SCHEDULING THEORY, E. G. Coffman ed., John Wiley and Sons, NY, 1976, pg 229-287.

6. French, S., SEQUENCING AND SCHEDULING, Halsted Press, NY, 1982.

7. Garfinkel, R. S. and Nemhauser, G. L., INTEGER PROGRAMMING, John Wiley and Sons, NY, 1972.

8. Papadimtriou, C. H. and Steiglitz, K., COMBINATORIAL OPTIMIZATION: ALGORITHMS AND COMPLEXITY, Prentice-Hall, Englewood Cliffs, NJ, 1982.

9. Salkin, Harvey, INTEGER PROGRAMMING, Addison-Wesley, Reading, MA, 1975.

USER ENGINEERING: A NEW LOOK AT SYSTEM ENGINEERING

Larry L. McLaughlin
TRW Defense Systems
One Space Park
Bldg. 02 Room 2779
Redondo Beach, CA 90278

ABSTRACT

The dawn of the missile and space era gave birth to System Engineering, and the evolution of human computer systems has spawned a new concept called User Engineering.

Because users are diverse, and their requirements are always subjective, and initially vague, unknown or incompletely defined, existing methodologies are insufficient to identify user issues which then cause major disruptions throughout the requirements, design and development of the system. Human thinking and decision making in critical environments such as space stations or space defense command and control centers demand new engineering approaches to tackle user complexity and reduce system development risk.

User Engineering is a new System Engineering perspective responsible for defining and maintaining the user's view of the system. It's elements are a process to guide the project and customer, a multidisciplinary team including hard and soft sciences, rapid prototyping tools to build user interfaces quickly and modify them frequently at low cost, and a prototyping center for involving users and designers in an iterative way. The main consideration is reducing the risk that the end user will not or can not effectively use the system.

The process begins with user analysis to produce cognitive and work style models, and task analysis to produce user work functions and scenarios. These become major drivers of the human computer interface design which is presented and reviewed as an interactive prototype by users. Feedback is rapid and productive, and user effectiveness can be measured and observed before the system is built and fielded. Requirements are derived via the prototype and baselined early to serve as an input to the architecture and software design.

INTRODUCTION

This paper details a methodology that has been used successfully to engineer complex human computer systems. It is a "process" technology, relating to how the government and corporations can go about conceptualizing, defining, and building systems. The motivation for this new process is described in terms of problems with existing approaches and the pressing need for a dramatic change in the nature and goal for these systems

The new process, called "User Engineering", is defined and related to the other perspectives of system engineering. Then it is described in terms of four primary elements: methodology, tools, team, and laboratory. Examples of systems where this technology has successfully been applied and the benefits are summarized.

PROBLEM

User interactive systems differ markedly from systems the government and aerospace companies have built in the past. Those systems were driven by critical simulation and/or data processing requirements, but required limited human interaction. With user interactive systems, the primary consideration is providing knowledge integration and fast problem solving, using powerful workstations connected to information sources supported by decision aids (Shneiderman, 1987).

Requirements for these systems are initially at best subjective and are often vague, unknown, and incompletely defined. Users do not know what they want until they see it, and frequently can neither describe how they do their work (Rasmussen, 1986), or communicate it to even the most able interviewers. Additionally, these requirements do not meet physical laws and can not be proven on paper in advance.

Classical approaches used successfully in the past on systems like the ICBM program or site defense, depend on functional partitioning and depend crucially on the study of interfaces. For human computer systems, these fail because of the need for symbiotic interaction between a primary subsystem, the human, and the computer. (Winograd and Flores, 1986)

The term "human computer interactive" that is now widely used to describe these systems, masks the incredible complexity and difficulties associated

with trying to combine the powers of the human brain and computers. Only recently has the aerospace industry begun to uncover the reasons why efforts to build such systems have had so many problems.

Analysis of costs for these systems reveals that for every dollar spent in development, approximately two and one half dollars are spent in operations and maintenance. Of the software development costs, thirty percent (30%) is breakage within the waterfall. It is now known that these costs are primarily traceable to failure to have complete, clear and consistent user/system requirements. More startling, of the software maintenance costs, forty percent (40%) is user enhancements. In other words, these costs are associated with making the system do what users suddenly discover they really needed in the first instance. In the worst case, over one third of the life cycle costs for these systems could be considered wasted if better requirements could be established early (Boehm, 1981).

GOAL

The challenge for each project then becomes how to understand and provide customized support for diverse users, to translate this knowledge into user/system requirements that achieve the best interface for their dialog with the system, and to produce the best technical design solution to accomplish needed system support. The ultimate system engineering goals, thus are to validate these requirements with users early enough to affect system design, and to architect systems to meet user needs, not force fit users to system architectures.

APPROACH

Traditionally, system engineering is understood to have four perspectives (Figure 1.). Mission engineering produces the operational concept for the system and concentrates on "who, what, why, where and how." Requirements engineering defines requirements in terms of "the system shall..." statements for contractual and testing purposes. Design engineering produces the physical view of the system, its architecture and functional partitioning, and simulations of performance. Implementation engineering makes decisions about order of build, languages, test beds, and testing. To strengthen the human analysis, a new perspective, we coined "User Engineering", is added to complement the others. It is responsible for defining, validating, and maintaining the user view of the system. User Engineering is supported by rapid prototyping to identify and resolve issues early, and identify requirements and drivers for the architecture design. The process is risk driven, as opposed to document driven, and thus initially avoids paper specifications which are costly to produce, read, and understand, and which often amount to little more than speculation about the utility and usability of the final system.

METHODOLOGY

The User Engineering process comprises seven steps that provide a framework (Figure 2.) to guide the engineering tradeoffs which lead to a baselined prototype and subsequent set of system and software requirements. The central and most important phases produce analyses of the user, the tasks the system will support, and how the users will interface with the system to perform work. For purposes of description, the phases are top down and hierarchical, but in practice are iterative, recursive, and flexible as needed. Each of the phases has a specific goal and product, and produces prototypes from the project's inception onward. The prototypes provide focus for continual interaction with users for exercising working models of the system which look and feel like the proposed system.

Specifically, the process begins with the definition of an operational concept, and the gathering of user and system data about how the current system (if it exists) works and could be improved. A user model is formed from results of interviews, observations, and cognitive, workstyle and personality measures. Then task analysis is then done from the user point of view, to complement the partitioning of internal system functions. Key to the task analysis is the preparation of scenarios with users to reflect more completely how their work is done. Candidate user interfaces are defined and the scenarios are prototyped to involve users hands-on. Software algorithms for critical functions can be included in the prototype and can operate on real or simulated data. User and designer timely feedback identifies issues and provides the information to do effective tradeoffs and make clear decisions.

As the prototype matures, it begins to reflect the final system at the level of detailed design. It looks, feels, and behaves like the real system, including simulating response times exactly. It remains as a tool throughout the development to interpret the system, and respond to the inevitably changing user requirements. Additionally, it provides a mechanism for measuring the performance of the user on the system in advance, and for beginning the development of training concepts. The user interface is baselined with attendant requirements documentation at a User Design Review, a new review proposed to occur at or before the System Design Review.

TOOLS

Key to successful User Engineering is the ability to build prototypes rapidly and be able to change them frequently at low cost. No customer will support a lengthy expensive process no matter what reduction in risk is promised. Prototyping tools must facilitate rapid construction (initial prototypes within 14 to 30 days), have real time interaction, be transportable to the user, and provide for user performance monitoring. Referred to as "user interface management systems", these tools promote rapid screen generation

and provide mechanisms to sequence the screens and couple them to special or library applications programs.

Additionally, the tools should (Figure 3.) allow User Engineers to select from any number of terminals, input/output devices, different user dialogues, and interchange these as required. Appropriate application programs and their associated data bases (e.g. map generation and display, or image processing) inherent to user interactive systems are included to facilitate effective modeling of systems.

TEAM

No single individual has the breadth of both "hard" and "soft" science expertise needed to analyze and design these complex human computer systems. Furthermore, a highly experiential process such as this requires a team approach (Figure 4.). Training in group process techniques is essential to effective utilization of both individual and team resources. Depending on the problem at hand, disciplines including organizational development, sociology, psychology, macroergonomics, human factors, man machine interface design, prototyping, and general system engineering must be represented. In addition, it is mandatory that the customers' end users and experts in the subject of the system be participants.

LABORATORY

It is also clear that computer centers, with noisy equipment and harsh lighting, are inappropriate settings for developing and gathering feedback about user interaction on the prototype. A laboratory, referred to as the "System Prototyping Center", must be established as a special environment (Figure 5.) keyed to the users experiencing of the proposed system. It has individual work areas into which proposed workstations or consoles are brought in and which are decorated to resemble the work place. The workstations house or are connected to prototyping and measurement tools. Another area in the center contains monitors, large screen projectors, video cameras and recorders, and storyboards for capturing and sharing ideas.

APPLICATIONS OF USER ENGINEERING

This technology is being successfully applied both in research trials and on contract at TRW for government customers (Figure 6.). These are all user based systems tending toward the most complex end of the spectrum of cognition, tasks, and MMI.

Ongoing analysis of these applications of User Engineering clearly indicates that the risks are dramatically reduced for requirements and design, and that users are more satisfied throughout the development and deployment of the system. Much less paper is prepared at the outset, as the prototype communicates the system concept quickly and effectively. User feedback is rapid, and meaningful, and issues are surfaced before the system is built and fielded.

Designers better understand what kind of architectures they should use, and the resulting systems are easier to extend and maintain. User productivity can be measured in advance, and training concepts can be explored on the prototype prior to system delivery. This approach can nicely complement more formal development processes which follow the User Design Review and are based on the specification derived from the prototype.

CONCLUSION

In summary, the User Engineering technology is the basis for a new way of developing heavily user interactive systems. Set within System Engineering, it focuses on the human stimulus response and the redefinition of the workplace. Prototyping is the key tool for rapid conceptualizing and communicating across customers, developers and users. This kind of experimenting provides the best approach to defining/validating requirements and getting hands on experience with the system before committing to large costly developments.

The incorporation of this technology into current practice is straightforward but requires modification of traditional modalities. Of utmost importance is obtaining active participation of end users, something not always deemed desirable by customers. Additionally, new procurement procedures are needed to insure these activities are conducted early. Formal documentation deliverables in contracts must initially yield to delivery of prototypes, and the analysis surrounding their development and trial use. These kinds of risk reduction techniques will in the long run not only improve user and system effectiveness, but will lower development costs as well.

ACKNOWLEDGMENTS

I would like to acknowledge Lance Valt for his full time support to the research and development of this technology and the contribution of ideas and words found in this paper.

REFERENCES

1. Shneiderman, B., "Designing the User Interface", Addison-Wesley Publishing Company, 1987

2. Rasmussen, J., "Information Processing and Human-Machine Interaction," North-Holland, 1986

3. Winograd, T., and Flores, F., "Understanding Computers and Cognition," Ablex Publishing Corporation, Norwood, New Jersey, 1986

4. Boehm, B., "Software Engineering Economics," Prentice-Hall, 1981

5. Norman, D. and Draper, S., "User Centered System Design," Lawrence Erlbaum Associates Publishers, Hillsdale, New Jersey, 1986

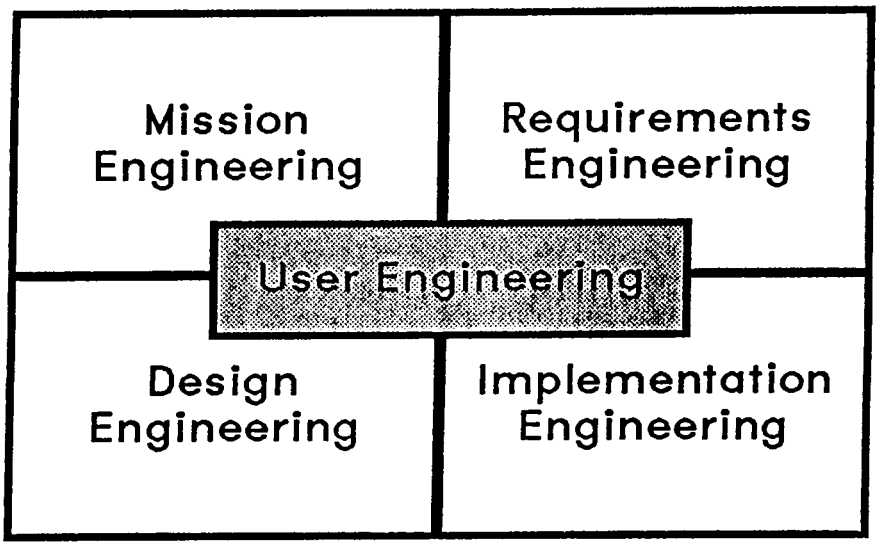


Figure 1. User Engineering Integrated Within System Engineering

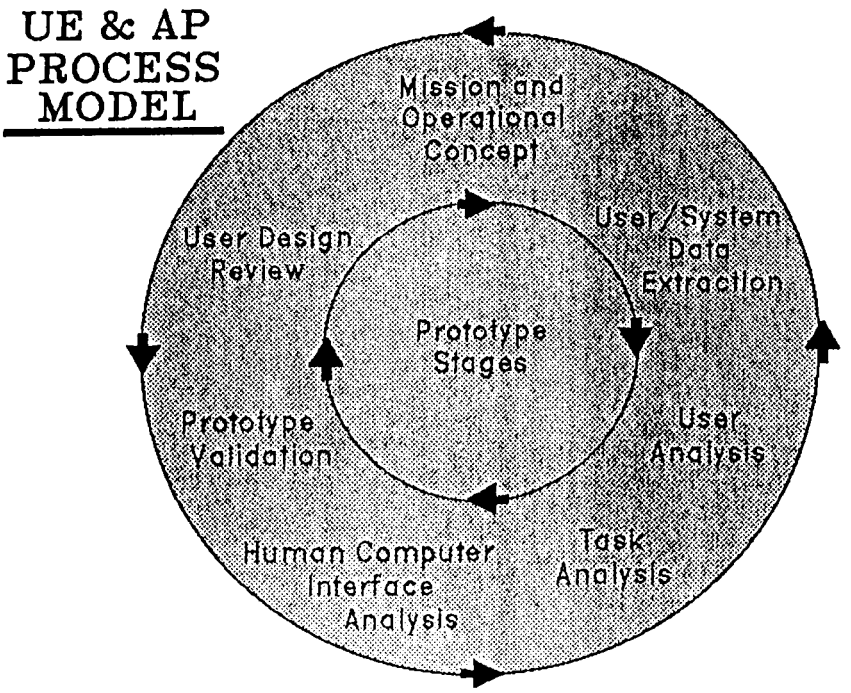


Figure 2. User Engineering Methodology

SOFTWARE PROTOTYPING TOOLS

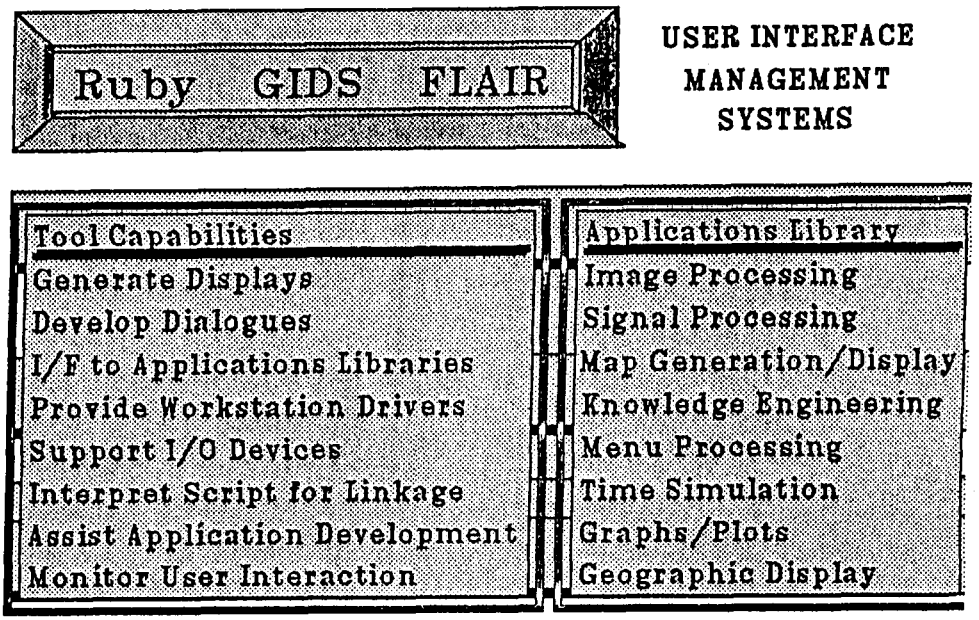


Figure 3. User Interface Prototyping Tools

INTERDISCIPLINARY USER ENGINEERING TEAM

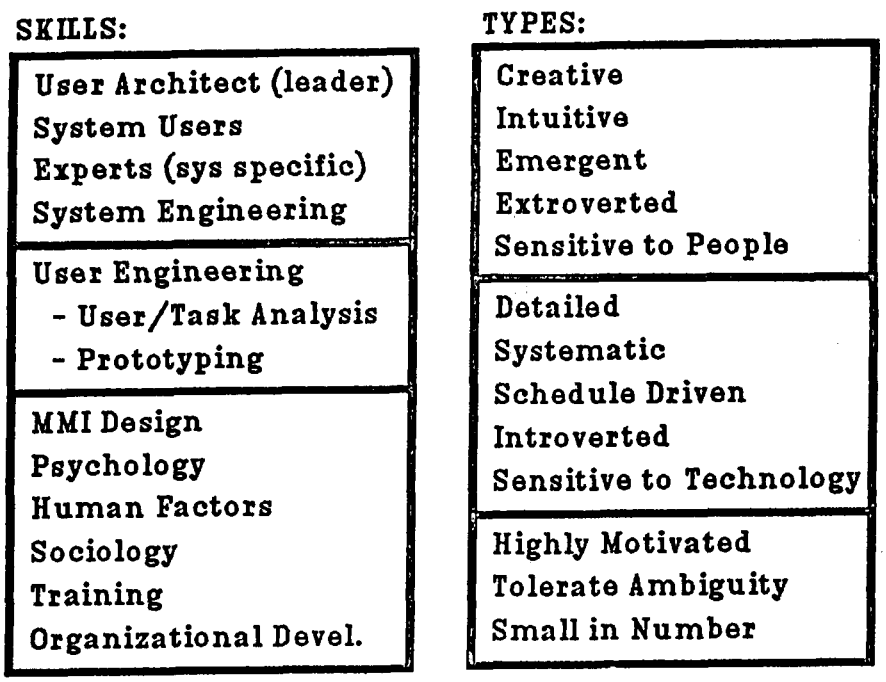


Figure 4. User Engineering Team

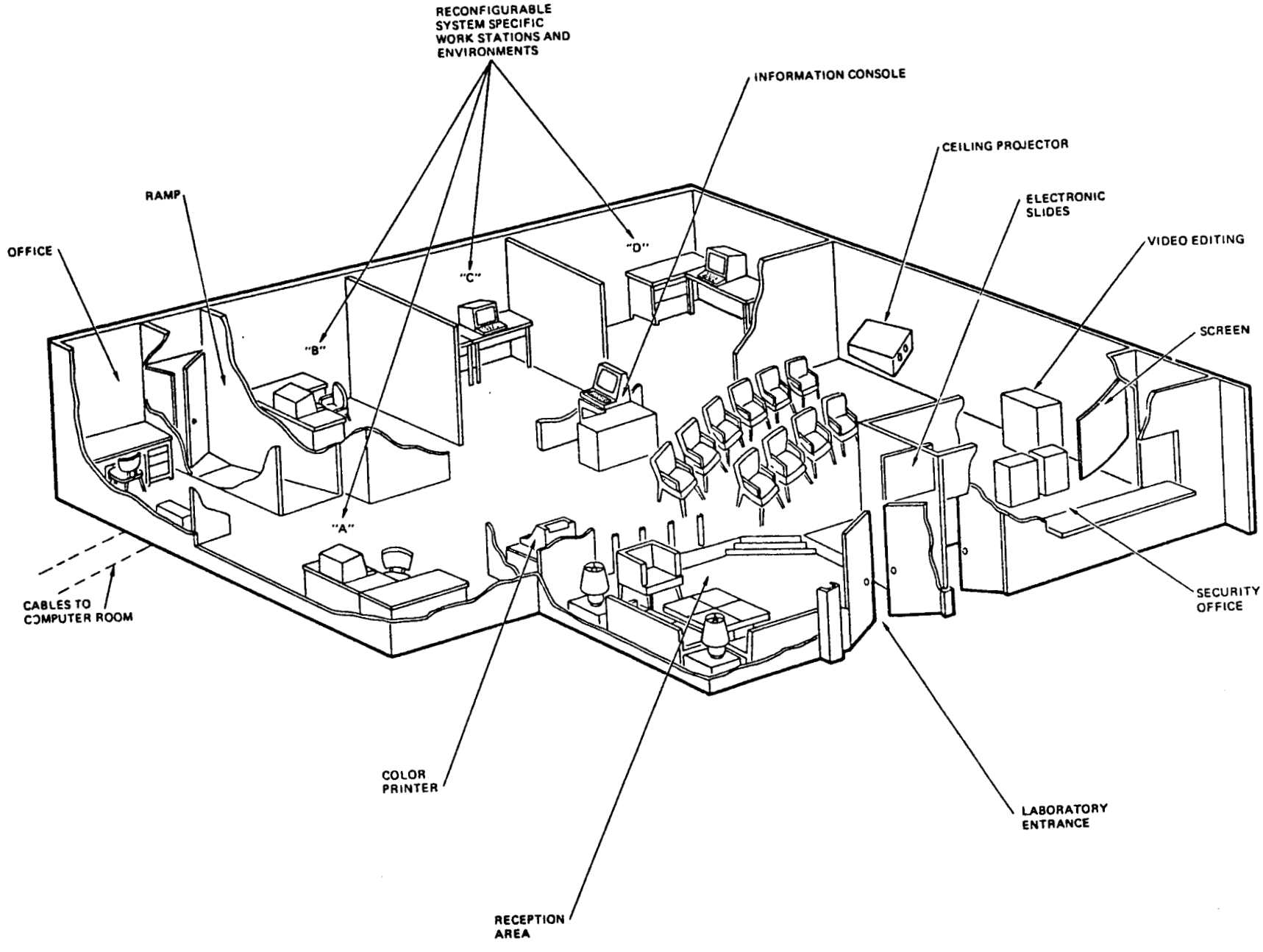


Figure 5. A Typical System Prototyping Laboratory

USER ENGINEERING EXPERIENCES

Name	Function	Customer	When
OBU	Ocean Surveillance	Navy	'82/12mo
SAFE	Intelligence Prod.	Class.	'82/3mo
SOGS	Space Telescope	NASA GOD.	'82/5mo
EXPR	Monoscopic Revision	TRW Res.	'83/1day
SPADOC	Space Threat Anal.	AF ESD	'83/18mo
CPGS	Map Finishing	TRW Res.	'84/.5mo
ASAS	All Source Anal.	Class.	'84/.5mo
BDIP	Bathymetric Integ.	Class.	'84/2mo
OMV	Orbiting Man. Veh.	NASA MAR.	'84/10mo
CMSS	Crisis Management	Class.	'85/4days
CHE	Space Station	NASA JOH.	'85/4mos
RMMS	Remote Maint. Mon.	FAA	'85/5mo
NTB	Space Defense	SDIO	'86/3mo
4700	Sensor Data Proc.	Class.	'86/6mo

Figure 6. User Engineering Applications

APPROACHES TO THE VERIFICATION OF RULE-BASED EXPERT SYSTEMS

Chris Culbert, Gary Riley, Robert T. Savely
Artificial Intelligence Section - FM72
NASA/Johnson Space Center
Houston, TX 77058

ABSTRACT

Expert systems are a highly useful spinoff of the artificial intelligence research efforts. One major stumbling block to extended use of expert systems is the lack of well-defined verification and validation (V&V) methodologies. Since expert systems are computer programs, the definitions of "verification" and "validation" from conventional software are applicable. The primary difficulty with expert systems is the use of development methodologies which don't support effective V&V. If proper techniques are used to document requirements, V&V of rule-based expert systems is possible, and may be easier than with conventional code. For NASA applications, the flight technique panels used in previous programs should provide an excellent way of verifying the rules used in expert systems. There are, however, some inherent differences in expert systems that will affect V&V considerations.

INTRODUCTION

Expert systems are one of the most important spin-offs from the artificial intelligence research efforts. Expert systems have been around for a number of years and some applications have proven highly successful. However, despite their apparent utility and the growing number of applications being developed, not all expert systems reach the point of operational use. One reason for this is the lack of well understood techniques for V&V of expert systems.

Developers of computer software for use in mission or safety critical applications have always relied upon extensive V&V to ensure that safety and/or mission goals were not compromised by software problems. Also, software developers have learned that aggressive V&V used early in the software life cycle can dramatically lower life cycle costs and improve software quality. Expert systems are computer programs, and without V&V they will not be accepted as either safe or cost-effective solutions to problems.

Despite the clear need for V&V, considerable confusion exists over how to accomplish V&V of an expert system. There are even those who question whether or not it can be done. As some authors have suggested (Green and Keyes[1]) this has led to a vicious circle: V&V of expert systems is not done because nobody requires it. Nobody requires V&V of expert systems because nobody knows how it can be accomplished. Nobody knows how to do V&V of expert systems because nobody has done it.

This cycle must be broken for expert system applications to succeed. However, we must first understand what we are talking about when we discuss validation and verification.

DEFINING THE TERMINOLOGY

One basic problem with V&V of expert systems has been the lack of consistent definitions for both validation and verification. Partly because expert systems have their own terminology, there seems to be a tendency to consider expert systems as something more than "just computer programs". Since the development of an expert system uses new concepts such as knowledge engineers, inference engines, and knowledge representation, it

would seem plausible that the meanings of verification and validation may also have changed. However, this is not true.

At the user level, an expert system *is* 'just a computer program' and this is the level that effective V&V must address. Therefore, it is appropriate to use the definitions for verification and validation that apply to conventional software. The following definitions come from the IEEE Standard Glossary of Software Engineering Terminology[2]:

Verification. The process of determining whether or not the products of a given phase of software development meet all the requirements established during the previous phase.

Validation. The process of evaluating software at the end of the development process to ensure compliance with software requirements.

Boehm[3] suggests more informal definitions might be:

Verification. "Am I building the product right?"

Validation. "Am I building the right product?"

When put in this framework, it is clear that expert systems should be both verifiable and 'validatable' in the conventional sense. If one accepts that V&V of expert systems can be done, the next question is how it should be done. As with conventional software, the key to V&V lies in the development methodology.

THE COMMON APPROACH TO DEVELOPING EXPERT SYSTEMS

Most existing expert systems are based upon relatively new software techniques which were developed to describe human heuristics and to provide a better model of complex systems. In expert system terminology, these techniques are called knowledge representation. Although numerous knowledge representation techniques are currently in use (rules, objects, frames, etc) they all share some common characteristics. One shared characteristic is the ability to provide a very high level of abstraction. Another is the explicit separation of the knowledge which describes how to solve problems from the data which describes the current state of the world.

Each of the available representations have strengths and weaknesses. With the current state-of-the-art, it is not always obvious which representation is best to use in solving a problem. Therefore, most expert system development is done by rapid prototyping. The primary purpose of initial prototype is to demonstrate the feasibility of a particular knowledge representation. It is not unusual for entire prototypes to be discarded if the representation doesn't provide the proper reasoning flexibility.

Another common characteristic of expert system development is that relatively few requirements are initially specified. Typically, a rather vague, very general requirement is suggested, e.g., "We want a program to do just what Charlie does". Development of the expert system starts with an interview during which the knowledge engineer tries to discover both what it is that Charlie does and how he does it. Often there are no requirements written down except the initial goal of "doing what Charlie does". All the remaining system requirements are formulated by the knowledge engineer during development. Sometimes, the eventual users of the system are neither consulted nor even specified until late in the development phase. As with conventional code, failure to consult the intended users early in the development phase results in significant additional costs later in the program.

So where does all this lead? The knowledge engineer is developing one or more prototypes which attempt to demonstrate the knowledge engineer's understanding of Charlie's expertise. However, solid requirements written down in a clear, understandable, *easy to test* manner generally don't exist. This is why most expert systems are difficult to verify and validate; not because they are implicitly different from other computer applications, but because they are commonly developed in a manner which makes them impossible to test!

NEW APPROACHES TO DEVELOPMENT METHODOLOGIES

From the preceding section, it should be clear that the problem is the use of development methodologies which generally do not generate requirements which can be tested. Therefore, the obvious solution is to use a methodology which will produce written requirements which can be referred to throughout development to verify correctness of approach and which can be tested at the end of development to validate the final program.

Unfortunately, it's not that simple. Some expert systems can probably be developed by using conventional software engineering techniques to create software requirements and design specifications at the beginning of the design phase (Bochsler and Goodwin[4]). However, the type of knowledge used in other expert systems doesn't lend itself to this approach. It is best obtained through iterative refinement of a prototype which allows the expert to spot errors in the expert system reasoning before he can clearly specify the correct rules.

Since it would appear that rapid prototyping and iterative development are a necessary part of expert system development, an appropriate model for expert system development might be the spiral model suggested by Boehm[5] and modified by Stachowitz and Combs[6] (Fig. 1). This model allows continued iterative development while still providing documented requirements.

Another approach would be to write most of the requirements and specification documentation after completion of the prototyping phase. In essence, the prototype would form the basis for the requirements and would act as a "living spec". This allows the knowledge engineer to find the most appropriate knowledge representation method and gain a reasonable understanding of the problem. It also requires that coding stop at the end of the design phase so the requirements can be written. This approach is outlined in figure 2 and was developed at a NASA workshop on Verification of Knowledge Based Systems at Ames Research Center in April, 1987.

Figure 1 - Boehm's Spiral Model

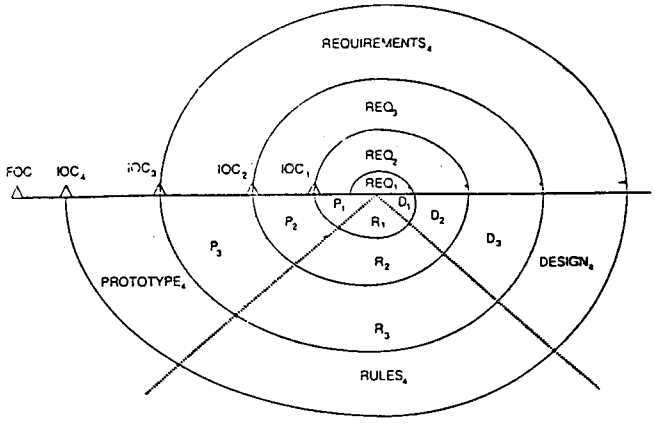
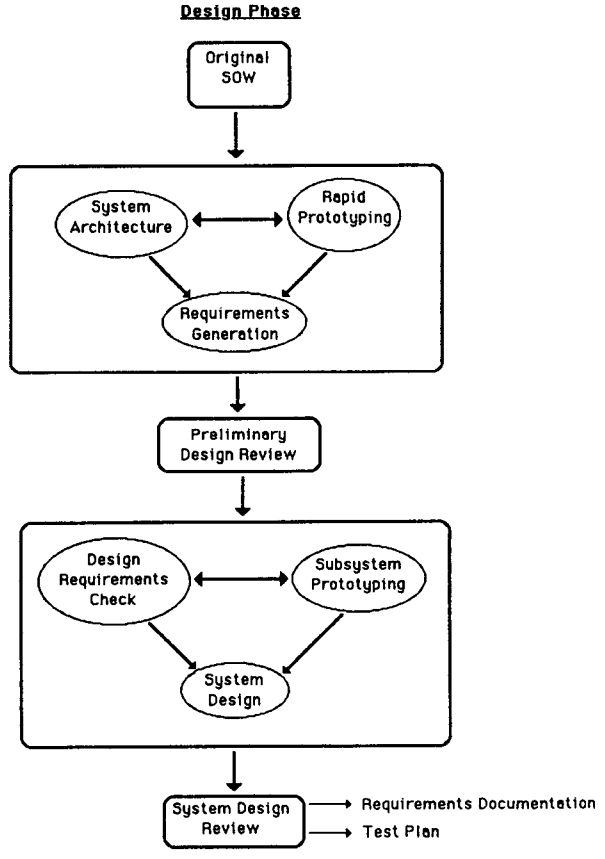


Figure 2 - Expert System Development Methodology



The approach chosen for use will probably depend on the size of the system, the complexity of the knowledge representation, and the eventual application environment. For large systems with many modules or functions it may be difficult to write all the requirements at the completion of a single prototyping phase. The spiral model would be most appropriate in this case. For smaller systems which require few iterations, the second model may be more appropriate.

MAKING THE REQUIREMENTS WORK

Once we accept that requirements and specifications must be written and a methodology for how and when to write them has been adopted, the actual work of verifying and validating the program must be done. There are some general issues which apply to any expert system and some issues that may apply to NASA expert systems in particular.

General Issues

Along with a requirements document a test plan should be written. Most of the criteria used to evaluate conventional software applies. The test plan should describe how the requirements and/or prototype will be checked for completeness, consistency, feasibility, maintainability and testability.

Some of this work can be done automatically. Testing a rule language for completeness and consistency may actually be easier than testing conventional programs. The explicit separation of knowledge elements from control and data elements may allow relatively straightforward analysis of the prototype by automated tools (Stachowitz and Combs[6]). If automated methods are not used, other standard methods such as code reviews and manual examination of the rules may also be comparatively easy, again due to the independent nature of the knowledge elements.

Feasibility of knowledge representation is usually fully tested in the early prototypes, but the feasibility of other elements of the expert system, such as performance, user interfaces, data interfaces, etc. must be verified and validated as well.

Finally, the requirements must be examined to ensure that they are able to be tested. They should be specific, unambiguous and quantitative where possible. Objective requirements will aid in the development of rigorous test cases for final validation.

Issues Specific to NASA Expert Systems

Expert systems applications for NASA programs such as the Space Station will be able to use verification techniques not necessarily applicable outside the NASA environment. These verification techniques are a direct derivative of the methods used to develop procedures, flight rules, and flight software for the Apollo and Shuttle programs. They consist of Flight Technique Panels which regularly review both the procedures for resolving a problem and the analysis techniques used to develop those procedures.

If expertise is not readily available, the analysis efforts typically use high fidelity simulations based on system models to derive and evaluate control parameters. If expertise is available through previous experience, the existing experts are reviewed by the panel and their knowledge placed in the appropriate context. The panels consist of system users, independent domain experts, system developers, and managers to ensure adequate coverage of all areas of concern. In previous programs, the typical output of such a panel was a set of flight rules describing the operational requirements for a system.

Sometimes these flight rules were translated into computer programs (typically as decision trees) and embedded in the onboard or ground computers. An additional verification step was needed to guarantee that the flight rules approved by the panel were properly coded. More often, computer limitations in the Space Shuttle caused the flight rules to remain in document form used directly by flight controllers and mission crews.

For future programs, many of the flight rules which come from the Flight Technique Panels can be coded directly into expert systems. Expert systems developed in this manner will have undergone extensive verification through the panel review. They should also prove easier to verify in code form because the rule language will allow the program to closely resemble the original flight rule.

Expert system applications outside of NASA could use this same "panel" approach. The disadvantage of using the approach discussed here is the relatively high cost of development and the need for extensive simulation capability to define unknown system characteristics. Programs of the complexity and size with which NASA regularly deals make this approach mandatory. Smaller programs may not be able to afford the resources or effort involved in verifying a system to this extent, but the size of the panel and the length of the review process can be scaled down to something appropriate for the complexity and size of the application. For some applications, the panel approach could look very similar to independent code review techniques.

Exhaustive testing through simulation remains the most effective method available for final validation. However, for any system of reasonable complexity, exhaustive testing is both prohibitively expensive and time consuming. Space Shuttle applications typically used extensive testing with data sets representative of the anticipated problems or failure modes. This method is not guaranteed to eliminate all software bugs, but it can prevent the *anticipated* problems. If used properly, representative testing can eliminate enough problems to make the software acceptable for mission and safety critical applications.

OTHER ISSUES FOR EXPERT SYSTEM V&V

So far, this paper has essentially ignored the differences between conventional software and expert systems. There are differences between these two types of software, and those differences will affect V&V efforts. Some of the differences are discussed in the following.

Verifying the Correctness of Reasoning

Verifying that an expert system solves a problem for the right reasons is sometimes as important as getting the right answer. This is particularly important for rule-based expert systems since each rule is essentially an independent module. In sequential programs, order of calculation is very easy to control and the possible paths through the program to a given solution can often be identified.

By comparison, a rule-based expert system fires rules opportunisticly and the number of potential rule combinations which lead to a solution can be combinatorially high. In such a language, identifying all possible paths to a solution is very difficult. Therefore, it is important to ensure that the expert system has gotten the right answer for the right reasons. This can be accomplished through explanations provided by the expert system or through tracing of the rule logic during execution.

Verifying the Inference Engine

The inference engine in a rule-based expert systems is a completely separate piece of code from the knowledge base. This portion of the program has rigid requirements that can be outlined and tested independently from the rest of the expert system. Often, it can be verified once and then used for multiple expert systems.

Verifying the Expert

An issue that is often raised with expert systems is how to certify the expert whose knowledge is used as the base of an expert system. For expert systems developed using the flight techniques panel method, the standard review process of the panel will ensure correctness of the experts approach in the final rules.

For expert systems developed in other manners, the question is automatically resolved as long as the expert system is validated. The entire purpose of validation is to ensure that the expert system meets all original requirements, including correctness of solution. If the expert system fails to meet these requirements, then one of three things has happened; the knowledge engineer has incorrectly coded the expert's knowledge, the expert has incorrectly described how he arrives at a solution (or does not understand it himself), or the expert's method of determining the solution is incorrect (in which case he probably isn't really an expert!). Any of these problems will be detected by the validation process and hopefully corrected.

Real-Time Performance

Expert systems which must provide guaranteed performance in real-time environments

are another area that has been questioned. Most conventional programs provide performance "guarantees" through extensive simulation of the expected performance environment. Expert systems can provide the same kind of performance "guarantees". It might be more appropriate to regard these "guarantees" as upper limits which will not be exceeded for any permitted inputs.

Less often, some kinds of conventional programs are analyzed at the machine instruction level to specifically determine the amount of time required to process a given data set. Achieving the same kind of capability in a rule-based expert system is more difficult. Examining a rule-language at the machine instruction level would be both laborious and time consuming. However, as with conventional code, it can be done for a given data set entered in a specific sequence.

Complex Problems with Multiple Experts

Although the majority of the expert systems currently being developed use expertise from a single, restricted domain, it is likely that expert systems will be developed which combine the expertise of multiple experts from multiple domains. This could lead to systems which produce answers beyond the capability of any one person to evaluate.

The panel review method already discussed for NASA applications is clearly the appropriate method for resolving a problem of this type. The review process used by the panel will allow inputs from any number of domain experts and will also establish the methods of validating system responses.

Traceability of Requirements

A key part of verification is the process of tracing each module or functional element of a program back to the requirements. This process helps guarantee that the program will solve the basic problem and have the desired characteristics. It also prevents unnecessary code or features.

However, tracing requirements after they have been coded in rules may be more difficult than for conventional code. Some requirements may require multiple rule firings or the interaction of many elements in the program to achieve the desired result. Some rules may be general in nature and therefore support multiple requirements. Specifically identifying which rules support which requirements may be difficult.

This problem can become even more difficult when hybrid representation techniques are used, i.e. when both rules and objects are used to satisfy the program's requirements. Tracing requirements through a combination of representation schemes could conceivably be very difficult. Clearly, this is an area that needs some work. The complexity of this issue may even preclude the use of hybrid tools in critical applications.

Verifying the Boundaries of the Expert System Domain

A problem common to most expert systems is the brittleness of the system near the boundaries of the problem domain. It is not difficult to design an expert system which recognizes when a problem is completely outside the bounds of it's domain. It is more difficult to develop expert systems which are able to handle problems which are right at the boundaries of it's domain. That is, problems which the expert system partially recognizes, but does not have all the information needed to solve. For safety or mission critical applications, the expert system must fail gracefully (e.g., fail-safe).

Verifying that the expert system handles such situations properly could be difficult. The boundaries of a problem domain are often somewhat fuzzy. Problems which fall on the boundaries may be best recognized during testing rather than identified early in development. V&V of an expert system must be carefully aimed at identifying these boundaries if the experts can not readily do so. V&V must also ensure that the expert system fails gracefully in these circumstances.

CONCLUSIONS

Verification and validation of expert systems is very important for the future success of this technology. Software will never be used in non-trivial applications unless the program developers can assure the users/managers that the software is reliable and generally free from error. Therefore V&V of expert systems

must be done. Although there are issues inherent to expert systems which introduce new complexities to the process, verification and validation can be done. The primary hindrance to effective V&V is the use of methodologies which do not produce traceable, testable requirements. Without requirements, V&V are meaningless concepts. For NASA applications, an extension of the flight technique panels used in previous programs should provide very high levels of verification for expert systems.

REFERENCES

1. Green, C. and Keyes, M., "Verification and Validation of Expert Systems", Workshop on Knowledge Based System Verification, NASA/Ames Research Center, Mountain View, CA., April 15-17, 1987.

2. IEEE Standard Glossary for Software Engineering Terminology, IEEE Std. 729-1983, Los Alamitos, CA., 1983.

3. Boehm, B.W., "Verifying and Validating Software Requirements and Design Specifications", IEEE SOFTWARE JOURNAL, January 1984.

4. Bochsler, D.C. and Goodwin, M.A., "Software Engineering Techniques Used to Develop an Expert System for Automated Space Vehicle Rendezvous", Proceeding of the Second Annual Workshop on Robotics and Expert Systems, Instrument Society of America, Research Triangle Park, NC., June 1986,

5. Boehm, B.W., " A Spiral Model of Software Development and Enhancement". ACM Software Engineering Notes, March 1986.

6. Stachowitz, R.A. and Combs, J.B., "Validation of Expert Systems", Proceedings Hawaii International Conference on Systems Sciences, Kona, Hawaii, January 6-9, 1987.

A Formal Approach to Validation and Verification for Knowledge-Based Control Systems

Glen Castore
Honeywell Inc.
Systems and Research Center
Minneapolis, MN.

Abstract

As control systems become more complex, in response to desires for greater system flexibility, performance, and reliability the promise is held out that artificial intelligence might provide the means for building such systems. An obstacle to the use of symbolic processing constructs in this domain is the need for verification and validation of the system. Techniques currently in use do not seem appropriate for knowledge-based software. An outline of a formal approach to V&V for knowledge-based control systems is presented in this paper.

1 Introduction

Knowledge-based systems have been applied in areas as diverse as medical diagnosis, machine tool programming, and VLSI design. Such applications have the common characteristic that the recommendations of the expert system can be dealt with in a fairly relaxed manner. A doctor reviews the diagnosis made by an expert system to see if it is sensible. If there is some question about it, the diagnosis can be ignored or the system can be queried as to the basis for the analysis. Time pressure is not severe and control of the situation, in particular control of the use of the output of the expert system, remains in human hands. With many of these systems problems with the implementation or design can be detected while the software is in use, be fixed, and the expert system is still be considered sufficiently reliable to be useful.

This casual mode of operation is unacceptable when the knowledge-based system is operating as part of autonomous or semi-autonomous units such as machine tool controllers, robots, the space station life support module, or an aircraft flight control system. In such applications it becomes essential to have a precise language for specifying what the knowledge-based system should do, and to have an effective procedure for insuring that a particular implementation does meet these requirements. This is the goal of validation and verification (V&V) procedures.

While there are no standard definitions for verification or validation there is a general understanding that *verification* addresses the issue of whether the program specification accurately reflects the functions to be performed while *validation* addresses the question of whether the specifications are correctly implemented. The ideas are

summarized in the phrases:

"Is the correct program being built?" - (verification)
"Is the program built correctly?" - (validation)

Verification is often largely a manual process. Specifications are read, cross-referenced, and checked for consistency and completeness. The quality of this work is heavily dependent on the environment available for developing and tracking specifications and requirements. Validation, on the other hand, has traditionally involved a large amount of testing, simulation and, to a much lesser extent, methods based in formal logic for establishing properties of a program.

For most knowledge-based systems, however, validation through testing and simulation is inappropriate. There are two dominant reasons for this. First, knowledge-based systems are usually most appropriately modeled as nondeterministic automata. A characteristic of nondeterministic machines is that identical inputs to the machine (in identical states) do not necessarily yield identical outputs. The basis for verification by simulation crumbles. Secondly, expert systems, a subclass of knowledge-based systems, are not always expected to give the right answer, just as experts do not always give the right answer. Thus the notion of program correctness cannot always be formulated in terms of input-output behavior, which is the assumption behind testing and simulation as well as some formal methods. While neither of these properties is unique to knowledge-based software, they are much more prominant than in, for example, operating system software. Moreover, these are not characteristics often found in software which must meet rigorous V&V criteria. Control logic for aircraft control systems, for example, is often designed explicitly in terms of finite state machines. Thus it is much more amenable to validation through testing and simulation.

In this paper a formal model is proposed for V&V for knowledge-based control systems. *Formal* means based in mathematical logic. *Knowledge-based control systems* (KBCS) are control systems in which symbolic processing methods are tightly coupled to standard control algorithms. The approach taken is to formulate a structural model for the KBCS. This model can be viewed as a representation of the nondeterministic automaton mentioned above. A logic is then developed for asserting and reasoning about properties of such structures. Specifications are interpreted as assertions about properties of the model. The role of the validation software is to prove these assertions.

2 V&V for KBCS

Within the domain of real time control the anticipated problems of V&V for knowledge-based systems are compounded by the real time aspects of the domain. These difficulties are ameliorated somewhat by restricting the domain of application to systems built in conformity with a prescribed model for KBCSs. This model applies to a class of control systems for which it appears that the use of knowledge-based methods can contribute to system performance and fault tolerance. It seems that such domain models may be necessary to reduce the computational complexity of the formal V&V methods to tractable proportions.

2.1 V&V Issues and AI

There are a number of aspects of artificial intelligence programs which appear to complicate the task of V&V.

1. There is often a strong nondeterministic flavor to A.I. programs.
2. Time of execution for inference algorithms can be extremely data dependent.

3. Interrupt handling is difficult and unreliable. There are no standard interfaces to other components of the system and no well defined methods for resuming an interrupted inference process.

4. Languages typically used for A.I. are usually weakly typed.

These are not unique characteristics of symbolic processing programs. It is the conjunction of these properties within A.I. programs, together with the conceptual complexity of the programs, which creates difficulties when attempting to base V&V procedures upon formal logic.

2.2 Issues Raised by Real Time Applications

The phrase *real time*, when applied to computer programs, is generally used to invoke images of dire consequences of failure and dismally restrictive time constraints on program execution. This view is not altogether untrue, but is is perhaps too imprecise. In the context of developing knowledge based control systems four aspects of real time performance seem to dominate design and implementation decisions.

1. *Time constraints on system performance, and thus implicitly on software execution.* The software must be viewed in the context of the entire system. Constraints on software performance result from percolating system requirements through an architecture. Inadequate software performance can be indicative of an inappropriate architecture, as well as an inadequate implementation of the software itself.

2. *Actions have consequences and the penalty for not meeting requirements can be severe.* These consequences may be economic, such as ruining a batch of toilet paper, or they may lead to injury or loss of life.

3. *The timing of events is determined by the system environment, not by the programmer.* As with performance requirements, these constraints can result from choices concerning the hardware and communication's architecture as well as the original system requirements.

4. *Demands on the system may occur in parallel rather than sequentially.* Contention for resources will occur in patterns that the programmer has not anticipated.

The real issue here is not some mythical intrinsic sluggishness of knowledge-based systems. In fact performance, in the sense of speed of execution, is often adequate for embedded control applications. The issue is adding constructs to the base language which enable the system developer to incorporate timing and sequencing constraints, for example, within the KBCS without sacrificing clarity and abstraction.

3 Knowledge-Based Control Systems

Verification and validation of the implementation software is a standard requirement for many control systems. Consequently the successful incorporation of constructs from artificial intelligence within the framework of control theory requires that there be a method for V&V of knowledge-based control systems. If methods based in formal logic are to be used as the foundation for V&V in this domain, it is necessary to be able to describe, in a precise way, what constitutes a well-formed knowledge-based control system.

Traditional control theory deals with systems which can be described in terms of a state vector, $(x_1(t), \ldots, x_n(t))$ where the $x_i(t)$ are usually reasonable

real-valued functions. The time evolution of the state is governed by differential, difference, or integral equations. Within this framework methods have been developed which enable designers to address questions of stability, coverage, reliability, and performance among other things. Control systems for a wide range of devices, from toasters to airplanes, have been built using these theories. There are problems, however, which fall naturally into the category of control but for which these methods appear to be inadequate [5]. Systems in which the state space description involves discrete, and perhaps nonnumeric, variables fall into this category. We such systems *hybrid*. Hybrid systems arise when there is mode selection, when switches or limiters are used, or when extensive fault management techniques are required. In such cases the "mode switching logic", or the "fault management logic", which constitutes the discrete aspect of the control system, is usually constructed in a fairly ad hoc manner.

The theory of knowledge-based control systems is meant to be an extension of traditional control theory which will enable integration of symbolic processing methods with standard approaches to control, while retaining the ability to rigorously address questions of stability, performance, and reliability. The model which has been developed is based largely upon work by Wonham and Ramadge [6] and will be described in detail in a forthcoming paper. *The value of such a formal domain model, from the perspective of V&V is that it enables a formal specification language to be built.* The language is complete in the sense that it completely describes this family of control systems, and statements in the specification language can be readily translated to assertions in a modal logic about the structure of the implemented KBCS.

If the modes of a control system are thought of as discrete entities defining the domain of applicability of some control law for a system, then the core of the KBCS is the *mode switching logic* which is generated by the *mode switching supervisor*. The mode switching logic (MSL) is a state-transition graph decsribing which mode transitions are enables. The MSL is generated by the mode switching supervisor (MSS), in accordance with the constraints of the control system design. The primary symbolic processing capability of the system of a KBCS is resident within the MSS. A control system may have several MSS, for example at each level of a hierarchy.

Thus a typical requirement for a control system is that the MSS always generates a finite state machine MSL. This is a statement in the specification language which becomes an assertion to be proved about the implementation of the KBCS. Another requirement might be that the MSL contain no infinite loops. That is, a control decision is always reached in every situation.

4 The Approach

4.1 Overview

The approach taken was to develop a model based upon modal logic which encompassed the control structure and the semantic content of the KBCS. Statements in the specification language could then be interpreted as assertions to be proved about the formal model. The intent is that the specification be developed in parallel with the KBCS and is refined while the system is being built. The environment in which the KBCS is built is based upon an expert system shell, RTBA (for Real Time Blackboard Architecture), developed at Honeywell S&RC. The developer never has to deal directly with the representation used for V&V purposes.

4.2 Representation of the KBCS

The KBCS is viewed as a *family* of graphs. A given graph in the family, corresponds to a "run" of the KBCS. It is built by tying together a number of graphs representing the knowledge and inferences used, much as a truth maintenance system builds a dependency graph. The graphs are a form of predicate transition net [2].

The "tying together" is done through a control graph which models the decision points and the knowledge sources of the system. In RTBA, in its current form, control of invocation of knowledge sources and oracles is represented separately from the domain knowledge sources. *The approach to V&V presented here presumes that such a separation can be made*, although it need not be done as explicitly as in RTBA.

Starting from the control graph of the KBCS form the path space of this graph consisting of all paths based at START. There is an infinite number of these paths. Let C denote the control graph and P(C) the path space of C, where it is understood that "path" means "path based at START". Each path can be expanded into a form of data flow graph representing the types of information and rules which are actually active when the control procedure follows the given path. These data flow graphs, each of which is a member of P(C), are predicate transition nets in the sense Genrich and Lautenbach [2].

A form of modal logic was developed for making statements about, and establishing properties of P(C), this very large family of graphs. The semantic content of the system is represented by interpreting the family of graphs as a Kripke model of the modal logic.

4.3 A Logic for Reasoning about KBCS

We have adapted a form of modal logic, called *computational tree logic* [1] to support making statements about, and proving properties of, this family of graphs. The use of modal logics as a basis for formal verification methods has been proposed by Hoare, Pratt, and a number of other researchers. This work builds upon their efforts.

The operators in this logic are built to form statements about properties of graphs. Examples of modal operators are: *A* meaning *for all paths*; *E* meaning *for some path*; and *X* meaning *nexttime*.

In the world of formal logic this family of predicate transition nets can be treated as a Kripke model of this modal logic. A Kripke model, is a triple **(G, R, INF)** where *G* is a set, *R* is a relation on the set, and *INF* is a set of inference rules. Intuitively *G* can be viewed as a set of possible worlds. it R tells which worlds are accessible from a given world. *INF* tells how true statements in a world are related to true statements in worlds accessible to that world. In our case, an element of *G* is a path in the control graph. *R* is subset inclusion of paths. *INF* is the set of inference rules for the logical operators described above. This model-theoretic interpretation provides a way to deal explicitly with the semantic content of the expert system. The realization of the model in term of graphs means that many of the computations of interest become linear algebra calculations.

5 Other Issues

If validation and verification are concerns when building a system, it is prudent to consider them when building the environment within which the system will be built. For knowledge-based systems this means having well defined methods for knowledge acquisition, including tools for checking the consistency and completeness of information. There also needs to be a formal language for expressing specifications, which supports refinement and explanation, much in the spirit of

Swartout's work. Without this type of support formal methods have little chance of success in practical terms.

It is well to keep in mind that there are often different levels of software criticality in a system. For example, subsystems of a flight control systems might be classified as; life critical, system critical, or mission critical. The level of V&V appropriate for a subsystem is governed in large part by the criticality level of that subsystem. A weakness of the approach to V&V outlined in this paper is that is does not incorporate a mechanism for tailoring the degree of rigor of V&V procedures to the level of criticality of a the knowledge-based system.

Intertwined with method for V&V are questions about software safety and reliability. The goal of V&V has been to insure that software is reliable in that the implementation meets the specifications and is reasonably free of errors. However techiques for achieving reliability and safety in software are sometimes at odds with the requirements for testing. It can be difficult to test software which has been written to mask faults. It is possible that formal methods for V&V offer a solution to this impasse.

It is possible that V&V may actually become easier for knowledge-based systems than for traditional software. As more capability is moved into compilers through the use of program transformation methods, the specifications move closer to becoming the program. Much of the work of validation may then become a one-time effort of insuring the quality of the compiler.

References

1. Emerson, E.A., and A.P. Sistala, "Deciding Full Branching Time Logic", **Information and Control**, vol. 61, 1984, (175-201).

2. Genrich, H.J., and K. Lautenbach, "System Modelling with High-Level Petri Nets", **Theoretical Computer Science**, vol. 13, 1981, (109-136).

3. Rang, E.R. and N.C. Wood, **The Use of Logic Languages for the Formal Specification of Flight Control Systems**, Honeywell SRC, contract P.O. No. RC 52571 prepared for Lockheed-Georgia Company, April 1986.

4. Cook, Thomas M., "KBS Validation and Verification: Issues for the NASA's Space Station Program", talk given at a conference on validation and verification of KBSs at NASA/AMES, May 14-15, 1987.

5. Levis at al., "Challenges to Control", **IEEE Trans. on Automatic Control**, vol. AC-32, no. 4, April 1987.

6. Ramadge, P.J., and W.M. Wonham, "Supervisory Control of a Class of Discrete Event Processes", **SIAM Journal on Control and Optimization**, Jan. 1987. (206-230).

EXPERT SYSTEM VERIFICATION CONCERNS IN AN OPERATIONS ENVIRONMENT

Mary Ann Goodwin and Charles C. Robertson
Rockwell Shuttle Operations Company
600 Gemini
Houston, Texas 77058-2777

ABSTRACT

The Space Shuttle community is currently developing a number of knowledge-based tools, primarily expert systems, to support Space Shuttle operations. This effort is based on the wide-spread realization of the potential benefits of these tools for premission flight planning and real-time flight support. Evolution of these tools into the operations environment is just beginning.

It is proposed that anticipating and responding to the requirements of the operations environment will contribute to a rapid and smooth transition of expert systems from development to operations, and that the requirements for verification are critical to this transition.

This paper identifies the verification requirements of expert systems to be used for flight planning and support and compares them to those of existing procedural software used for flight planning and support. It then explores software engineering concepts and methodology that can be used to satisfy these requirements, to aid transition from development to operations and to support the operations environment during the lifetime of expert systems. Many of these are similar to those used for procedural software.

INTRODUCTION

The range and diversity of specialties and subspecialties required to support Space Shuttle operations develop an enormous amount of the type of skill recently designated "expertise". Expert systems to support flight operations appear to offer significant potential benefits to flight design and dynamics, such as:

- Reducing the manpower and resources required for flight design and dynamics.
- Reducing the dependency on highly skilled people to intervene periodically in fairly standardized tasks, thus freeing them for new development or nonstandard problems.
- Preventing single-point failures or delays due to the unavailability of skilled "experts"; and, similarly, reserving the "corporate knowledge base" should a skilled person become unavailable.
- Improving the quality of certain decisions which require more factors than a human can comfortably consider at once, but which are no problem for a computerized expert system.

A number of expert systems have been built and others are presently being built so that these benefits can be realized. Many more can be expected as the technology becomes an accepted part of the engineer's problem-solving capability and a larger skill base is available for their implementation.

Most of the existing systems are considered prototypes. However, once in the operations environment, they must satisfy the demands of that environment. Because of their potential for affecting flight design decisions that have broad and sometimes critical implications, engineering confidence in the veracity of their results across their lifetime will be of foremost importance to their successful acceptance and integration into flight operations. Therefore, anticipating and preparing to support verification during the lifetime of the expert system should ensure that their potential is realized as well as reduce the time and the human and computer resources required to maintain them.

Following is a discussion of software requirements in the Shuttle operations environment, what can be considered a verified expert system, historical approaches to aid and accomplish verification of conventional programs, and, last, approaches that can be taken during prototype development to aid verification and ease integration of expert systems into the operations environment.

SHUTTLE OPERATIONS SOFTWARE ENVIRONMENT

Flight design and flight dynamics software have an overlapping set of modeling and performance requirements. Premission conceptual flight design may require fairly simple analytic models, whereas operations flight design will require extremely high fidelity models that run much slower than real-time. Flight dynamics, software must support real-time performance and, at the same time, the best possible performance achievable under that constraint

is desired. Expert systems for flight design and dynamics must meet these same modeling and performance requirements.

Presently, a great deal of effort is being made to streamline and standardize Shuttle flight design and dynamics. For example, procedures for developing flight products and ensuring their quality have been explicitly defined and documented. Techniques are being implemented to track product development, to ensure that the defined procedures are followed and that approved software are used for product generation. Software approved for product generation is being placed under configuration control, and changes must be formally requested, approved, and verified prior to being made available to the flight designers.

When expert systems are used in the generation of flight products and in support of flight dynamics, they will become subject to the same controls. Often the expert system decision-making capability will be embedded in the application software, so that it will simply account for a portion of a larger system that is under configuration control.

Greater use of database technology is planned to manage flight data and ensure its integrity and commonality among products. Electronic storage and retrieval will pass data among the software programs generating the products. Expert systems will also be required to access and store information in these databases.

This environment will require that expert system verification be one component in the verification process of a complex multilanguage software system that includes conventional languages such as Fortran or C, the embedded expert system shell language, and the embedded database query language.

WHAT IS A VERIFIED EXPERT SYSTEM?

If we are to produce "correct" expert systems, we must produce systems that reflect "correct" knowledge, which for practical purposes may be considered the "best" knowledge, judgement, or decision-making capability the expert possesses or can derive. The verification problem as discussed herein is to provide expert systems that reflect this knowledge and will never reflect any contrary results.

Responsibility for the "correctness" of knowledge belongs by definition to the expert. In flight design this is analogous to responsibility for the requirements for conventional flight software belonging to the flight designer. For the expert system, the knowledge must first be correctly acquired from the expert, a responsibility shared by the expert and the "knowledge engineer" or implementer. Then the implementer must reflect exactly what the expert means by creating a rule and fact base in the expert system shell language. This sounds very familiar to those who have generated conventional software systems. The implementation must match the specifications.

Verification of an expert system, then, must verify the adequacy and accuracy of the knowledge base implementation according to specific performance criteria.

LESSON FROM THE PAST (AND PRESENT)

Existing flight design and dynamics software reflect various design and development software engineering methodologies that evolved over the years the software was developed, i.e., the design and development techniques and philosophies vary widely.

Verification is generally considered to be one part of the software engineering process, but the ease with which it can be accomplished has been recognized as being dependent on the techniques and methods used for the preceding development phases. That is, how one does design and implementation affects how one does verification. Approaches that encourage early discovery of errors reduce the time, design impact, and computer and human resources required for corrections.

Verification first occurs prior to the initial delivery of software and it recurs each time the software is modified over its lifetime. For some flight software, the lifetime can be considered essentially unlimited. Modifications of flight design software, whether procedural or expert system, can be expected as

- New flight design requirements occur.
- The knowledge in some area so improves that it is desirable for the flight design software to reflect these improvements.
- Updates are made to the Shuttle hardware or software.
- Significant changes in the state of the art of software and hardware occur that offer desired performance improvements.

It is now recognized that the cost to maintain flight software during its lifetime far exceeds the development cost. Measures taken during development to reduce cost during maintenance are cost effective.

A great deal of research has been done and effort made to develop techniques to support verification of conventional software. These techniques can be placed in two categories: (1) design and implementation techniques that either reduce the likelihood of errors or make them easier to find and correct; and (2) software development support tools that detect and remove errors from the code.

The following techniques are in the first category:

- Project management techniques such as top-down development, design and code reviews, use of program libraries, etc.
- Project design techniques, such as top down design, code modularization, and, more recently,

information hiding, object-oriented design, entity-relationship modeling, data flow diagrams, etc.

- Languages modifications which simplify code and make it easier to understand and debug; e.g., structured code, strong typing.
- Documentation standards, both external and internal to the code.
- Coding standards which not only standardize how code is written but which may also outlaw code considered to be error-prone.

The following techniques are in the second category:

- Development of static code analyzers and dynamic code analyzers. With static code analyzers, the code is parsed, and the parsed information is stored in such a manner that a postprocessor can cross-reference information to detect errors. An example is locating variables used but not set or vice versa. With dynamic code analyzers, the code is "instrumented" with "probes". Special-purpose code is inserted at strategic locations to capture and output data of interest as a routine executes. This output is then postprocessed to provide information to aid verification. For example, it is possible to determine what part of the code is executed and what is not for all test cases in a test case library (See reference 1)
- Improvement of compilers to aid error detection.
- Development of test case libraries that satisfy such criteria considered beneficial to verification as exercising as much of the code as possible and doing "stress testing" to exercise numerically sensitive code.

Development of an automated software development and maintenance support environment for use in all phases of program development, from program design to code generation and program verification, is presently occurring and may greatly impact the update of existing flight software systems and the creation of future systems.

VERIFICATION TECHNIQUES FOR OPERATIONAL EXPERT SYSTEMS

Enter expert systems into the Space Shuttle operations environment. A large flight design simulation could conceivably have "pockets of reasoning" for decision-making at various points in its execution. A decision might be made to determine the type of flight to simulate, the characteristics of the sensors to be simulated, or the environmental models to be invoked. Another decision might be made to output recommendations about the simulation or information about its results that help the flight designer. These types of decisions presently occur at defined points in flight design programs. They simply occur with a man in the loop and sometimes in an off-line mode. Therefore, one might consider them already a part of the software design, and it appears reasonable that the design of an expert system decision-making capability for a flight simulation can be accomplished by extending the conventional system design techniques.

One technique that successfully attempted to do this is documented in references 2 and 3. The well-known hierarchical input process output (HIPO) technique was used to develop requirements, construct the design, and support implementation of an expert system to demonstrate automated rendezvous. Verification was then conducted systematically because of the method of design and implementation.

In the pragmatic Shuttle flight operations climate, where expert system design, development and verification is one part of the design, development, and verification of an existing or emerging software system using conventional and database query languages, it appears that it would be helpful to identify where commonality in verification techniques may be applied and where uniqueness is required in the verification of the overall system.

The interfaces between the symbolic reasoning (or expert system) "modules" and conventional modules or database tables or files can identify type conversions required to go from symbolic facts in the expert system module to digital data or other data types in conventional modules and vice versa. This information should be amenable to data flow diagrams, data dictionaries, or other data/information tracking techniques.

Language improvements can be found as revisions to expert system shells are released. Of great significance for flight design and support is the development of expert system shells that work on conventional hardware and allow the shell language to be embedded with conventional languages. While this does not appear to support verification directly, it allows simpler and more natural interfaces with the rest of the software system which will therefore be less error prone.

A set of preliminary experimental documentation standards and complementary coding standards have been defined (reference 4) for the Automatic Reasoning Rool (ART) developed by Inference, and a subset has been adapted to the Clips shell language developed at the Johnson Space Center (reference 5). The standards have been successfully adapted to a number of expert systems being developed to support flight design (references 6 and 7).

The standards were constructed to support the later development of a maintenance tool. Consequently, they were designed using keywords that could cue a parser to the contents of the various types of comments. Two categories of comments were defined: those to support user explanation of the rules and those to support the programmer in implementation and maintenance. Comments in the first category were intended to be extracted automatically to produce documentation for the users. The standards established were designed to support an automated tool that could generate cross-reference information for rules, patterns, and variables.

The standards are divided into three areas. First, a major file is defined which includes the history of the

expert system and other pertinent information regarding supporting functions and files. All files should be loaded from this major file, which is the program driver. The second area is the declarative knowledge which consists of ART viewpoint information and definitions of relations, facts, and global variables. The third area deals with procedural knowledge, which consist of rules. The commenting template and explanations for this third area are given in figure 1. It has been found that the procedural template can be adapted to the design phase and used for "knowledge reviews". An additional use found for the procedural knowledge template is training support. As the design expands from functional to various levels of detail, the declarative and major file information can be developed as needed.

The authors propose that a relational database management system could be used to perform useful static analysis for error detection of rule-based expert systems and could be implemented independently or in conjunction with the proposed automation of the documentation and coding standards just mentioned. The relational theory allows semantic relations to be conveniently expressed. Some of the simpler relations that could be expressed as relational tables are defined in figure 2. Table 1 lists various relationships that could then be determined by querying the tables. Some of the most aggravating problems that can occur during debugging have to do with simple typing errors that could often be detected by locating occurrences of a unique, one-time-only pattern. Properly constructed data base queries could isolate unique variable names that are likely in error. Further error detection possibilities exist that space does not permit exploring (e.g., see reference 8).

Several other possibilities exist. It is apparent that with sufficient effort the tables in figure 1 could be utilized to automatically construct the expert system shell code, which would be error free. The specific nuances of the languages in expert system shells will undoubtedly introduce aggravating problems in the implementation of the above, but the goals seem achievable.

Additionally, it is possible to express similar type of relational information about the conventional code, such as that captured by the system in reference 1. The query language could then verify interfaces across the two languages by querying the applicable information in particular tables for each language.

CONCLUSIONS

As expert systems are integrated into the Shuttle flight design and support software packages, an integrated project management and software development and maintenance plan that encompasses conventional procedural languages, the expert system shell language, and database query languages is needed so that verification can be accomplished at a minimum cost and results in the highest confidence in the software system over the life cycle of the flight software.

This goal seem achievable. Design and development methods and coding and documentation standards based on those used for procedural code have been applied to expert system prototypes with good results. Additionally, verification tools for expert systems similar to those for procedural code but relying on database systems to simplify implementation appear conceptually to be beneficial and extendable to include conventional code. The latter method could possibly be extended to produce expert system shell code automatically.

The ultimate validity of the expert system reasoning, however, lies with the expert. No amount of programmer effort can improve the judgement or reasoning communicated to the programmer by the expert.

It is recommended that seriously developing and refining these methods as part of prototype development will contribute greatly to a smooth transition of expert system programs from the development to the Shuttle operations environment.

ACKNOWLEDGEMENTS

The authors wish to acknowledge the community of individuals from the Johnson Space Center Mission Planning and Analysis Division, Computer Science Corporation, LinCom Corporation, McDonnell Douglas Corporation, and Rockwell Shuttle Operations Company, who contributed to the documentation standards of reference 4. Grace Hua and Ann S. Baker of CSC must be singled out for their special contributions.

REFERENCES

1. Software Development and Maintenance Aids Catalog, JSC-22342. Dennis Braley, Mission Planning and Analysis Division, JSC, October, 1986.

2. Software Engineering Techniques Used to Develop an Expert System for Automating Space Vehicle Rendezvous, Daniel C. Bochsler, LinCom Corporation, and Mary Ann Goodwin, Johnson Space Center, Second Annual Workshop on Robotics and Expert Systems, Johnson Space Center, June, 1986.

3. A Software Engineering Approach to Expert System Design and Verification, David C. Bochsler, LinCom Corporation, and Mary Ann Goodwin, Johnson Space Center, Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, November, 1986.

4. Documentation Standards for Expert Systems, JSC Memorandum FM7/86-187, Robert H. Brown, October 22, 1986.

5. CLIPS Reference Manual, JSC-22552,Christopher J. Culbert, Mission Planning and Analysis Division, Johnson Space Center, March, 1987.

6. Orbital Navigation Expert System (ONEX); McDonnell Douglas Memorandum TM1.2-TM-FM87044-022; A. J. Reich/T5B; December 29, 1986.

7. Guidelines and System Requirements for the Onboard Navigation (ONAV) Console Expert/Trainer System, JSC-22433; Artificial Intelligence Section, Mission Planning and Analysis Division, JSC, December, 1986.

8. Knowledge Base Verification; Tin A. Nguyen, Walton A. Perkins, Thomas J. Laffey, Deanne Pecora; AI Magazine V.8, N.2, Summer, 1987.

```
;---------------------------------------------------------------------------
;;; GROUP
;;  <group-name>
;;  <narration on purpose, description of control, objective,
    assumptions, etc.
;;; HISTORY AND RESPONSIBILITY: GENERAL
    (contains information common to all rules in the group)
;;  Name of programmer(s):   <name>
;;  Name of expert(s):       <name>
;;  Created on:              <date>
;;; CONTROL FACTS
    (those unique facts inherited from a parent group)
;   <fact>
;;; PARENTS
    (Used for cross-referencing)
;;  <parent group-name>
;---------------------------------------------------------------------------

(defrule      <rule-name>
;;  If
;;  <english sentence definition of the rule conditions>
;;  Then
;;  <english sentence definition of the rule actions>
;;  End
          .
          .
    <actual rule body>  ;  simple programmer code comment
          .
          .
)
;;; HISTORY AND RESPONSIBILITY: EXCEPTIONS AND UPDATES
;;  Name of programmer(S):   <name>
;;  Name of expert(S):       <name>   (rule-specific exception
                                       to general )
;;  Created on:              <date>
;;  Modified by:             <name>
;;  Modified on:             <date>   (rule-specific update)
;;; RATIONALE
;;  <narrative on heuristics, reasoning, rule-specific assumptions
    and limitations, etc.>
```

Figure 1 - Template for Procedural Knowledge Documentation

A) LEFT HAND SIDE TABLE

RULE NAME	CONDITION SYNTAX	CONDITION CLASSIFICATION
<name>	<condition-1> <condition-2>	< (function, e.g. control, test, pattern, external-function, etc.)>

B) RIGHT HAND SIDE TABLE

RULE NAME	ACTION SYNTAX	ACTION CLASSIFICATION
<name>	<action 1> <action-2>	< (function, e.g. external function, pattern-set, pattern-retract, etc.)>

C) RULE SALIENCE TABLE

RULE NAME	CONDITION CLASSIFICATION
<name>	<number>

Figure 2 - Proposed Expert System Static Analyzer Rule Data Base Format

TABLE I - POSSIBLE USES OF EXPERT SYSTEM STATIC ANALYZER DATA BASE

- Reconstruct rules, including, for example all rules subject to the same condition
- Find patterns set on LHS, but not used on RHS and vice versa
- Determine effect of new rules or rule changes on rule base
  - If new rule sets or retracts a pattern, other rules that use that pattern can be identified
  - Rule dependencies can be identified.
- Find all rules with same salience, or ordered by increasing or decreasing salience
- Find all rules with same control pattern
- Find all external function calls and rules which made call
- Construct English description of rules. (Requires tables not shown in Figure 2.)

Building Validation Tools For Knowledge-Based Systems

R.A. Stachowitz, C.L. Chang, T.S. Stock and J.B. Combs
Lockheed Missiles & Space Company, Inc.
Lockheed Artificial Intelligence Center, O/90-06, B/30E
2100 East St. Elmo Road
Austin, Texas 78744

Abstract

The Expert Systems Validation Associate (EVA), a validation system under development at the Lockheed Artificial Intelligence Center for more than a year, provides a wide range of validation tools to check the correctness, consistency and completeness of a knowledge-based system.

Using a declarative meta-language (higher-order language), we want to create a generic version of EVA to validate applications written in arbitrary expert system shells.

In this paper we outline the architecture and functionality of EVA. The functionality includes Structure Check, Logic Check, Extended Structure Check (using semantic information), Extended Logic Check, Semantic Check, Omission Check, Rule Refinement, Control Check, Test Case Generation, Error Localization, and Behavior Verification.

INTRODUCTION

The growing reliance on knowledge-based systems (KBS's) in industry, business and government requires the development of appropriate methods and tools to validate such systems to ensure their correctness, consistency and completeness. Furthermore, as these systems become operational, an increasing number of knowledge engineers will be involved in their development and maintenance. Hence, insuring the integrity of a particular KBS over its entire life cycle makes the need for automated validation even more crucial.

Early attempts at validating KBS's did not progress beyond basically "syntactic" validation [Nguyen et al. 1985, Nguyen 1987, Reubenstein 1985, Suwa et al. 1982]. Significantly more advanced validation tools are being built at the Lockheed AI Center using semantic information and meta-knowledge for KBS validation [Stachowitz et al. 1987a, 1987b].

The system under development is called the Expert Systems Validation Associate (EVA). Our goal is to create a generic version of EVA which can validate applications written in arbitrary expert system shells, such as ART*, KEE**, OPS5, and others, by mapping an application written in any specific shell into internal data structures in a general and declarative meta-language (higher-order language).

The purpose of EVA is to improve the validation process by finding mistakes and omissions in the knowledge base, by proposing knowledge base extensions and modifications, and by showing the impact of changes to the knowledge base. In other words, EVA addresses not only the question "Is a KBS application correct?", but also the question "Is the knowledge used by the expert for the application correct?"

In this paper, we outline the architecture and functionality of EVA. The functionality includes structure check, logic check, extended structure check (using semantic information), extended logic check, semantic check, omission check, rule refinement, control check, test case generation, error localization, and behavior verification.

EVA ARCHITECTURE

To permit the addition of future functionality EVA is designed to be a *meta*knowledge-based system shell. All the validation modules (checkers) will be built on a unifying, extensible platform. The unifying architecture will be based upon: (a) A single user interface for all checkers; (b) A single meta-knowledge base for all checkers; and (c) A common meta-language for specifying constraints. The advantages of the architecture are uniformity and extensibility.

We intend to implement all the checkers in Quintus Prolog [Quintus 1985]. That is, information about an application represented in a knowledge base, algorithms for the checkers, and domain knowledge will be represented by clauses in Quintus Prolog. Then, invoking a checker is essentially comparable to entering a goal (query) in Quintus Prolog. Our selection of Prolog is based on the following considerations:

(1) The validation of KBS's requires extensive automatic theorem proving facilities such as a unification subroutine. Prolog has a built-in automatic resolution-based theorem prover. By using Prolog we can expedite the development of EVA.

(2) Maintaining the meta-knowledge base requires the functionality of a database management system. Prolog provides a built-in database management system, which makes it unnecessary to develop such functionality separately.

(3) The meta-language required for representing validation criteria as meta-statements can be defined and implemented in Prolog. This makes it unnecessary to develop a separate abstract machine to interpret this meta-language.

EVA FUNCTIONALITY

The functionality of EVA, represented by means of a data flow diagram, is depicted in Figure 1.

An application is written in an object shell, while validation statements for semantic and control constraints, and behavior descriptions are written in a very expressive metashell (meta-language). The metashell will provide many higher-order constructs to support high-level predicates such as *symmetric, nonsymmetric, transitive, nontransitive, reflexive, irreflexive, mandatory, synonymous, compatible, incompatible*, etc.

An analyzer uses conversion algorithms to translate the application and the validation statements into the format to be used by the EVA validation modules; each of which performs a static or dynamic analysis of the application. The analyzer will build a connection graph from facts and rules in the application. An arc in the connection graph denotes a match between a literal in the LHS (Left-Hand Side) of a rule and a literal in the RHS (Right-Hand Side) of a rule. Note that a fact is considered as a rule consisting of a RHS only.

For static analysis, EVA will provide the structure checker, logic checker, extended structure checker, extended logic checker, semantics checker, omission checker, rule refiner and control checker. For dynamic analysis, EVA will provide the test case gen-

*Automated Reasoning Tool, Inference Corporation

**Knowledge Engineering Environment, IntelliCorp

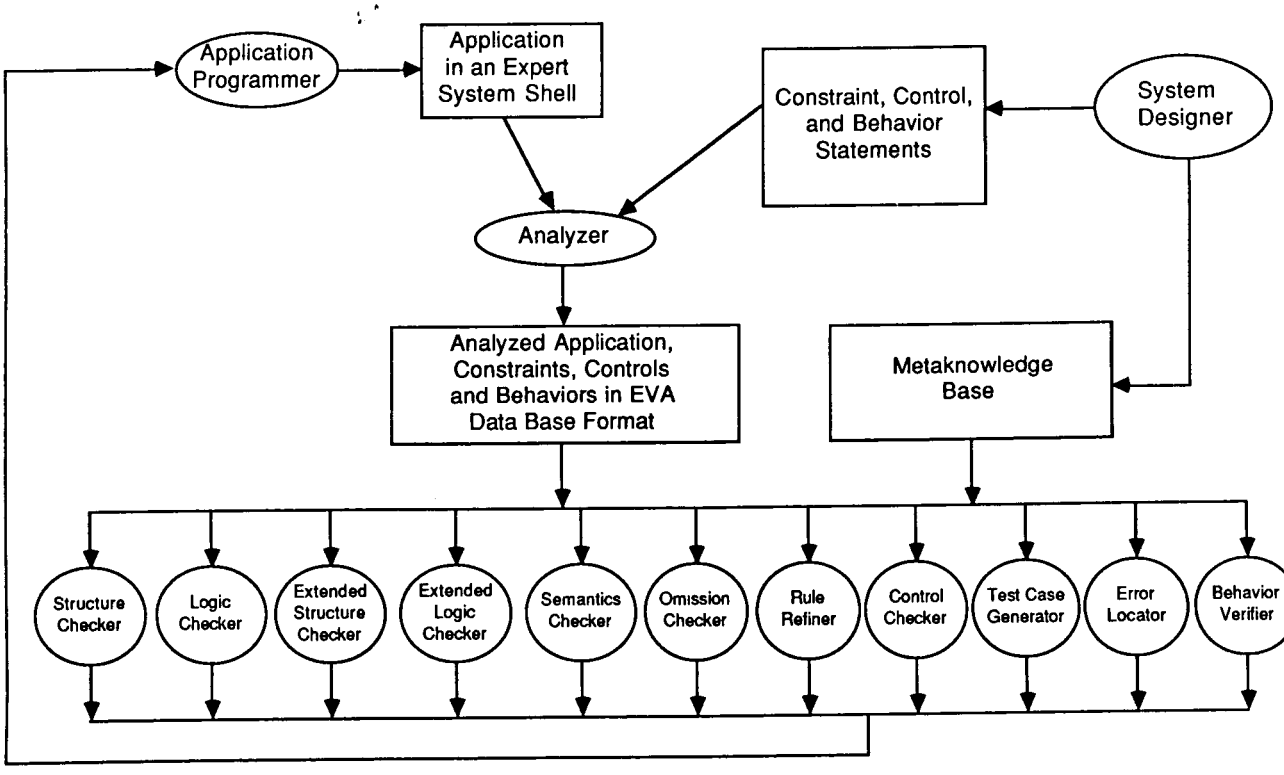


Figure 1. Validation Functionality

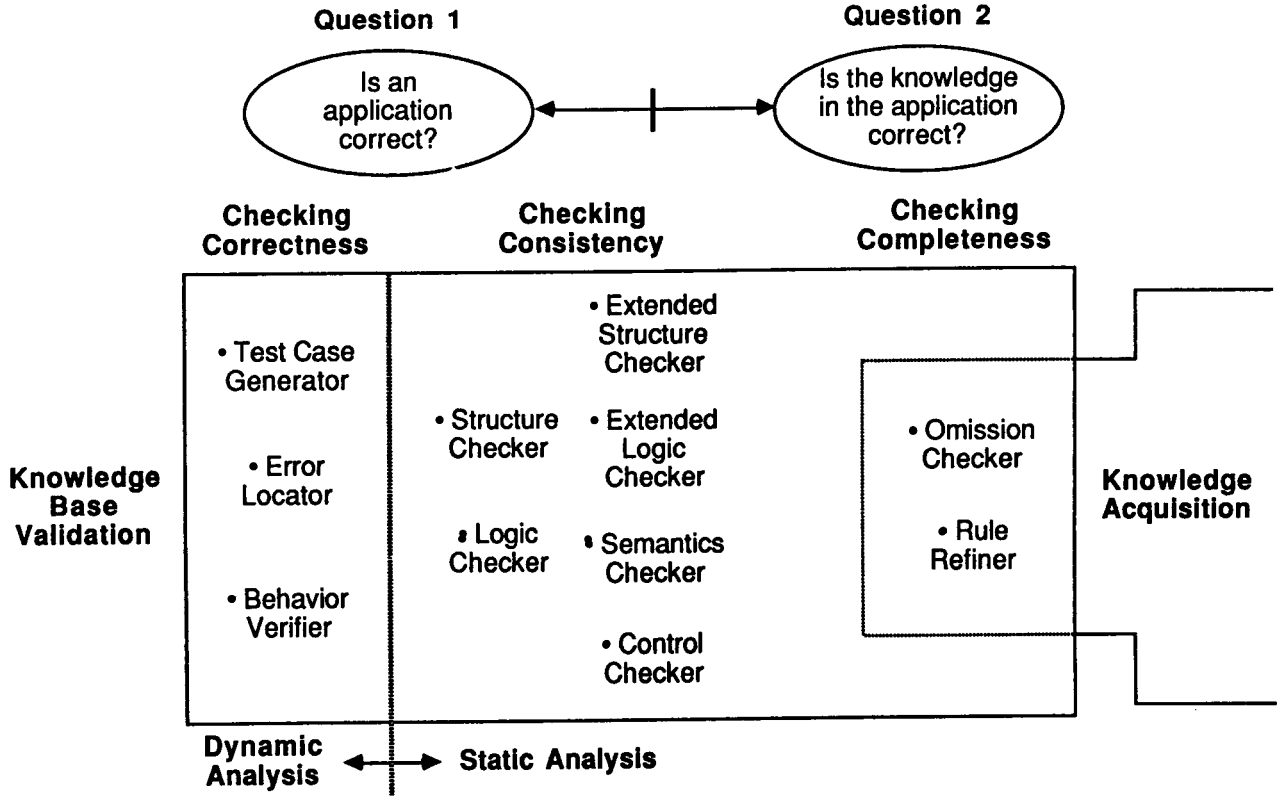


Figure 2. Validation Objectives and Functionality

erator, error locator and behavior verifier. The validation objectives and functionality of EVA are shown in Figure 2.

Using ART and LISP, we have implemented prototypes of the structure checker, logic checker and semantic checker for ART-based expert systems. We have also implemented considerable portions of the structure checker and extended structure checker in Quintus Prolog. Other modules, and more elaborate functions for the structure checker, logic checker and semantics checker are being designed and will be implemented in the future.

We now give detailed descriptions of the EVA modules as follows:

STRUCTURE CHECKER

As discussed before, a knowledge base can be represented by a connection graph. That is, the connection graph basically shows the structure of the knowledge base. The purpose of the structure checker is to detect any anomalies in the connection graph. For example, it will identify rules and facts which will never be used, rules which are superfluous, and rules which possibly lead to an infinite loop.

Reachability

One type of "common" error is the use of an undefined literal in the LHS of a rule. This may occur in top-down and bottom-up specifications. In the top-down approach, the specifier starts with the highest modules, planning to define the lower modules afterwards. However, he may forget them. Therefore, they are left undefined. In the bottom-up approach, the lower modules are defined first. However, when the specifier tries to use them, he may type the module names incorrectly.

A LHS literal may be undefined because either its predicate is not defined or it cannot be unified with any RHS literal. In terms of the connection graph, a literal is undefined if it is not pointed to by any arc.

Another type of related anomaly is the case where RHS literals are not linked to any LHS literals. That is, they are defined but not used. They are called "unreachable" literals. The presence of the unreachable literals may indicate some anomalies, namely, omissions. This is analogous to the experience we have with repairing a car. We may disassemble a carburetor. However, when we put the parts back again, we may find some parts are left unused.

In terms of the connection graph, a RHS literal is unreachable if there is no arc going from it to another literal.

Redundancy

The types of redundancies the structure checker will check are duplications and subsumptions. The duplication and subsumption checks can be done by the same algorithm because duplication is a special case of subsumption.

Given two rules R1 and R2, if the LHS of R1 subsumes the LHS of R2, then whenever R2 can be fired R1 also can be fired. Therefore, any actions in the RHS of R2 which also occur in the RHS of R1 can be eliminated. If after the elimination no actions are left in the RHS of R2, then R2 can be eliminated.

The following are examples of duplication and subsumption:

Duplication:

$male(X) \wedge parent(X) \rightarrow father(X)$.

$parent(Y) \wedge male(Y) \rightarrow father(Y)$.

Subsumption:

$tenured(X) \wedge \neg staff(X) \rightarrow university\_member(X)$.

$tenured(X) \rightarrow university\_member(X)$.

Cycles

In a knowledge base, cycles will occur if recursive rules are used to define predicates. Some cycles are harmless, while others cause problems. EVA will identify "potentially bad" cycles.

First, we consider the case where rules are used to define predicates [Chang 1976, 1978, 1981]. A predicate is called a basic predicate if it is used only for representing facts. A predicate is called a derived predicate if it is defined in terms of basic predicates, or in terms of basic and derived predicates by a rule or a set of rules. For example, the predicate "ancestor" may be defined in terms of the basic predicate "parent" as follows:

$parent(X,Y) \rightarrow ancestor(X,Y)$.

$parent(X,Y) \wedge ancestor(Y,Z) \rightarrow ancestor(X,Z)$.

The first rule is a terminating rule and the second rule is a recursive rule. If the terminating rule is not given by the developer, it will cause the second rule to fire repeatedly, ie., result in an infinite loop. Therefore, if EVA finds that cycles of rules are used to define a derived predicate, it will try to check whether terminating rules are given.

Second, we consider the case where rules are used to establish equivalent predicates. For example, consider the following rules:

$human(X) \rightarrow person(X)$.

$person(X) \rightarrow human(X)$.

These two rules result in a cycle. However, the developer may intend to show that predicates "human" and "person" are equivalent or synonymous. In this case, he should choose one of the predicates as the standard predicate and delete one of the rules.

Finally, we consider the case where a cycle (loop) is used to perform a repeated task in a distributed or non-distributed environment. In this case, cycles are allowed.

EVA will also check if a knowledge base contains overlapped cycles (loops) [Chang 1978]. A knowledge base with overlapped cycles is less modularly structured than one without any overlapped cycles. Since we can always represent a knowledge base without using overlapped cycles, we should enforce this software methodology. This is analogous to structured programming where (since only single-entry and single-exit statements are allowed) loops are well structured in structured programs.

LOGIC CHECKER

The logic checker checks if at some particular time inconsistent or conflicting actions can be triggered by facts in a knowledge base.

For example, consider the rules:

$big(X) \rightarrow expensive(X)$.

$big(Y) \rightarrow \neg expensive(Y)$.

If the fact

$big(truck)$

is in the knowledge base, then the rules will assert the inconsistent facts:

$expensive(truck)$.

$\neg expensive(truck)$.

On the other hand, consider the rules:

$new(X) \wedge big(X) \rightarrow expensive(X)$.

$broken(Y) \wedge big(Y) \rightarrow \neg expensive(Y)$.

Depending upon what set of facts exists at a particular time, these two rules may or may not assert inconsistent facts. For example, if we only have the facts

$new(truck1)$.

$big(truck1)$.

$big(truck2)$.

$broken(truck2)$.

then the rules will assert the facts

$expensive(truck1)$.

$\neg expensive(truck2)$.

which are not inconsistent. However, if we only have the facts

$new(truck1)$.

$big(truck1)$.

$broken(truck1)$.

then we will get inconsistent facts:

$expensive(truck1)$.

$\neg expensive(truck1)$.

The question is whether such a set of facts is ever possible. If the developer thinks it is impossible that an object can be simultaneously new, big and broken, then the inconsistency may not arise. Otherwise, he should be warned. Note that to make sure that a certain set of facts is impossible, the knowledge base will be verified by the semantic checker to be discussed later. That is, the knowledge base will contain "negative" semantic constraints to be used by the semantic checker. For the above example, the negative constraint is represented as

incompatible(new(X), big(X), broken(X)).

Given a set S of rules, the logic checker finds if there exists a set T of facts that may lead the rules in S to generate inconsistent facts. If such a set T exists, the validation system will prompt the developer if T is possible. If he says that T is impossible, a corresponding negative constraint will be added into the knowledge base.

EXTENDED STRUCTURE CHECKER

The extended structure checker checks for reachability, redundancy and cycles caused by generalization hierarchy or synonymy.

Most expert system shells support generalization hierarchy based upon the subclass relationship "isa". For example, if

submarine **isa** ship,

then "submarine" may not be undefined if "ship" is defined, and "ship" may not be unreachable if "submarine" is used in a LHS-literal of a rule.

Similarly, generalization hierarchy can affect the redundancy and cycle properties of a knowledge base. For example, given the rules

submarine(X) → launch(X).

ship(X) → launch(X).

clearly the second rule subsumes the first one.

Now, let us consider synonymy. An expert system may be developed by more than one person. Different persons may use different schemas (names or structures) for the same type of objects. When their knowledge is merged, some standard should be established. EVA will provide the meta-language to map the schemas into a standard schema by using rules. For example, if "sub" and "submarine" are synonymous, and if we choose "submarine" to be the standard name, then the mapping rule is given as

sub(X) → submarine(X).

Mapping rules map a slot or a function of many slots into a standard slot. For example, the slots "year" and "date-of-birth" may be mapped into the slot "age". This mechanism is similar to "view" definitions in a relational data base.

Once the standard schemas are established, the functionality of the extended structure checker for the knowledge base is essentially the same as the functionality of the structure checker for the knowledge base appended by the mapping rules.

EXTENDED LOGIC CHECKER

The purpose of the extended logic checker is to check for inconsistencies and conflicts caused by generalization hierarchy, incompatibility, or synonymy. If the application contains rules that can derive contradictory conclusions from the same set of facts with the properties of generalization hierarchy, incompatibility, or synonymy, then the application is inconsistent.

Inconsistency Under Generalization Hierarchy

Given the metafact that *submarine* is a subclass of *ship*, rules 8 and 9 are inconsistent.

Rule 8: E(X) ∧ F(X) → ¬ship(X).

Rule 9: E(Y) ∧ F(Y) → submarine(Y).

Inconsistency Under Incompatibility

Given the metafact that *boy* and *girl* are incompatible, rules 10 and 11 are inconsistent.

Rule 10: A(X) ∧ B(X) → boy(X).

Rule 11: A(Y) ∧ B(Y) ∧ C(Y) → girl(Y).

Inconsistency Under Synonymy

Given the metafact that *submarine* and *sub* are synonymous, rules 12 and 13 are inconsistent.

Rule 12: E(X) ∧ F(X) → submarine(X).

Rule 13: E(Y) ∧ F(Y) → ¬sub(Y).

In each of the above examples, the conditions in each of the rule pairs are exactly the same or one condition is a proper subset of another. However, the inconsistency can occur in a more general case. The examples below are in **conflict** if **all** of the facts A(e), B(e), C(e), and D(e) are present in the knowledge base **at the same time**.

Conflict Under Generalization Hierarchy

Given the metafact that *submarine* is a subclass of *ship*, rules 14 and 15 are in conflict.

Rule 14: A(X) ∧ B(X) → ¬ship(X).

Rule 15: C(Y) ∧ D(Y) → submarine(Y).

Conflict Under Incompatibility

Given the metafact that *boy* and *girl* are incompatible, rules 16 and 17 are in conflict.

Rule 16: A(X) ∧ B(X) → boy(X).

Rule 17: C(Y) ∧ D(Y) → girl(Y).

Conflict Under Synonymy

Given the metafact that *submarine* and *sub* are synonymous, rules 18 and 19 are in conflict.

Rule 18: A(X) ∧ B(X) → ¬sub(X).

Rule 19: C(Y) ∧ D(Y) → submarine(Y).

If the conditions A(X), B(X), C(X), and D(X) can be satisfied by future facts in the application (recognized from assertions in the RHS of rules), the logic checker warns of *potential conflict*.

The extended logic checker will use an algorithm similar to the one used by the logic checker. The extended logic checker will first choose a goal consisting of a complementary pair of RHS literals or a set of incompatible RHS literals. It will then try to find a set of facts from the goal by performing unifications and substitutions on literals, using rules in the knowledge base as rewriting rules.

EVA provides the meta-predicate "incompatible" for the developer to specify a set of incompatible literals. It has the following structure

incompatible(L1,...,Ln),

where L1,...,Ln are literals for n equal to 1 or more. The meaning of this statement is that the conjunction of L1,...,Ln can not be true at any time. Incompatible statements are interpreted as "negative constraints". In a knowledge base or meta-knowledge base, there are general and domain-specific negative constraints. The following are some examples of incompatibility statements:

(1) incompatible(A, ¬A)

means that an atomic formula A and its negation are incompatible. Note that A may contain a first-order or higher-order predicate.

(2) incompatible(assert(X), retract(X))

means that asserting and retracting the same fact at the same time are incompatible, where X is a literal.

(3) incompatible(add(O,S,V1), delete(O,S,V2),
single_value(O,S))

means that adding a value V1 to a slot S of an object O and deleting a value V2 from the same slot of the same object are incompatible if the slot is a single-valued slot.

(4) incompatible(modify(O,S,V1), delete(O,S,V2),
single_value(O,S))

means that modifying the value of a slot S of an object O to V1 and deleting a value V2 from the same slot of the same object are incompatible if the slot is a single-valued slot.

(5) incompatible(room(X), in(john,X), in(mary,X))

means that for a particular application we have the negative constraint "John and Mary can not be in the same room".

(6) incompatible(on(X,X))

means that an object X can not be on itself.

SEMANTICS CHECKER

The semantics checker has two major functions: cnecking facts in a knowledge base of an application written in the object shell for violations of the semantic constraints, and checking the semantic constraints themselves for internal consistency and agreement with other metaconstraints represented in the metashell.

We can have "positive constraints" and "negative constraints". The facts in the knowledge base must satisfy the former, but must not satisfy the latter. The constraints will be stated by using the meta-predicates of the meta-language. A negative constraint can be represented by using the meta-predicate "incompatible" as discussed before. Some of the meta-predicates (meta-relations) for specifying positive constraints such as range constraints, minimum/maximum cardinality constraints, legal value constraints, value compatibility constraints, subrelations, inverses, data types, etc. are shown below:

(1) lower_upper(slot, class, lower, upper)

This metarelation defines the legal range of numerical values: the values for the <slot> of the <class> must be between <lower> and <upper>. EVA discovers and flags values that exceed these bounds.

EVA not only checks facts against semantic constraints, but also checks that the semantic constraints themselves are consistent. Since a "child" is a "person", the age range of "child" must fall within the age range of "person". Thus the following semantic constraints would be inconsistent.

is_a(child, person)

lower_upper(age, person, 1, 110)

lower_upper(age, child, 0, 12)

(2) legal_value(slot, class, values)

This metarelation defines the legal values of a slot: the values for the <slot> of the <class> must be listed in <values>.

For example, the following semantic constraint

legal_value(gender, student, [male,female,hermaphrodite])

states that the gender of a student must be male, female, or hermaphrodite. EVA flags any "student" record where "gender" has a nonlegal value.

(3) relation(rel(domain-1,...,domain-n))

This metarelation defines both the number of legal arguments of a relation <rel> and the legal data type of each argument; <domain-i> is either a class of objects or a set of values for i=1,...,n.

This kind of semantic constraint is used to permit EVA to enforce strong data-type checking for relations.

For example, given the two facts

person(charlie)

and

dog(snoopy)

and the metafact

relation(murderer_of(person,person))

EVA flags as erroneous the fact

murderer_of(charlie,snoopy).

Another example of constraints

relation(enroll(freshman,{math101,english101,...}))

relation(enroll(sophomore,{math201,art201,...}))

states that a freshman can only enroll in Math101, English101,... that a sophomore can only enroll in Math201, Art201, ..., and so on.

(4) min_max_rel(relation, domain, min, max)

This metarelation specifies the minimum and maximum number of tuples (records) of a relation: the number of records of <relation> with object in <domain> must be between <min> and <max>.

For example, the following semantic constraint

min_max_rel(enroll, student, 0, 5000)

means that up to 5000 student enrollments are allowed in the data base at any one time. The enrollments are represented by records or tuples of the relation "enroll".

(5) min_max_role(relation, domain, min, max)

This metarelation defines that each object in <domain> must have at least <min> and at most <max> objects for the relation <relation>.

Thus the semantic constraint

min_max_role(enroll, sophomore, 3, 4)

states that each sophomore must enroll in at least 3 and at most 4 courses. EVA flags any sophomore who enrolls in fewer than three or more than four courses.

(6) subrelation(relation1, relation2)

This metarelation defines that <relation1> is a subrelation of <relation2>. EVA checks that the number of arguments for <relation1> is greater than or equal to the one for <relation2>, and that the data types of the arguments of <relation1> are subclasses of the corresponding arguments of <relation2>.

For example, EVA determines that the semantic constraints

relation(killer_of (animate_obj, animate_obj))

relation(murderer_of (person,thing))

subrelation(murderer_of, killer_of)

are inconsistent since the second argument "thing" of "murderer_of" is not a subclass of the second argument "animate_obj" of "killer_of".

EVA also checks that the inverse of <relation1> is a subrelation of the inverse of <relation2>, and creates the missing inverse of a subrelation, if one does not exist.

Our meta-language will also permit the developer to define properties of predicates or relations, such as *transitive, non-transitive, symmetric, non-symmetric, reflexive, irreflexive, antonymous* (male-female, ie., non-male implies female and vice versa), *contrary* (young-old, ie., non-young does not imply old), etc.

OMISSION CHECKER

Knowledge can be organized around the concept of set, eg., a class of objects, a class of relations, a class of rules, and a set of values for a multi-value slot. Given a set written by the developer, the basic question to ask is "Is the set complete?". In other words, does the set contain all the necessary elements or lack some elements? The goal of the omission checker is to answer this question by investigating and identifying useful techniques and representations for defining completeness of a knowledge base. Some of these techniques are given as follows:

(a) Class Taxonomy

If the developer creates only the classes for BOY, MAN and WOMAN,

PERSON(sex:{male,female},age:integer)

>>> BOY(sex=male,age≤12)

>>> MAN(sex=male,age>12)

>>> WOMAN(sex=female,age>12)

(where the classes are written in upper case, the slots in lower case, and the subclass relationship is denoted by >>>,) then the

omission checker will prompt him if there should be a class for persons who are female and not older than 12.

(b) Relation Taxonomy

Given a knowledge base as shown below,

PERSON >>> MAN
>>> WOMAN

PARENT-OF(person,person)
v
v
v
FATHER-OF(man,person)

the omission checker will find that the taxonomy for PARENT-OF is incomplete because the PERSON class for the first argument of PARENT-OF seems to split into the MAN and WOMAN classes. Therefore, it will prompt the developer whether there should be another subrelation of PARENT-OF that holds between WOMAN and PERSON, namely, MOTHER-OF.

(c) Omission of Rules Or Facts

Arguments of predicates may be associated with classes. By analyzing and comparing the arguments, the omission checker may detect that certain facts or rules for some classes are missing.

Consider the following knowledge base:

PERSON>>>ADULT>>>WOMAN
>>>MAN
>>>CHILD >>>GIRL
>>>BOY

GO(passenger,from,to).

TAKE(passenger,flight,fare).

DEPARTMENT(department_name,floor).

ADULT(x) $\wedge$ GO(x,austin,atlanta) $\rightarrow$ TAKE(x,f#7,150)

CHILD(x) $\wedge$ GO(x,austin,atlanta) $\rightarrow$ TAKE(x,f#7,75)

DEPARTMENT(man,2).

DEPARTMENT(woman,1).

If we look at the rules defining TAKE, we know the domain for the first argument of TAKE is the union of ADULT and CHILD, namely, PERSON. However, the domain for the first argument of DEPARTMENT is the union of MAN and WOMAN, namely, ADULT. The idea of checking missing rules or facts is to find the minimal class that is the domain for some argument of a predicate. If two minimal classes are related by the subclass relationship, then the rule or fact set associated with the smaller minimal class is likely to be incomplete. For the above example, the fact set for DEPARTMENT is incomplete. That is, the omission checker will prompt the developer on which floor the "child" department is.

(d) Incomplete Slot Values

There may be a set of typical objects for a slot of an object. For example, a room is a complex object that contains many other objects as parts. The room can be represented by a schema which has a slot named "containing". A value of this slot is a set of other objects, typically such as table, chair, board, PC, etc. These typical parts of an object can be stored in the meta-knowledge base. When a specific object is created and it does not contain some of the typical parts, the omission checker will tell the developer.

RULE REFINER

A rule may be too general or too restrictive. Specific test cases will be chosen from the knowledge base to prompt the expert if the rule should apply to the test cases. Any "no" answer will indicate that the rule is too general, and more specific rules will be proposed. If he answers "yes" to all the test cases, it may indicate that the rule is too restrictive, and other test cases in sibling classes of the generalization hierarchy will be chosen.

Consider a knowledge base given as follows:

PERSON>>>MAN---MAN(Sam,22,USA)
---MAN(Ted,42,USA)
---MAN(Ray,52,France)
>>>WOMAN---WOMAN(Sara,37,USA)

where the schema for PERSON is PERSON(*name,age,place_of_birth*).

If we have the following rule saying that every man can be the president of USA

MAN(x) $\rightarrow$ CAN-BE-PRESIDENT-OF-USA(x),

then the rule refiner will test the rule by presenting some instances of MAN and asking the developer if Sam, Ted and Ray can be the president of USA. The answer will be "no" for Sam and Ray, and "yes" for Ted. Since there are "no" answers, the rule is too general, so that the developer will change the rule to

MAN(x) $\wedge \geq$ age(x,35) $\wedge$ place_of_birth(x,USA)
$\rightarrow$ CAN-BE-PRESIDENT-OF-USA(x).

Now, the rule refiner will test the modified rule by presenting an instance of WOMAN and ask if Sara can be the president of USA. The answer will be "yes". This means that the modified rule is too restrictive. Therefore, the rule will be changed to

PERSON(x) $\wedge \geq$ age(x,35) $\wedge$ place_of_birth(x,USA)
$\rightarrow$ CAN-BE-PRESIDENT-OF-USA(x).

The goal of the rule refiner is to help the developer refine his rules. Since this is an interactive process, a good and comprehensive user interface is required.

CONTROL CHECKER

As larger knowledge bases for complex applications are implemented, some software engineering methodology [Jacob and Froscher 1986] should be developed. One method is to partition a large knowledge base into smaller subsets of facts and rules. These subsets can be labeled by meaningful names such as activity names. Conversely, we can start with the activity names and then implement each activity by a set of facts and rules.

The implicit execution model of a knowledge-based system is given as follows: A rule has a RHS and LHS. The rule will be fired if the LHS is satisfied by the knowledge base. When the rule is fired, the RHS tells the system to add, change or delete facts and objects.

In an application, there may be "ordering" constraints (called control constraints) among the activities. The control checker permits the developer to specify the control constraints, and then verify if rules in a knowledge base will be executed in a sequence that does not violate the control constraints. For example, in an office system, there are the activities for clearing and publishing papers. An ordering constraint is that a paper must be "cleared" before it is "published". Assume the activities are specified as follows:

CLEARING_ACTIVITY:

paper(X) $\wedge$ approved(X) $\rightarrow$ cleared(X).

PUBLISHING_ACTIVITY:

paper(X) $\wedge$ accepted(X) $\rightarrow$ publish(X).

EVA will recognize that the control constraint is violated because there are no data dependencies between these two activities. That is, there are no RHS-literals in the first activity used in the LHS of the rule in the second activity.

TEST CASE GENERATOR

As discussed before, a knowledge base consisting of facts and rules can be represented by a connection graph. In the connection graph, there are two kinds of leaf nodes, namely, *input* nodes and *output* nodes. An input node is a node representing a fact that is connected to LHS-literals of some rules. An output node is a node representing a RHS-literal that is not connected to any LHS-literals.

There are two ways to test the knowledge-based system. One approach is to generate different sets of input nodes (input test cases) to exercise the system and observe data produced at the output nodes. The goal is to traverse each arc in the connection graph at least once. The input test cases must satisfy semantic constraints that are imposed on the system.

The other approach is to specify requirements on data at the output nodes. Each requirement will be represented as a query. EVA will check that all input facts will satisfy all the queries.

Consider an example where the speed of an engine is controlled by the position of a valve of a fuel system. A value of the position and a value of the speed are input and output data, respectively. We may generate different values of the position and observe the values of the speed. On the other hand, we can specify that the speed should fall within a certain range and then check if such a requirement can be fulfilled.

ERROR LOCATOR

The error locator is to locate "incorrect" rules which derive "incorrect" facts from input facts. For example, consider an adder that is specified by the following rule:

input(1,N,V1) $\wedge$ input(2,N,V2) $\wedge$ adder(N)

$\rightarrow$ output(N, V1+V2).

The adder has two input ports 1 and 2. It takes values V1 and V2 at the input ports, and produces the sum of the values at its output port. If we have the following input facts for adder a

input(1,a,10)

input(2,a,50)

the system should produce

output(a,60).

Now, if the adder is specified by the incorrect rule

input(1,N,V1) $\wedge$ input(1,N,V2) $\wedge$ adder(N)

$\rightarrow$ output(N, V1+V2),

the system will produce the incorrect output

output(a,20).

To help the developer, the error locator will present him the deduction tree of the incorrect fact so that he can debug it. For the above example, the deduction tree uses only one fact, namely, input(1,a,10), with the rule. This should give the developer the necessary hint to correct the incorrect rule.

BEHAVIOR VERIFIER

A system may be decomposed into many subsystems. A subsystem may be represented by a collection of facts and rules in the object shell. However, the subsystem must have external input/output interfaces to communicate with the outside world. For example, in the space shuttle flight software system, the navigation controller is a subsystem that sits in a control loop, collects and analyzes data, and then sends control signals to the vehicle manipulator.

The behavior of the subsystem is a description of relationships among the external input/output interfaces and internal states of the subsystem. The subsystems are connected together to form the total system. The purpose of the behavior verifier is to prove that the intended behavior of the system can be derived from the behaviors of the subsystems and the description of their connections.

CONCLUSION

This paper has described the architecture and functionality of EVA. It is evident that EVA provides a powerful means for representing knowledge about an application domain and for verifying that the knowledge is correct, consistent and complete. EVA increases the reliability of knowledge-based systems, speeds up their development, and assists in their continuing modification. The necessity for such validation tools will continue to grow as future knowledge-based systems play a more critical role in business, industry, government, and the sciences.

ACKNOWLEDGMENTS

The authors would like to thank Linda B. Burris of Lockheed Artificial Intelligence Center for her careful reading of the paper and many useful comments.

REFERENCES

Chang, C.L. [1976] DEDUCE --- A deductive query language for relational data bases, in *Pattern Recognition and Artificial Intelligence* (C.H. Chen, Ed.), Academic Press, Inc., New York, 1976, pp.108-134.

Chang, C.L. [1978] DEDUCE 2: Further investigations of deduction in relational data bases, in *Logic and Data Bases*(H. Gallaire and J. Minker, Eds.), Plenum Publishing Corp., New York, 1978, pp.201-236.

Chang, C.L. [1981] On evaluation of queries containing derived relations in a relational data base, in *Advances in Data Base Theory --- Volume 1* (H. Gallaire, J. Minker and J.M. Nicolas, Eds.) Plenum Publishing Corp., 1981, pp.235-260.

Jacob, R.J.K., and Froscher, J.N. [1986] *Developing a software engineering methodology for knowledge-based systems*, NRL Report 9019, Computer Science and Systems Branch, Information Technology Division, Naval Research Laboratory, Washington, D.C. 20375-5000.

Nguyen, T.A. [1987] Verifying consistency of production systems, *Proc. of the 3rd IEEE Conference on AI Applications*, February 1987, pp.4-8.

Nguyen, T.A., Perkins, W.A., Laffey, T.J., and Pecora, D. [1985] Checking an expert systems knowledge base for consistency and completeness, *Proc. of the 9th International Joint Conference on Artificial Intelligence*, 1985, pp.375-378.

Quintus [1985] *Quintus Prolog Reference Manual*, Quintus Computer Systems, Inc., 2345 Yale Street, Palo Alto, CA 94306.

Reubenstein, H.B. [1985] *OPMAN: An OPS5 rule base editing and maintenance package*, MIT Master's thesis, MIT, AI Laboratory, 545 Technology Square, Cambrideg, Ma 02139.

Stachowitz, R.A., and Combs, J.B. [1987a] Validation of Expert Systems, *Proc. of the 20th Hawaii International Conference on Systems Sciences*, 1987, pp.686-695.

Stachowitz, R.A., Combs, J.B., and Chang, C.L. [1987b] Validation of Knowledge-Based Systems, *Proc. of the 2nd AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program*, Arlington, Virginia, March 9-11, 1987.

Suwa, M., Scott, A.C., and Shortliffe, E.H. [1982] An approach to verifying completeness and consistency in a rule-based expert system, *The AI Magazine*, 1982, pp.16-21.

THE AUTHORS

Dr. Stachowitz received his Ph.D. in Linguistics from the University of Texas at Austin in 1969. His background includes design and development of a large-scale knowledge-based mechanical translation system, computer hardware and software performance evaluation, and research in applicative programming languages, semantic data models, and analytic modeling and performance evaluation of data base machine architectures. He also has performed research in logic and functional knowledge base manipulation and query languages. He is currently a research scientist at Lockheed's Artificial Intelligence Center where he is the co-principal investigator of the Knowledge-Based Systems Validation project, the topic of this paper.

Dr. Chang received his Ph.D. in Electrical Engineering from the University of California, Berkeley, CA in 1967. His background includes design and developemnt of large-scaled knowledge-based systems, and research in program generation, very high level languages, compilers, rapid prototyping, relational data bases, natural language query systems, mechanical theorem proving, and pat-

tern recognition. He wrote two books "Symbolic Logic and Mechanical Theorem Proving" and "Introduction to Artificial Intelligence Techniques", and published more than 50 papers. He is currently a co-principal investigator of the Knowledge-Based Validation project.

Todd Stock received his B.A. in Computer Science from the University of Texas at Austin in 1986. He worked for the UT CS department doing research and implementation of artificial intelligence and machine learning systems under Dr. Bruce Porter. He is currently at Lockheed's Artificial Intelligence Center where he is implementing the Knowledge-Based Systems Validation project in Prolog.

Ms. Combs received her M.S. in Computer Science/Mathematics from the University of Texas at Dallas in 1984, where she was awarded a research assistantship in the area of artificial intelligence. She was a member of the technical staff of the Corporate Engineering Center at Texas Instruments in Dallas, where her work included the design and development of process control knowledge-based systems and expert system shells. She is presently working at Lockheed's Artificial Intelligence Center where she is the associate principal investigator on the Knowledge-Based Systems Validation project.

DIAGNOSIS: REASONING FROM FIRST PRINCIPLES AND EXPERIENTIAL KNOWLEDGE

Linda J.F. Williams and Dennis G. Lawler
McDonnell Douglas Astronautics Company
Engineering Services
16055 Space Center Blvd.
Houston, TX 77062

1. INTRODUCTION

The overall goal of this paper is to show that completeness and efficiency can be obtained when automating diagnostic reasoning systems by using a combination of two approaches; diagnosis from first principles and diagnosis from experiential knowledge.

What we mean by diagnosis from first principles is reasoning about a malfunction using design knowledge; the design knowledge being the description of the system structure and behavior that can be obtained from documentation, schematics, drawings, etc. This approach to diagnosing a malfunctioning system through the use of a deep understanding of the fundamental structure and behavior of the system and its components has the target of providing an expert's troubleshooting ability without explicitly modeling the expert (6). In contrast to this is the approach of diagnosis from experiential knowledge. Experiential knowledge is of course knowledge gained from experience; knowledge of how to troubleshoot a system or reason about a malfunctioning system using knowledge gained from a practical viewpoint. Earlier diagnostic systems such as MYCIN used this approach; (3,7) MYCIN derived its capabilities by implementing a model of an expert's experienced-based reasoning. In section 2 and 3 we discuss further the ideas of diagnosis from first principles and diagnosis from experiential knowledge.

The efficacy of diagnosis from experiential knowledge has been proved through the development of useful application programs currently being integrated into various operations and organizations. Applications that display reasoning behavior based on first principles, i.e. a 'deep knowledge' of the system, are not pervasive yet but research is very active in this area (2). First principle knowledge about the structure and behavior of common system components allows the development of library packages of knowledge for general use in many applications. Additionally, it is thought that first principle diagnostic reasoning systems will provide a more complete understanding of both system function and malfunction than the experiential reasoning approach. Both approaches have distinct advantages to offer for development of a diagnostic application and both approaches are sufficient for use in some situations; (3,7)(6) however, both approaches have disadvantages when implemented as the only diagnostic reasoning process in an automated system. These disadvantages prevent either approach from being adequate or robust enough to handle diagnostic reasoning needed for the highly complex systems being studied and developed in todays space programs. We expand our discussion of advantages and disadvantages of both approaches in section 4 and give examples showing why a combination of the two approaches is necessary to obtain a complete and efficient automated diagnostic reasoning system in section 5.

Knowledge acquisition for intelligent diagnostic reasoning systems is an issue that deserves some discussion. Although we are capable of automating diagnostic information, obtaining a 'complete set' of knowledge needed for an efficient diagnostic reasoning system can be a difficult and endless process. We feel that a systematic approach to acquiring knowledge will facilitate defining the program's scope of competence and assist in guiding the knowledge acquisition process. In section 6 we discuss knowledge acquisition relative to our observation about a combined reasoning approach to diagnosis and elaborate on what knowledge is necessary and where the knowledge is likely to be found.

2. DIAGNOSIS FROM FIRST PRINCIPLES

A complete description of diagnosis from first principles would require a much deeper discussion than is appropriate for this paper. In lieu of this we present a brief discussion based on illustrating what we feel are the major concepts of the approach and their interrelation and definitions.

Tersely, diagnosis from first principles is

reasoning solely from a description of the structure and behavior of the system and its components to explain discrepancies between observed and correct system behavior. By structure we mean a complete list of a system's components and the specification of their connectivity.

The components in this description are all the elements of the system that conceivably effect the system's behavior. Each component specification should include a description of the generic characteristics and behavior of the component as well as any additional configuration-specific characteristics and behavior peculiar to the system in question (e.g. resistance, EPROM programming, etc.). In operation the diagnostic program would use such information to determine the qualitative state of both the internal component behavior as well as the component's external environment (e.g. is it receiving appropriate input, is the component overheated, etc.).

Connectivity here means the specification of all possible connections that provide a path for a relevant effect or influence on a component. The term effect here must be taken in the broadest sense possible meaning not only physical but abstract effects such as instruction or information passing between components. Specific types of effects that are used depend on the type and resolution of the specific model being used to represent the behavior of the component in question. Where appropriate, distinction might be made between intended and unintended effects on a component and the attendant paths. Effects, of course, must be observed in some sense so that the appropriate component behavior can be computed and compared with the actual behavior exhibited by the system. Hence a set of observables must be specified which are needed for this computation. These would include all component inputs and outputs as well as the value of any internal state variables (5). When determining the health of particular component the value of every pertinent observable may not be available. In such cases a measurement may be needed. Commonly this involves the use of a specific piece of instrumentation, either in place (e.g. BIT/BITE) or provided by the expert (e.g. the use of a multimeter), or by reconfiguring the system in some sense and then remeasuring specific observables. If making a measurement is not possible, the value may have to be derived indirectly via a computation involving other observables, or by a heuristic relationship.

In addition to the above descriptive knowledge about a system, a general computational approach to diagnosis that utilizes this knowledge is necessary for a complete diagnostic program. Several of these computational approaches do exist (6) however this is still an active research area and is beyond the scope of this paper.

3. DIAGNOSIS FROM EXPERIENCE

When diagnosing a system the expert(s) have several conceptual tasks confronting them: monitoring the system observables, detecting malfunctioning behavior from this monitoring task, isolating the component(s) responsible for the behavior, and recovering from the malfunction. These tasks are usually approached with a specific strategy to optimize the task performance. The monitoring strategy (e.g. frequency of observation) is developed a priori.

The detection and isolation tasks are performed with the monitored information and with strategies whose specifics commonly change depending on the type of behavior being observed. For any reasonable size of system the strategy can become so complex as to be arbitrary to an uninformed observer of the expert's performance. As mentioned in the discussion above various means may be needed to gather information about a particular state variable and so the development of a behavior specific measurement strategy is commonly required. Additionally, a strategy for recovering the system to a safe and/or effective behavior is also required.

The above discussion of diagnosis from first principles includes many of the concepts needed to support these four general task areas but not all the considerations that are used by the expert in performing the diagnosis. Specifically missing is a discussion of those situation observables that are required for the development of the measurement and recovery strategies.

These strategy development tasks very frequently involve reasoning about extra-system factors that would not necessarily be explicitly modeled in the structure and behavior of the system itself or would be impossible to model adequately given the current understanding of the system in question. Also, this reasoning is quite likely to be context dependent, highly judgmental, and almost certainly experientially derived. The reasons for this vary but some common ones are: cost and/or design considerations limit the amount of instrumentation in the system, cost and/or design considerations limit the amount of measurement access (e.g. IC packaging), quality of information may vary (e.g. instrumentation failure), context may limit plan for measurement (e.g. safety limitations, politics or other indeterministic reasoning) may come into play.

4. COMPARISON OF APPROACHES

In order to understand why a combination of diagnosis from first principles and diagnosis from experiential knowledge is needed to automate complete and efficient diagnostic reasoning systems, we begin to explore the advantages and disadvantages of each approach.

There are several advantages in basing reasoning on first principle knowledge. A model developed from a description of structure and behavior provides a fairly complete and indepth description of the system that is to be automated. This description provides a deep understanding of the system that the expert troubleshooter will often refer to in order to compliment experiential knowledge. Since this model includes descriptions of all components incorporated in the system being modelled, there is knowledge about components that can be ported across applications. For example, if the system we wish to model is a device built from digital logic components, each digital logic component that is described can be reused when modeling a system that requires the same component (2). Since we can describe every component in a system, irregardless of the level of detail, a library of component descriptions can be built to facilitate construction of new programs that describe different (but similar) systems (4). A system based on reasoning from first principles can be easier to maintain because modifications to the design are fairly simple to implement. Structure and behavior specifications can be updated for each modified component rather than modifying the overall behavior of the entire system (2).

Another advantage of reasoning from first principles involves reasoning about novel faults. Since reasoning from first principles is not dependent on a catalog of observed malfunctioning behavior, it is possible to reason about a fault that has not occurred previously (2). Reasoning about novel faults is something experts are equipped to handle, but this type of experiential knowledge is difficult to encode in a diagnostic reasoning program simply because you can't inquire about malfunctioning behavior the expert has not dealt with.

There are several areas of experiential knowledge that can be encoded into a program that assist in developing a more complete diagnostic reasoning system. An expert is often aware of connectivity and adjacency not explicit in first principle knowledge. The expert also uses common sense reasoning and has the ability to reason outside a closed system domain. For example; if a system that contained exposed electrical circuits exhibited a fault after a rainstorm, the expert would tend to associate a malfunction with information about bad weather and check exposed circuits before performing a systematic search to isolate the malfunction. An expert can also develop a list of ordered categories of failure for each observed malfunction. By categories of failure we mean a list of possible search paths used to isolate a particular failure. In order to reason about a malfunction, the expert uses several pieces of information (e.g. similarities to other malfunction behavior, experience from similar systems, component failure history, external effects, knowledge about where and how to take measurements) and constructs a plan or measurement strategy to guide the troubleshooting process. The plan the expert has developed is essentially an ordered list of categories of failure. The expert also has the ability to know when an incorrect assumption has been made (i.e. an incorrect path has been followed) and how to regress and continue the troubleshooting process.(2)

Enumerating the advantages of reasoning from first principles and the advantages of reasoning from experiential knowledge has shown us that both are good approaches to automating diagnostic reasoning systems, however, in order to understand why both are not completely adequate as stand alone approaches we will now examine the disadvantages of each approach.

A program designed to reason from first principles will have difficulties constraining possible causes of failure. When reasoning solely from first principles, the troubleshooting process involves a systematic search of all possible paths that might lead to the malfunctioning component. There is seldom enough information to indicate a reasonable ordering of search paths, or to constrain the systematic search to a reasonable subset of search paths. In contrast to this, the expert rarely begins a troubleshooting process without constraining and ordering the possible search paths, this will allow the expert to reach a conclusion about a fault more rapidly than a system based entirely on first principle knowledge, if the fault can be reached by one of the paths the expert has selected. If the fault lies outside of the planned search paths the expert will be required to consult first principle knowledge located in some form of documentation, this will then increase the time necessary for the to expert to diagnose a fault.

Any time a program is required to reason with incomplete data, deficiencies occur. Many systems that are candidates for automation have incomplete, inaccurate, or unavailable documentation, this causes difficulties when attempting to reason using either first principles or experiential knowledge. An automated system reasoning from first principle knowledge is only as accurate as the documentation that was used to build it, however, experiential knowledge that can be encoded in programs is quite often based on reasoning about a system where system documentation is incomplete.

A program designed to reason from experiential knowledge is based on empirical associations and is usually difficult to construct. The process of attempting to model a person is long and there is often no way to know if a complete set of knowledge has been extracted from the expert. When developing a program that reasons from experiential knowledge, the developer often must choose between experts who have different opinions about how to solve problems

and different ideas about the cause of faults that have occurred previously. When making these choices the developer will possibly limit the efficiency of the program. In contrast, a program that reasons from first principles is not limited to a set of predefined solutions and can reason about a fault without any limitations on the conclusions that can be reached.

Programs that reason from experiential knowledge are restricted by an expert's sample cases, and restricted to the domain there were intended for. As mentioned earlier, the process the expert uses to reason about a novel fault is not an area of reasoning that can be automated using todays technology. Therefore, reasoning strictly from experiential knowledge restricts the program to reasoning only about cases elaborated on by the expert.(2)

5. COMBINING APPROACHES

From the discussion so far, it should be obvious to the reader (as it might have been prior to this reading, since we are not pretending to present unique or unusual ideas but simply our understanding of the requirements for a diagnostic system in the field) it is our view that both first principle and experiential reasoning are necessary for a robust diagnostic program operating outside the laboratory environment. We offer the following examples from the Space Program to illustrate the nature of this mixture of reasoning approaches.

For the handling of malfunctions on the Shuttle a set of procedures called Malfunction Procedures (MALF's) are generated for the crew to follow after they observe some abnormal behavior in a system. These MALF's, which take the form of a decision tree, embody both the first principle and experiential reasoning discussed above. They are developed to take into consideration a wide variety of possible system failures in a variety of system contexts in order to plan well understood and safe malfunction responses. However, a complete specification of all system failures is not practicable (let alone possible) and where appropriate the crew is directed to simply contact Mission Control and report their observations. The personnel on the ground in turn have a more detailed set of MALF's that are used as above, but also have a complete set of documents that represent a first principle understanding of the system and are frequently referred to during diagnosis. Additionally, these personnel have a wealth of experience with the system in question, and the world in general, which they draw upon for reasoning that must go beyond an understanding available from the documentation only.

As an example of extra-system knowledge being brought to bear on system diagnosis we can look at the actual experience of a Manned Maneuvering Unit pilot on the way back to the Shuttle during STS 41C. He observed that the relative size of the Shuttle was growing faster than appropriate for an approach at constant velocity. This indicated that there was a definite relative acceleration along the line-of-sight to the Shuttle. The situation occurred within an agreed context between the individuals involved in the approach, the MMU pilot would be the only individual imparting a relative velocity between the vehicles. One possible explanation for this, that can be derived from an in-depth understanding of the MMU's structure and behavior, is a stuck-on MMU thruster; a fairly serious malfunction. According to good training and his understanding of the situation context, the pilot then began to correct for the unintended acceleration by slowing the MMU and proceeded to inform Mission Control of his situation. At that time he learned that the Shuttle pilot was actually accelerating the Shuttle toward him. With this new contextual information the pilot then could explain his observables in terms of these extra-system factors.

Given this view of the nature of a major portion of the reasoning needed for a comprehensive diagnostic system, how does one go about developing such a program. It as our view that the best approach would be to divide the needed reasoning tasks into those tasks and subtasks directly accomplished via the first principle approach and those tasks subtasks that are better handled by modeling the behavior of the expert directly.

6. KNOWLEDGE ACQUISITION

Dividing reasoning tasks into sections corresponding to the two approaches has some effects on the knowledge acquisition process that are advantageous to the program construction process. Combining the approaches permits the systematic enumeration of a large portion of the knowledge needed to construct a diagnostic reasoning program; this facilitates defining the program's scope, completeness and competence, and assists in bounding, controlling and guiding the knowledge acquisition process. In this paper we consider knowledge acquisition to be acquisition of all knowledge that will be incorporated into a program; this includes knowledge from document sources as well as knowledge from an expert(s).

First principle knowledge to be acquired from documentation, schematics, drawings, etc. can be stated explicitly by enumerating components, correct component behavior, component connectivity and effects, and description of observables. Although extracting knowledge from documented material is not terribly difficult, (especially when compared to extracting knowledge from a human) it is still a complex task and can be simplified by extracting the needed knowledge using a systematic method (2). Once the process of gathering first principle knowledge is completed, the task of extracting knowledge from the expert is reduced because you are not relying on the expert to provide

you with necessary design knowledge about structure and behavior.

Unfortunately, the approach to enumerating knowledge required from the expert is not as systematic as with first principle knowledge gathering; however, a set of goals can be generated to help elicit knowledge from the expert. The list of goals are intended to guide the interaction with the expert toward the elicitation of strategies used to perform the monitoring task, detection task, isolation task, and recovery task (discussed in section 3). Knowledge acquired from this effort includes (but is not limited to) determining components not deseribed in documentation, implied connections not explicit in documentation, environmental effects on components and component behavior, previous failures, failure trends, untestable observables, information about components inferred from measurements of other components (8), functional leveling (the amount of structural and behavior detail needed to model components varies with level of abstraction), and ordering of categories of failure (discussed in section 4).

When attempting to acquire knowledge from experts, it should be realized that they tend to have a variety a models that are used during a diagnosis. Gaining insight into what model the expert is using and how the expert developed the model can reveal valuable information about how the expert performs a troubleshooting task and information about different levels of structural and behavioral detail needed to reason about a fault (4). For example; when diagnosing a car that won't start, you would rarely begin by reasoning about a wiring harness diagram and its connections to the ignition system. Rather, you would most likely think of the wiring harness as a 'black box' until there is an indication that the fault lies within the wiring harness. Unfortunately, an expert's selection and development of models is a process that is not well understood (1) making the extraction of these models a difficult and involved part of knowledge acquisition. Since experts are rarely explicit about the models they use, it is advisable to construct scenarios the expert can reason about and develop the model from strategy paths the expert uses (8).

Since we are combining two approaches, the question arises concerning how much knowledge from each area should be included in the program and when is the knowledge acquisition job complete? As with any project, the desire is to make the program as complete as possible, however, the underlying issue is still the balancing of completeness (detail of structure and behavior) and the need to constrain the search paths (categories of failure) the program will reason about when diagnosing a fault. A program that cannot respond quickly to a malfunction will certainly be unacceptable in certain domains, as such each project must determine the overall requirements of the program before the knowledge acquisition process is initiated (8).

7. CONCLUSIONS

Completeness, efficiency and autonomy are requirements for future diagnostic reasoning systems. Methods for automating diagnostic reasoning systems include diagnosis from first principles (i.e. reasoning from a thorough description of structure and behavior) and diagnosis from experiential knowledge (i.e. reasoning from a set of examples obtained from experts), however, implementation of either as a single reasoning method fails to meet these requirements. The approach of combining reasoning from first principles and reasoning from experiential knowledge does address the requirements discussed above and can possibly ease some of the difficulties associated with knowledge acquisition by allowing developers to systematically enumerate a portion of the knowledge necessary to build the diagnosis program. The ability to enumerate knowledge systematically facilitates defining the program's scope, completeness, and competence and assists in bounding, controlling, and guiding the knowledge acquisition process.

REFERENCES

1. Davis, R., "Robustness and Transparency in Intelligent Systems", Symposium on Human Factors Needs in Space Station Design, Washington, DC, Jan 1987.

2. Davis, R., "Diagnostic Reasoning Based on Structure and Behavior", Artificial Intelligence, Elsevier Science Publishers B.V. (North-Holland), 24, 1984, 347-410.

3. Davis, R., Buchanan, B. and Shortliffe, E., "Production Rules as a Representation in a Knowledge-Based Consultation System", Artificial Intelligence, Elsevier Science Publishers B.V. (North-Holland), 8, 1977, 15-45.

4. De Jong, K., "Knowledge Acquisition for Fault Isolation Expert Systems", Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov 1986.

5. Hamscher, W., and Davis, R., "Diagnosing Circuits with State: An Inherently Underconstrained Problem", Proceedings Of National Conference on AI, Austin, TX, August 1984.

6. Reiter, R., "A Theory of Diagnosis from First Principles", Artificial Intelligence, Elsevier Science Publishers B.V. (North-Holland), 32, 1987, 57-95.

7. Shortliffe, E., Computer-Based Medical Consultations: Mycin, American Elsevier, New York, 1976.

8. Woods, D. and Hollnagel, E., "Mapping Cognitive Demands and Activities in Complex Problem Solving Worlds", Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, Nov 1986.

VERIFYING SHUTTLE ONBOARD SOFTWARE USING EXPERT SYSTEMS

William B. Wingert
IBM
Federal Systems Division
3700 Bay Area Boulevard
Houston, Texas 77058-1199

ABSTRACT

The Space shuttle uses a complex set of software to guide, navigate, and control it through all phases of flight. Adding to the complexity is the fact that the software is "reconfigured" for each flight. That is, thousands of constants in the software are changed to reflect the unique properties of a given mission (e.g., launch weight, orbit inclination). In the last level of test, The software is "flown" through end-to-end nominal and abort scenarios taking the shuttle from liftoff to landing. The analysis of the results of the Level 8 testing is experience and labor intensive. A set of pass/fail criteria have been defined for each testcase and in parallel with the knowledge acquisition, tools were developed which allowed the automation of the knowledge being gathered on paper. A prototype of the Analysis Criteria Expert System (ACES) has been put into production in the verification of the reconfigured onboard flight software. The system currently uses 3 PL/I programs, the ESE/VM program product and two large host systems to accomplish the task. The total system has approximately 3000 rules. The knowledge acquisition has begun again to take the knowledge base beyond simple pass/fail to the ability to determine the source of the criteria failures.

I. BACKGROUND

A. Space Shuttle Software

Four IBM AP-101B flight computers host a set of highly critical and complex programs to guide, navigate and control the Space Shuttle through all phases of flight (see Figure 1). The flight software, developed by IBM under contract to the National Aeronautics and Space Administration (NASA), also drives a set of instruments and graphic displays, accepts keyboard and other inputs from the astronauts, and interfaces with various hardware sensors and effectors. The flight computer operating system ensures that all active computers operate simultaneously, each performing identical functions. A failed computer is detected automatically and removed from the set of redundant computers. Due to the complexity of the avionics and data processing systems, size of the software and reliability requirements, independent verification has been employed in all phases of the software life-cycle to increase product quality. The goal is flight software which is "error-free."

B. Flight Software Testing

A software test plan provides for a structured process to identify and facilitate correction of software implementation or requirements errors, leading to demonstration that the software satisfies all design requirements. Testing is divided into two main phases: development tests, which are concurrent with the software development and performed by the software development organization, and verification tests, which are carried out by the independent test organization. Figure 2 describes the elements of the test program.

C. Level 8 Testing

The first seven levels of testing are performed on the basic set of programs whose software logic can support many flights. Thousands of constants in the software are changed to reflect the unique properties of a given mission. Level 8 testing consists of testing the software under simulated flight conditions and stresses, with the flight software configured for a particular shuttle flight. The tests are conducted through flight simulations exercising the onboard software and computers in a simulated flight environment provided by the Software Production Facility. The volume of simulation data required to adequately analyze the performance of the flight software is impressive. Each test generates over two

million plotted data points and 30,000 lines of printed output. There are 15 - 25 tests run for each shuttle mission.

The analysis consists of evaluation of simulation variables at various events in the shuttle trajectory, analysis of flight sequences, and analysis of plots and cockpit displays. Execution and analysis of the suite of tests is both labor and skill intensive and requires up to six weeks for completion.

The goal of the Analysis Criteria Expert System (ACES) is to automate the analysis of the logged data. The benefits are numerous and include reducing labor costs, improving the quality and consistency in interpreting the data, and reducing the total time currently required to manually analyze the simulations.

II. KNOWLEDGE ACQUISITION AND ENGINEERING

The knowledge acquisition and engineering for ACES began in early 1985 when "pass/fail" criteria were created for a subset of our verification testcases. That effort was later extended to include more criteria and more testcases. In parallel with the knowledge acquisition, we began looking into different ways to automate the evaluation of the criteria. By the Spring of 1986, we had decided on a method for criteria automation and completed the first phase of knowledge acquisition. The knowledge base consisted of about 250 rules for each of the 14 testcases.

After using the criteria for about a year, we felt it was a good idea to take these criteria a step further and document how criteria violations were analyzed. In this process, we decided to reorganize the criteria. This reorganiztion meant duplicating some of the previous efforts. However, we felt that the expected size of the comprehensive criteria demanded the reorganization. The final set of the written criteria were in a form that could be picked up and independently implemented into an expert system. This process has been completed for about a third of our verification testcases.

III. IMPLEMENTATION

Implementing the pass/fail criteria proved to be much more difficult and complex than we anticipated. We encountered problems with automated data transfer from the simulation to the expert system, volume of data, rule base creation, and results presentation.

The problem of data volume turned out to be the biggest challenge. Each simulation produces over three million data points and 30,000 lines of printed output. The expert system shell we were using was the Expert System Environment (ESE) and was the only internally available mainframe expert system software. Due to ESE's lack of speed when handling large amounts of data, some sort of pre-processor would be necessary. We wrote three PL/I programs to handle the type of data we analyze.

AutoProg is used to process plotted data. However, it cannot handle textual data such as the simulation chronology (the online) and the textual representations of the Shuttle's onboard displays (the DEU images). To handle the Online message criteria we produced the Online Event Extractor (OLEVE). This program looks at the online output for missing, out of order, and unexpected messages in the chronology of the simulation. Each simulation logs images of the Shuttle's displays and later stores them in a file for processing by the Display Electronics Unit Criteria Evaluator (DEUCE). This PL/I program searches for a particular DEU based on some criteria such as time or within a certain amount of time of an event. This program can look for text strings in a group of DEUs, perform math operations, and print DEU images and captions for use in testcase reports. See figure 1 illustrating the flow of these programs

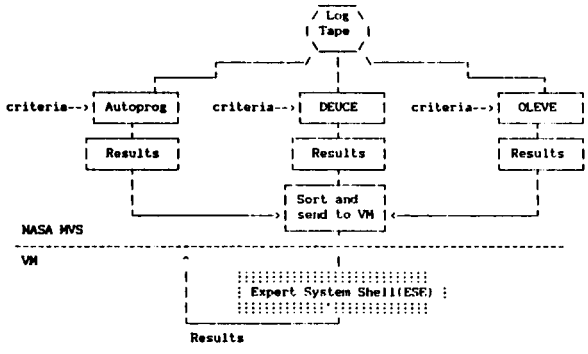


Figure 1 - ACES Flow

The entire process is automated from start to finish and requires no human intervention other than to submit the simulation job. ESE/VM can handle the smaller amounts of data produced by the PL/I programs within a reasonable timeframe. However, since most of our criteria reside in the PL/I program the ESE knowledge base is relatively small with about 20 rules for each testcase compared to more than 250 PL/I criteria. We are currently working on prototypes using other expert systems in the hopes of enlarging the knowledge base and producing an integrated expert system.

IV. VERIFICATION OF ACES

The verification of the expert system was divided into two distinct parts: tool verification and knowledge base verification. Tool verification concentrated on verifying AutoProg, DEUCE, and OLEVE. Knowledge Base verification centered on testing the rules supplied to the above tools as well as the knowledge base contained in the expert system.

A standard test approach was used for the tool verification. All of the tools went through a requirements, design, and code review process. Prior to code review, all of the tools were put through a set of unit tests designed to exercise all of the capabilities of the tool. Known valid inputs were fed to each of the tools and the developer analyzed the output for expected results. This review process, well known on the onboard Shuttle project, was easily implemented with very few tool problems.

The knowledge base verification is a different story altogether. Since the criteria were designed to work on any shuttle mission (of which there are essentially limitless combinations), we decided each knowledge base should be applied to at least three different shuttle flights. This was an arbitrary number, but we felt three different flights would give us good coverage over a range of shuttle trajectories. Wherever possible we had someone other than the developer of the knowledge base perform the testing. The analysts first performed manual analysis of the testcase using the written criteria as a guide. All problems and violations of the criteria were noted. The analyst then ran the expert system against the testcase and compared the results to that of the manual analysis. Differences were noted and probable source of the difference was noted, for example, knowledge base deficiency, knowledge base error, or tool error. All tool errors resulted in a Discrepancy Report or Change Request against the tool to bring it into compliance. Knowledge base errors were returned to the knowledge base developer for resolution.

As new versions of the tools become available, they are unit tested and executed with a small subset of the actual rules to insure identical results are produced. As knowledge is added it is tested as a standalone entity before being merged with the existing data. In production, if a tool or knowledge base problem is discovered, a Discrepancy Report is written to document the problem.

V. CURRENT USE AND FUTURE PLANS

The Flight Software Certification organization uses ACES today in their verification of the Shuttle onboard flight software. After a simulation is made, a job is submitted which automatically executes ACES and puts the output in a dataset. Some manual analysis is still performed on those criteria which are not yet covered by ACES.

Our main focus for the future centers on the expert system tools that we are evaluating. The expert system shell we are evaluating is ES PL/I, a PL/I-based imbeddable expert system shell. PL/I has proven to be faster, but all development is done using a standard text editor. We have built a prototype of our knowledge bases using ES PL/I. We have found it to be faster than ESE/VM. The main drawback has been documenting the rules. Rules are documented with ES PL/I by putting PL/I comments in line to the knowledge base. However, if the knowledge is well organized and catalogued, this poses no major technical problem. Development using ES PL/I requires a good understanding of PL/1 and the number of lines generated is similar to ES PL/I.

In general, we have found it useful to build a prototype of our knowledge base in order to evaluate its relative merits. We try to scope the size of the prototype such that it can be completed in a two to four week period. On the other hand, we try to include a good mix of rule types such that we can get a good feel for the amount of time it takes to implement the various types of rules. With the prototype, we can analyze the relative merits of an expert system, provide demonstrations for our customer, and tailor our paper knowledge to meet the requirements of a particular expert system shell, if necessary.

Expert systems can be used successfully to verify critical software. The time and resources required can be reduced and the quality of the verification can be maintained or improved by applying expert systems technology to an existing software verification effort. However, the transition takes time to perform the proper knowledge engineering and acquisition. In addition, with proper planning, it is possible to insert expert systems technology into an existing production environment.

Smart Bit

(Paper not written)

A Multiexpert Knowledge System Architecture for Qualitative Process Theory

(Paper not provided by publication date)

Teaching Artificial Neural Systems to Drive: Manual Training Techniques for Autonomous Systems

J. F. Shepanski and S. A. Macy

TRW, Inc.
One Space Park, O2/1779
Redondo Beach, CA 90278

Abstract

We have developed a methodology for manually training autonomous control systems based on artificial neural systems (ANS). In applications where the rule set governing an expert's decisions is difficult to formulate, ANS can be used to extract rules by associating the information an expert receives with the actions he takes. Properly constructed networks imitate rules of behavior that permits them to function autonomously when they are trained on the spanning set of possible situations. This training can be provided manually, either under the direct supervision of a system trainer, or indirectly using a background mode where the network assimilates training data as the expert performs his day-to-day tasks. To demonstrate these methods we have trained an ANS network to drive a vehicle through simulated freeway traffic.

Introduction

Computational systems employing fine grained parallelism are revolutionizing the way we approach a number of long standing problems involving pattern recognition and cognitive processing. The field spans a wide variety of computational networks, from constructs emulating neural functions, to more crystalline configurations that resemble systolic arrays. Several titles are used to describe this broad area of research, we use the term artificial neural systems (ANS). Our concern in this work is the use of ANS for manually training certain types of autonomous systems where the desired rules of behavior are difficult to formulate.

Artificial neural systems consist of a number of processing elements interconnected in a weighted, user-specified fashion, the interconnection weights acting as memory for the system. Each processing element calculates an output value based on the weighted sum of its inputs. In addition, the input data is correlated with the output or desired output (specified by an instructive agent) in a training rule that is used to adjust the interconnection weights. In this way the network learns patterns or imitates rules of behavior and decision making.

The particular ANS architecture we use is a variation of Rummelhart et. al. [1] multi-layer perceptron employing the generalized delta rule (GDR). Instead of a single, multi-layer structure, our final network has a a multiple component or "block" configuration where one block's output feeds into another (see Figure 3). The training methodology we have developed is not tied to a particular training rule or architecture and should work well with alternative networks like Grossberg's adaptive resonance model[2].

The equations describing the network are [1]:

$$\text{Transfer function:} \quad o_j = (1+e^{-S_j})^{-1}, \quad S_j = \sum_{i=0}^{n} w_{ji} o_i; \tag{1}$$

$$\text{Weight adaptation rule:} \quad \Delta w_{ji} = (1-\alpha_{ji})\eta_{ji}\delta_j o_i + \alpha_{ji}\Delta w_{ji}^{\text{previous}}; \tag{2}$$

$$\text{Error calculation:} \quad \delta_j = o_j(1-o_j)\sum_{k=1}^{m}\delta_k w_{kj}, \tag{3}$$

where o_j is the output of processing element j, w_{ji} is the interconnection weight leading from element i to j, n is the number of inputs to j, Δw is the adjustment of w, η is the training constant, α is the training "momentum," δ_j is the calculated error for element j, and m is the fanout of a given element. Element zero is a constant input, equal to one, so that w_{j0} is equivalent to the bias threshold of element j. The $(1-\alpha)$ factor in equation (2) differs from standard GDR formulation, but it is useful for keeping track of the relative magnitudes of the two terms. For the network's output layer the summation in equation (3) is replaced with the difference between the desired and actual output value of element j.

These networks are usually trained by presenting the system with sets of input/output data vectors in cyclic fashion, the entire cycle of database presentation repeated dozens of times. This method is effective when the training agent is a computer operating in batch mode, but would be intolerable for a human instructor. There are two developments that will help real-time human training. The first is a more efficient incorporation of data/response patterns into a network. The second, which we are addressing in this paper, is a suitable environment wherein a man and ANS network can interact in training situation with minimum inconvenience or boredom on the human's part. The ability to systematically train networks in this fashion is extremely useful for developing certain types of expert systems including automatic signal processors, autopilots, robots and other autonomous machines. We report a number of techniques aimed at facilitating this type of training, and we propose a general method for teaching these networks.

System Development

Our work focuses on the utility of ANS for system control. It began as an application of Barto and Sutton's associative search network[3]. Although their approach was useful in a number of ways, it fell short when we tried to use it for capturing the subtleties of human decision-making. In response we shifted our emphasis from constructing goal functions for automatic learning, to methods for training networks using direct human instruction. An integral part of this is the development of suitable interfaces between humans, networks and the outside world or simulator. In this section we will report various approaches to these ends, and describe a general methodology for manually teaching ANS networks. To demonstrate these techniques we taught a network to drive a robot vehicle down a simulated highway in traffic. This application combines binary decision making and control of continuous parameters.

Initially we investigated the use of automatic learning based on goal functions[3] for training control systems. We trained a network-controlled vehicle to maintain acceptable following distances from cars ahead of it. On a graphics workstation, a one lane circular track was constructed and occupied by two vehicles: a network-controlled robot car and a pace car that varied its

speed at random.. Input data to the network consisted of the separation distance and the speed of the robot vehicle. The values of a goal function were translated into desired output for GDR training. Output controls consisted of three binary decision elements: 1) accelerate one increment of speed, 2) maintain speed, and 3) decelerate one increment of speed. At all times the desired output vector had exactly one of these three elements active. The goal function was quadratic with a minimum corresponding to the optimal following distance. Although it had no direct control over the simulation, the goal function positively or negatively reinforced the system's behavior.

The network was given complete control of the robot vehicle, and the human trainer had no influence except the ability to start and terminate training. This proved unsatisfactory because the initial system behavior--governed by random interconnection weights--was very unstable. The robot tended to run over the car in front of it before significant training occurred. By carefully halting and restarting training we achieved stable system behavior. At first the following distance maintained by the robot car oscillated as if the vehicle was attached by a spring to the pace car. This activity gradually damped. After about one thousand training steps the vehicle maintained the optimal following distance and responded quickly to changes in the pace car's speed.

Constructing composite goal functions to promote more sophisticated abilities proved difficult, even ill-defined, because there were many unspecified parameters. To generate goal functions for these abilities would be similar to conventional programming--the type of labor we want to circumvent using ANS. On the other hand, humans are adept at assessing complex situations and making decisions based on qualitative data, but their "goal functions" are difficult if not impossible to capture analytically. One attraction of ANS is that it can imitate behavior based on these elusive rules without formally specifying them. At this point we turned our efforts to manual training techniques.

The initially trained network was grafted into a larger system and augmented with additional inputs: distance and speed information on nearby pace cars in a second traffic lane, and an output control signal governing lane changes. The original network's ability to maintain a safe following distance was retained intact. This grafting procedure is one of two methods we studied for adding new abilities to an existing system. (The second, which employs a block structure, is described below.) The network remained in direct control of the robot vehicle, but a human trainer instructed it when and when not to change lanes. His commands were interpreted as the desired output and used in the GDR training algorithm. This technique, which we call coaching, proved useful and the network quickly correlated its environmental inputs with the teacher's instructions. The network became adept at changing lanes and weaving through traffic. We found that the network took on the behavior pattern of its trainer. A conservative teacher produced a timid network, while an aggressive trainer produced a network that tended to cut off other automobiles and squeeze through tight openings. Despite its success, the coaching method of training did not solve the problem of initial network instability.

The stability problem was solved by giving the trainer direct control over the simulation. The system configuration (Figure 1), allows the expert to exert control or release it to the network. During initial training the expert is in the driver's seat while the network acts the role of apprentice. It receives sensor information, predicts system commands, and compares its

predictions against the desired output (ie. the trainer's commands). Figure 2 shows the data and command flow in detail. Input data is processed through different channels and presented to the trainer and network. Where visual and audio formats are effective for humans, the network uses information in vector form. This differentiation of data presentation is a limitation of the system; removing it is a task for future research. The trainer issues control commands in accordance with his assigned task while the network takes the trainer's actions as desired system responses and correlates these with the input. We refer to this procedure as master/apprentice training, network training proceeds invisibly in the background as the expert proceeds with his day to day work. It avoids the instability problem because the network is free to make errors without the adverse consequence of throwing the operating environment into disarray.

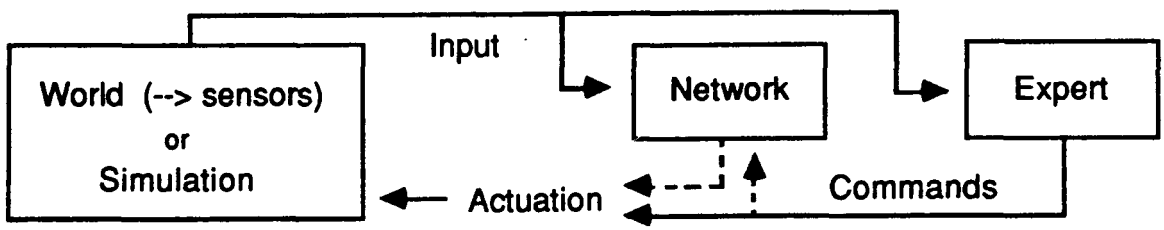


Figure 1. A scheme for manually training ANS networks. Input data is received by both the network and trainer. The trainer issues commands that are actuated (solid command line), or he coaches the network in how it ought to respond (broken command line).

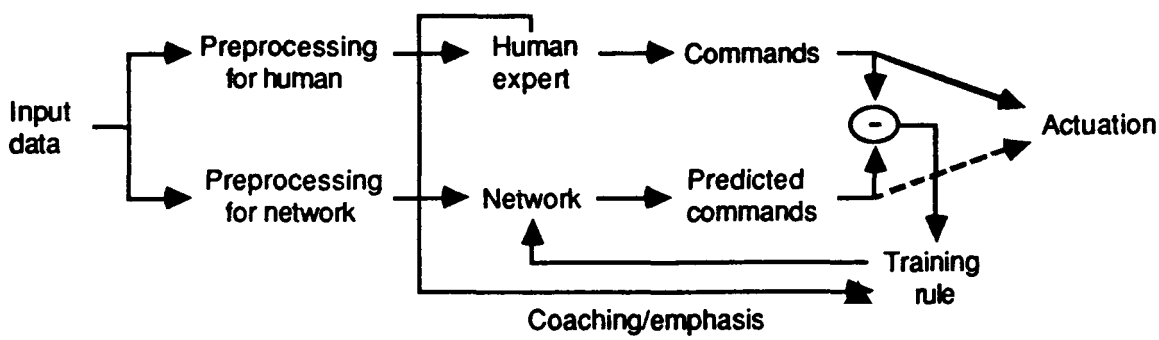


Figure 2. Data and command flow in the training system. Input data is processed and presented to the trainer and network. In master/apprentice training (solid command line), the trainer's orders are actuated and the network treats his commands as the system's desired output. In coaching, the network's predicted commands are actuated (broken command line), and the trainer influences weight adaptation by specifying the desired system output and controlling the values of training constants--his "suggestions" are not directly actuated.

Once initial, background training is complete, the expert proceeds in a more formal manner to teach the network. He releases control of the command system to the network in order to evaluate its behavior and weaknesses. He then resumes control and works through a series of scenarios designed to train the network out of its bad behavior. By switching back and forth between human and network control, the expert assesses the network's reliability and teaches correct responses as needed. We find master/apprentice training works well for behavior

involving continuous functions, like steering. On the other hand, coaching is appropriate for decision functions, like when the car ought to pass. Our methodology employs both techniques.

The Driving Network

The fully developed freeway simulation consists of a two lane highway that is made of joined straight and curved segments which vary at random in length (and curvature). Several pace cars move at random speeds near the robot vehicle. The network is given the tasks of tracking the road, negotiating curves, returning to the road if placed far afield, maintaining safe distances from the pace cars, and changing lanes when appropriate. Instead of a single multi-layer structure, the network is composed of two blocks; one controls the steering and the other regulates speed and decides when the vehicle should change lanes (Figure 3). The first block receives information about the position and speed of the robot vehicle relative to other cars in its vicinity. Its output is used to determine the automobile's speed and whether the robot should change lanes. The passing signal is converted to a lane assignment based on the car's current lane position. The second block receives the lane assignment and data pertinent to the position and orientation of the vehicle with respect to the road. The output is used to determine the steering angle of the robot car.

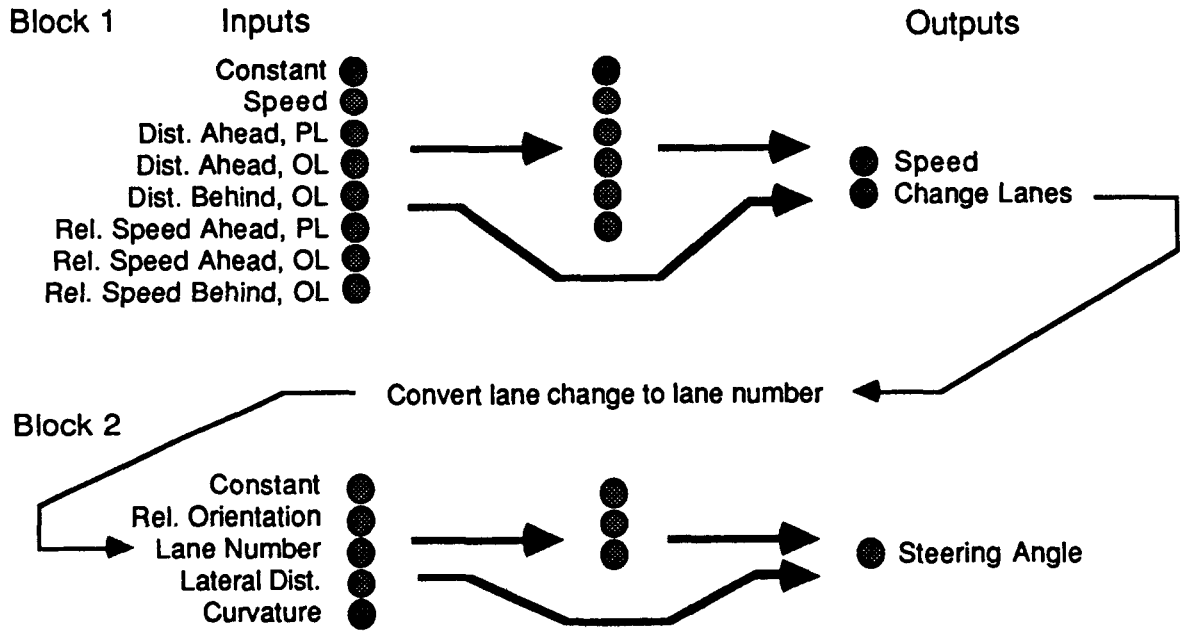


Figure 3. The two blocks of the driving ANS network. Heavy arrows indicate total interconnectivity between layers. PL designates the traffic lane presently occupied by the robot vehicle, OL refers to the other lane, curvature refers to the road, lane number is either 0 or 1, relative orientation and lateral distance refers to the robot car's direction and position relative to the road's direction and center line, respectively.

The input data is displayed in pictorial and textual form to the driving instructor. He views the road and nearby vehicles from the perspective of the driver's seat or overhead. The network receives information in the form of a vector whose elements have been scaled to unitary order,

O(1). Wide ranging input parameters, like distance, are compressed using the hyperbolic tangent or logarithmic functions. In each block, the input layer is totally interconnected to both the output and a hidden layer. Our scheme trains in real time, and as we discuss later, it trains more smoothly with a small modification of the training algorithm.

Output is interpreted in two ways: as a binary decision or as a continuously varying parameter. The first simply compares the sigmoid output against a threshold. The second scales the output to an appropriate range for its application. For example, on the steering output element, a 0.5 value is interpreted as a zero steering angle. Left and right turns of varying degrees are initiated when this output is above or below 0.5, respectively.

The network is divided into two blocks that can be trained separately. Beside being conceptually easier to understand, we find this component approach is easy to train systematically. Because each block has a restricted, well-defined set of tasks, the trainer can concentrate specifically on those functions without being concerned that other aspects of the network behavior are deteriorating.

We trained the system from bottom up, first teaching the network to stay on the road, negotiate curves, change lanes, and how to return if the vehicle strayed off the highway. Block 2, responsible for steering, learned these skills in a few minutes using the master/apprentice mode. It tended to steer more slowly than a human but further training progressively improved its responsiveness.

We experimented with different training constants and "momentum" values. Large η values, about 1, caused weights to change too coarsely. η values an order of magnitude smaller worked well. We found no advantage in using momentum for this method of training, in fact, the system responded about three times more slowly when $\alpha = 0.9$ than when the momentum term was dropped. Our standard training parameters were $\eta = 0.2$, and $\alpha = 0.0$.

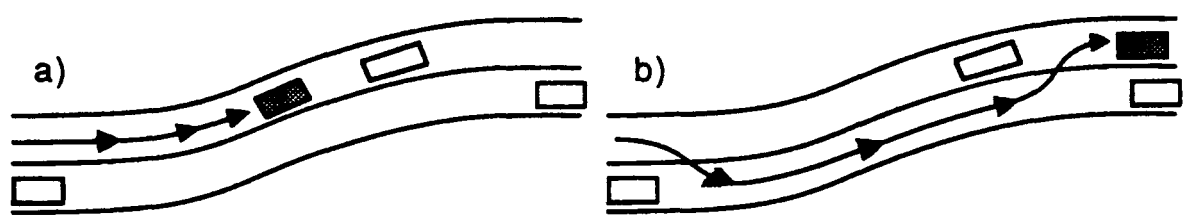


Figure 4. Typical behavior of a network-controlled vehicle (dark rectangle) when trained by a) a conservative driver, and b) a reckless driver. Speed is indicated by the length of the arrows.

After Block 2 was trained, we gave steering control to the network and concentrated on teaching the network to change lanes and adjust speed. Speed control in this case was a continuous variable and was best taught using master/apprentice training. On the other hand, the binary decision to change lanes was best taught by coaching. About ten minutes of training were needed to teach the network to weave through traffic. We found that the network readily adapts the behavioral pattern of its trainer. A conservative trainer generated a network that hardly ever passed, while an aggressive trainer produced a network that drove recklessly and tended to cut off other cars (Figure 4).

Discussion

One of the strengths of expert systems based on ANS is that the use of input data in the decision making and control process does not have to be specified. The network adapts its internal weights to conform to input/output correlations it discovers. It is important, however, that data used by the human expert is also available to the network. The different processing of sensor data for man and network may have important consequences, key information may be presented to the man but not the machine.

This difference in data processing is particularly worrisome for image data where human ability to extract detail is vastly superior to our automatic image processing capabilities. Though we would not require an image processing system to understand images, it would have to extract relevant information from cluttered backgrounds. Until we have sufficiently sophisticated algorithms or networks to do this, our efforts at constructing expert systems which handle image data are handicapped.

Scaling input data to the unitary order of magnitude is important for training stability. This is evident from equations (1) and (2). The sigmoid transfer function ranges from 0.1 to 0.9 in approximately four units, that is, over an O(1) domain. If system response must change in reaction to a large, O(n) swing of a given input parameter, the weight associated with that input will be trained toward an O(n^{-1}) magnitude. On the other hand, if the same system responds to an input whose range is O(1), its associated weight will also be O(1). The weight adjustment equation does not recognize differences in weight magnitude, therefore relatively small weights will undergo wild magnitude adjustments and converge weakly. On the other hand, if all input parameters are of the same magnitude their associated weights will reflect this and the training constant can be adjusted for gentle weight convergence. Because the output of hidden units are constrained between zero and one, O(1) is a good target range for input parameters. Both the hyperbolic tangent and logarithmic functions are useful for scaling wide ranging inputs. A useful form of the latter is

$$x' = \begin{cases} \beta[1+\ln(x/\alpha)] & \text{if } \alpha < x, \\ \beta x/\alpha & \text{if } -\alpha \leq x \leq \alpha, \\ -\beta[1+\ln(-x/\alpha)] & \text{if } x < -\alpha, \end{cases} \tag{4}$$

where $\alpha > 0$ and defines the limits of the intermediate linear section, and β is a scaling factor. This symmetric logarithmic function is continuous in its first derivative, and useful when network behavior should change slowly as a parameter increases without bound. On the other hand, if the system should approach a limiting behavior, the tanh function is appropriate.

Weight adaptation is also complicated by relaxing the common practice of restricting interconnections to adjacent layers. Equation (3) shows that the calculated error for a hidden layer-given comparable weights, fanouts and output errors-will be one quarter or less than that of the output layer. This is caused by the slope factor, $o_i(1-o_i)$. The difference in error magnitudes is not noticeable in networks restricted to adjacent layer interconnectivity. But when this constraint is released the effect of errors originating directly from an output unit has 4^d times the magnitude and effect of an error originating from a hidden unit removed d layers from the output layer.

Compared to the corrections arising from the output units, those from the hidden units have little influence on weight adjustment, and the power of a multilayer structure is weakened. The system will train if we restrict connections to adjacent layers, but it trains slowly. To compensate for this effect we attenuate the error magnitudes originating from the output layer by the above factor. This heuristic procedure works well and facilitates smooth learning.

Though we have made progress in real-time learning systems using GDR, compared to humans-who can learn from a single data presentation-they remain relatively sluggish in learning and response rates. We are interested in improvements of the GDR algorithm or alternative architectures that facilitate one-shot or rapid learning. In the latter case we are considering Hecht-Nielsen's counterpropagation[4] and Grossberg and Carpenter's adaptive resonance models[3,5].

The construction of automated expert systems by observation of human personnel is attractive because of its efficient use of the expert's time and effort. Though the classic AI approach of rule base inference is applicable when such rules are clear cut and well organized, too often a human expert can not put his decision making process in words or specify the values of parameters that influence him. The attraction of ANS based systems is that imitations of expert behavior emerge as a natural consequence of their training.

References

1) D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Internal Representations by Error Propagation," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. I,* D. E. Rumelhart and J. L. McClelland (Eds.), chap. 8, (1986), Bradford Books/MIT Press, Cambridge

2) S. Grossberg, *Studies of Mind and Brain,* (1982), Reidel, Boston

3) A. Barto and R. Sutton, "Landmark Learning: An Illustration of Associative Search," *Biological Cybernetics,***42**, (1981), p. 1

4) R. Hecht-Nielsen, "Counterpropagation Networks," *Proceedings of the IEEE 1st Annual International Conference on Neural Networks,* San Diego, June 21-24, 1987

5) G. A. Carpenter and S. Grossberg, "A Massively Parallel Architecture for a Self-organizing Neural Pattern Recognition Machine," *Computer Vision, Graphics and Image Processing,* **37**, (1987), p.54

DESIGN KNOWLEDGE CAPTURE FOR THE SPACE STATION

K. R. Crouse
NASA/Johnson Space Center
Mail Code EF5, Houston, Texas 77058

D. B. Wechsler
The MITRE Corporation
1120 NASA Road One
Houston, Texas 77058

INTRODUCTION

NASA's Space Station Program (SSP) is approaching its System Design Review milestone. NASA's SSP development community is preparing to begin detailed design; to be followed by testing, evaluations, and launch. The first elements of the Space Station are planned for launch in 1994. The development environment for generation and capture of design information over this extended period will be characterized by:

a) Continuing engineering efforts resulting in many changes.

b) Personnel phasing in and out of the program.

c) Technology developments bringing opportunities to apply design information in powerful and hardly-imagined ways.

The challenge of design knowledge capture will be to create and populate a base of design knowledge, to provide a sufficient foundation of collective technical memory for support of applications of the year 2000 and beyond. NASA and The MITRE Corporation have identified a technical approach and implementation plan for the capture and storage of design knowledge.

NEEDS AND BENEFITS

The capture, storage, and availability of design knowledge can benefit the Space Station Program throughout its life cycle.

Continuing Engineering and Future Designs

The importance of integration through a common database is emphasized in "team engineering" approaches to reducing product design lead time. In a team engineering environment, the work of an individual will affect many others on a project. As the size of the team increases, cooperative interaction leads to productivity improvement, while standalone solutions reach a point of diminishing returns, due to the overhead costs of handling redundant information. As the team development of the Space Station progresses, increasingly complex engineering models will be exercised with "what-if" simulations. If root design values and inputs are retained, together with a definition of the engineering analysis descriptions, then the same or similar analysis types can be reproduced using different parameters. If the same analysis program is available, the original analysis can be re-created.

Captured and retained design knowledge can also provide a basis for solving new design problems, by using design rationale to replay design histories of similar problems. If a goal in the old problem was met using some plan, and the reasons the old design worked also hold true in the new problem; then the plan can assist in determining the new solutions as well.

Manufacturing

The integration benefits of design knowledge capture are embodied in the term "CIM" (Computer-Integrated Manufacturing), adopted in principle by industrial corporations worldwide. The CIM concept involves intelligent combination and use of "... computer and information/communication technologies to effectively integrate all of:

a) the engineering/design functions,
b) the manufacturing planning functions,
c) the equipment/process technologies,
d) the manufacturing control processes, and
e) the management functions

necessary to convert raw materials, labor, energy, and information into a high quality, profitable product, within a reasonable amount of time." [16]

The Corporate CIM Committee at Garrett Corporation has observed that the manufacturing organizations operating in 1995 and beyond will be fundamentally different from today's typical manufacturing companies. CIM technologies will make well-defined long-range CIM plans essential to the maintenance of competitive position.

Logistics and Field Operations

In its introduction to the Integrated Design Support Project, the U. S. Air Force Logistics Command reports: [14]

> "In its wake, high technology has created massive amounts of technical information-- a mountain of paper which today must be managed manually... engineering technical data is volatile, complex, iterative, and addressable by a variety of applications. These special requirements make manual data-handling extremely labor- and cost-intensive. To date, application has been focused on design and manufacturing, with no consideration being given to integrating the overall... life cycle process."

On-Board Applications

The initial findings of NASA's Automation and Robotics Panel stated that: [3]

> "... Shuttle operators rely heavily on paper backup for every mission. This mass of documents must be condensed, coordinated, and unified into a usable database if the Space Station is to reach its planned level of capability."

When the actual behavior of a system fails to match its intended behavior, the reason for failure is more easily localized if a record is available of how the system specification was decomposed and implemented. However, this debugging cannot be conducted using the design definition only. The designer's knowledge must also be applied.

CURRENT STATE

Capture is the process of obtaining information for retention in computer-interpretable form. Captured information is organized in meaningful context for retention and use. The definition of capture does not include information held in manual media. Neither does it include data entered into storage without association of meaning, such as a scanned code, or unidentified text string. Additional conversion or interpretation of these data forms is required.

With increased use of CAD/CAM, the electronic retention of design definitions is on the increase. But near-term emphasis of Computer-Aided Design (CAD) as a drafting tool can divert attention from the need to effectively organize design definitions. Chester Fleszar, an Applicon Marketing Product Specialist, elaborates: [11]

> "The problem is that we act as if we're making blueprints rather than parts. Once a company becomes involved in CAD/CAM, the electronic representation of the part becomes all-important, while the blueprint becomes obsolete. Translating all the information on the original paper drawing does nothing to improve manufacturing quality or efficiency..."

Manufacturing industry has recognized the benefit of integrating conventional CAD/CAM applications through design definition information in engineering and product databases. But for planning and logistics organizations, field operators, and customers, the design definition alone is inadequate. Users of this information must adjust to the information shortfall by attempting ad hoc to manipulate the available information, or by expending extra resources to collect the needed information.

Nevertheless, efforts to capture the designer's knowledge are rare. For the NASA community, unless the knowledge of SSP designers and engineers is captured, the SSP collective technical memory expected to be available will diminish with the development team's decreasing accessibility over time.

DESIGN KNOWLEDGE REPRESENTATION

Overview

Design knowledge encompasses not only what designs are, but how and why they satisfy the design's functional requirement. Design knowledge is represented as a linked design object structure. This "object-organized" structure is composed of design objects, object attributes, and declarations or

assertions called "designer's knowledge".

Design Object Structures

The principal unit of design knowledge organization is the design object. The linkage of design objects defines the design arrangement. Design objects are defined over a range of abstraction levels, in which each object is linked with its constituents. For example, a representation for a physical design object of a component assembly is decomposed to the constituent components. The components are decomposed to constituent features of each component. The features can be implemented as graphical elements, if the feature has a visual interpretation.

The definitions and values of each object's attributes are contained within that object's structure.

Graphics of the Design Object

The U. S. CAD/CAM community has in cooperation defined a computing system-independent means of exchanging CAD graphics files. This evolutionary standards development, coordinated by the National Bureau of Standards, is called Initial Graphics Exchange Specification (IGES). [13]

IGES defines standard data structures called entities, for geometric and other graphics-based elements. A graphics system that supports IGES can translate an IGES data structure into a geometric pictorial.

The IGES data structures corresponding with the design object's geometric elements are represented as attributes of the object. This approach allows a design object possessing this information to "draw itself", using methods including IGES translation by the CAD delivery system.

Designer's Knowledge

Design objects are the parent representations for designer's knowledge. Designer's knowledge attaches to its parent object at the highest applicable level of abstraction.

Designers' knowledge includes the information the designers used; the analysis they conducted; and the decisions they made to develop the design object. Designers' knowledge defines what the design does, and why it does so. Such definition encompasses functional and behavioral descriptions.

Examples of designer's knowledge include declarations of functional requirements for the design object; criteria or intent for selection of a particular design approach or solution over its alternatives; declarations of analysis results and conclusions; and assertions of expected behavior in normal, marginal or failed modes.

BOUNDING OF DESIGN KNOWLEDGE

The potential size of a comprehensive SSP design knowledge aggregation suggests that approaches must be defined to avoid the capture of extraneous knowledge. The three following approaches to knowledge organization and bounding have been identified.

Bounding of Knowledge Content

The "perspective" parameter is a classification based on the use of design knowledge. Users' perspectives are defined in terms of application problems. However, since designers may both produce and use design knowledge, their analytic disciplines are also users' perspectives. Thus, a perspective could be based on either an intermediate configuration, such as a model for design analysis; or on a "flying" design, as in an on-board SSP application.

Definitions of perspective provide for the subsequent retrieval of design knowledge, by requiring that only knowledge be captured for which a perspective can be identified. Since complex future applications might involve several disciplines, perspectives can also serve to clarify the boundaries of expert knowledge in multi-discipline problem-solving.

Bounding of Knowledge Volume

The "visibility" parameter of a design object is an indicator for determining the depth of knowledge detail to be captured, and for selecting the appropriate capture tool.

Visibility is a combined valuation of the probability of failure within the design object, factored with the results of such failure. Valuations of visibility can be taken from reliability and redundancy projections. For example, if a design object (including its designed redundancies) has a high reliability and negligible results from failure, then this object will have a low visibility rating.

Bounding of Capture Frequency

The "version" parameter enables temporal support to be established in the design knowledge structure. This parameter can

be used to manage multiple knowledge versions of the same design object. Knowledge which affects a temporal value will require an accurate accounting of changes and their rationale.

THE DESIGN KNOWLEDGE SYSTEM

In the approach identified, captured knowledge will be stored in relational databases. Since applications based on advanced technologies such as object oriented programming and database inferencing are not yet in wide use, the interim step of retaining design knowledge in a relational database "object-organized" form will assure the availability of SSP design knowledge for these future application technologies.

The facilities of the database manager may be used as a capture tool. Descriptions of additional capture tools follow.

CAD and Engineering Analysis

The integration of Computer-Aided Design (CAD) and Computer-Aided Engineering (CAE) provides a common view of the design, to facilitate reasoning and analysis about the design object. CAD provides for automation of the graphical representation process. CAE provides for automation of those engineering analysis methods used to predict the behavior of the design object. The CAE analysis perspective may be defined as a knowledge bounding parameter.

CAD/CAE Documentation

Basic documentation can be generated from within the CAD system. Attribute values may be entered when the pictorials of the design objects are created. Then the resultant information is extracted and loaded into the relational database system.

With integrated documentation, the results of the CAE analysis can be included with the CAD information. The analysis program is then easily identified, and the program itself can be referenced for future use.

Specification Language System

A Specification Language System (SLS) is a specification environment based on a formal notation for expressing design requirements in terms of function, structure, or behavioral description. The methodology of the SLS provides an organized approach for capturing design knowledge early in the development cycle, when insufficient system design definition exists to apply CAD approaches. The automated tool set of an SLS can provide valuable assistance in assuring consistency of identifiers and terms, and in enforcing documentation and project standards.

Where well-defined relationships exist between the functional and physical definitions, certain SLS's synthesize and simulate physical structure from a functional definition. An example in VLSI circuitry is the VHSIC Hardware Description Language (VHDL). [1] Executable hardware description languages should be considered only for well-structured design problems with unambiguous physical implications.

Designer's Apprentice System

The designer's apprentice can capture design knowledge as a by-product of its interaction with the designer. A designer's apprentice may perform the following functions:

- Suggestion of goals and constraints
- Recognition of past successful solutions
- Conducting and recording of system-designer dialogues
- Assistance with tedious details

While the notion of a designer's apprentice holds promise; presently the necessary organization of design knowledge is barely understood well enough to effectively apply this tool. The strongest candidate areas are narrow domains of expertise having codified design rationale.

TECHNOLOGY FOR FUTURE KNOWLEDGE ACCESS

Accommodating Knowledge Access Technologies

Applications based on the following technologies are not in wide use. For an interim period, captured design knowledge will be arranged by object, and retained in relational storage. Advanced applications, as they are developed, will be supported with the captured SSP design knowledge, made compatible through economically tolerable modification.

Object Oriented Environment

Object-oriented technology complements design knowledge organization by characterizing systems in terms of a configuration. This approach centers descriptions around the objects that are pieced together, rather than centering on transformations of data about these systems. This organization is similar to

the linked design object structures in the database system.

Object-oriented programming holds promise for acceptance because of claims for improved programmer productivity and easier program maintenance. More important from a user viewpoint, the program organization allows those not initially familiar with the program to rapidly and accurately understand its content.

An object consists of data private to the object, and of a set of operations which can access that private data. A "consumer" object requests a "provider" object to perform one of its operations, by sending it a message telling it what to do. The provider object responds by choosing an appropriate method; executing the operation; and returning control to the consumer.

Object-oriented programming systems are now evolving into complete development platforms, including both language and database features. Commercial products are beginning to emerge.

Database-Inferencing Systems

Bridging is needed between expert system knowledge bases and database management systems.

A promising current approach involves schema translation. In comparing programming language commands with database operators and query commands, researchers have developed a mapping of schema between the two.

Schema translation may lead to development of database-inferencing systems which share schema. These systems would use a common database for a number of knowledge-based applications. The inferencing procedure would be integrated with the database management capabilities. Then application development would consist of developing the appropriate goal statements, and confirming that the supporting descriptions are in the database.

Another approach to schema translation is to locate the translation intelligence within the intelligent system development facility. In this approach, the knowledge-based application initiates a database query.

FUTURE TASKS AND ISSUES

Overview

The project of developing a design knowledge capture system involves substantial planning and preparation. A serious implementation issue is the coordination of timing between capture system installation and Space Station development. Until capture is implemented, the risk of losing designer's knowledge is ongoing.

Many of the facilities for design knowledge will be provided as part of NASA's Technical Information Management System (TMIS). The TMIS will be used to support technical management functions of the overall Space Station Program, including the design, development, and operation of the orbital facility. The TMIS user community will include all NASA personnel involved with the Space Station, all primary contractor personnel, and all personnel representing the international partners. The TMIS resources will be based on commercial,"off-the-shelf" technology.

Following are major near-term tasks for implementation of design knowledge capture, related to computing facilities.

Relational database facility

A TMIS-compatible relational database facility will be employed as the development contractors' design knowledge repository. Adequate description of this facility will be provided in ample time to allow for development contractors' knowledge capture planning.

Standardization Issues

Standardization issues will be resolved, which arise from the resources to be provided. Such issues include common methods for CAD data exchange.

Following are major near-term tasks for implementation of design knowledge capture, related to knowledge organization.

Schemes for Knowledge Bounding

To support the development contractors' planning for capture resources and methods, initial valuations of visibility parameters will be provided for identified design objects. A classification of engineering analysis perspectives will be supplied. A common contractor approach for implementation of knowledge versioning will be defined.

Design Knowledge Content

The design knowledge base must be defined and organized, before it can be populated. Guidelines will be established for common semantics and input definitions. Available application developers will assist by providing requirements.

Existing NASA databases will be evaluated for compatibility with requirements for design knowledge content. Conforming portions will be integrated within a design knowledge context.

An evaluation of the planned content of future milestone deliverables will also be conducted, for suitability as design knowledge.

CONCLUSIONS

The benefits of design knowledge availability are identifiable and pervasive. The implementation of design knowledge capture and storage using current technology increases the probability for success, while providing for a degree of access compatibility with future applications. The Space Station design definition should be expanded to include design knowledge. Design knowledge should be captured. A critical timing relationship exists between the Space Station development program, and the implementation of this project.

REFERENCES

1. Ackley, Dave, et al., "VHSIC Hardware Description Language," IEEE COMPUTER, February, 1985.

2. Addleman, David K., Davis, Margaret J., and Presson, Edward P., "Specification Technology Guidebook," CONTRACT F30602-84-C-0073, U. S. Air Force, August, 1985.

3. Advanced Technology Advisory Committee, ADVANCING AUTOMATION AND ROBOTICS TECHNOLOGY FOR THE SPACE STATION AND FOR THE U. S. ECONOMY, NASA TECHNICAL MEMORANDUM 87566, April 1, 1985.

4. Anderson, Sandra V., Bell, John B., and Smylie, Susan E., INFORMATION ASSET MANAGEMENT, VOLUME IV, The MITRE Corporation Technical Report Number 8613, June, 1986.

5. Andrews, Timothy and Harris, Craig, "Combining Language and Database Advances in an Object-Oriented Development Environment," (To Be Published) OOPSLA '87 PROCEEDINGS, October, 1987.

6. Beazley, William G., "Available Expert System Knowledge in CAD/CAM Databases," CONTRACT F19628-86-C-0001, The MITRE Corporation, July, 1986.

7. Brown, D. C. and Chandrasekaran, B., "Expert Systems for a Class of Mechanical Design Activity," WORKING CONFERENCE ON KNOWLEDGE ENGINEERING IN COMPUTER-AIDED DESIGN, Elsevier Science Publishing Company, 1984.

8. Clancey, William J., "Heuristic Classification," ARTIFICIAL INTELLIGENCE JOURNAL, December, 1985.

9. Cox, Brad J., OBJECT-ORIENTED PROGRAMMING, Addison-Wesley Publishing Company, 1986.

10. Crouse, K. J. and Spitzer, J. F., "Design Knowledge Bases for the Space Station," PROCEEDINGS OF ROBEXS '86, June, 1986.

11. Fleszar, Chester, "Is IGES the Problem, or are We?," CIM TECHNOLOGY, Fall, 1986.

12. Genesereth, Michael R., "The Use of Design Descriptions in Automated Diagnosis," QUALITATIVE REASONING ABOUT PHYSICAL SYSTEMS, The MIT Press 1985.

13. IGES Organization, INITIAL GRAPHICS EXCHANGE SPECIFICATION- VERSION 3.0, National Bureau of Standards, April, 1986.

14. INTEGRATED DESIGN SUPPORT, U. S. Air Force Wright Aeronautical Laboratories, 1986.

15. Maier, David, et al., "Development of an Object-Oriented DBMS," OOPSLA '86 PROCEEDINGS, September, 1986.

16. Mize, Joe H., Seifort, Deborah J., and Settles, Stan F., "Formation and Workings of a Corporate-Wide CIM Committee at Garrett Corporation," INDUSTRIAL ENGINEERING, November, 1985.

17. Mostow, Jack, "Towards Better Models of the Design Process," AI MAGAZINE, Spring, 1985.

18. Rehak, Daniel R., Howard, H. Craig, and Sriram, Duvvuru, "Architecture of an Integrated Knowledge-Based Environment for Structural Engineering Applications," WORKING CONFERENCE ON KNOWLEDGE ENGINEERING IN COMPUTER-AIDED DESIGN, Elsevier Science Publishing Company, 1984.

19. Rosenfeld, Lawrence W. and Belzer, Avrum P., "Breaking Through the Complexity Barrier--ICAD," PROCEEDINGS OF AUTOFACT '85, November, 1985.

20. Steinke, George C. and Schussel, Martin D., "Engineering by the Book... and On-Line," MECHANICAL ENGINEERING, November, 1985.

21. Snodgrass, Richard and Ahn, Ilsoo, "Temporal Databases," IEEE COMPUTER, September, 1986.

22. Stefik, Mark and Bobrow, Daniel, "Object-Oriented Programming: Themes and Variations," AI MAGAZINE, Winter, 1986.

23. Sowa, J. F., CONCEPTUAL STRUCTURES: INFORMATION PROCESSING IN MIND AND MACHINE, Addison-Wesley Publishing Company, 1984.

24. Sullivan, Jerry S. and Rummler, David C., "Second-Generation CAE Tools Learn to Share One Database," ELECTRONIC DESIGN, July 10, 1986.

25. Walker, Robert A. and Thomas, Donald E., "A Model of Design Representation and Synthesis," PROCEEDINGS OF THE IEEE 22ND DESIGN AUTOMATION CONFERENCE, 1985.

CONFIG: QUALITATIVE SIMULATION TOOL FOR ANALYZING BEHAVIOR OF ENGINEERED DEVICES

Jane T. Malin and Bryan D. Basham
Systems Development and Simulation Div.
Engineering Directorate
NASA - Lyndon B. Johnson Space Center
Houston, TX 77058

Richard A. Harris
MITRE Corporation
1120 NASA Rd. 1
Houston, TX 77058

ABSTRACT

To design failure management expert systems, engineers mentally analyze the effects of failures and procedures as they propagate through device configurations. CONFIG is a generic device modelling tool for use in discrete event simulation, to support such analyses. CONFIG permits graphical modeling of device configurations and qualitative specification of local operating modes of device components. Computation requirements are reduced by focussing the level of component description on operating modes and failure modes, and specifying qualitative ranges of variables relative to mode transition boundaries. A time-step approach is avoided, and simulation processing occurs only when modes change or variables cross qualitative boundaries. Device models are built graphically, using components from libraries. Components are connected at ports by graphical relations that define data flow. The core of a component model is its state-transition diagram, which specifies modes of operation and transitions among them. Process statements describe state transitions and within-state processing, and have three parts: invocations (preconditions for effects execution), effects, and delays for each effect. A process language supports writing statements with several qualitative and quantitative syntaxes, including table lookups. CONFIG has been used to build device models in two domains, digital circuits and thermal systems.

INTRODUCTION

Designing, testing, and operating engineered devices requires analysis of the effects of failures and procedures as they propagate through device configurations. Such analysis is required in development of failure management expert systems, in failure modes and effects analysis (FMEA), and in procedures development. Information about device configuration and operating modes is used to predict effects of local changes in components on the device as a whole, and to plan how to diagnose failures and recover from them [1]. Early in design and testing, many of these analyses are performed mentally by engineers on the basis of qualitative information.

The purpose of the CONFIG project is to develop a generic device modeling tool to support commonsense analysis of system behavior by designers and operators. The tool should permit engineers to graphically model device configuration (components and connections) and qualitatively specify local operating modes of components. The tool should provide for integration of models from multiple domains, e.g., electrochemical/thermal processing and digital circuits.

The CONFIG project approach has been to develop both a CONFIG prototype and a prototyping environment. The approach to the CONFIG tool has been developed to support rapid informal analysis early in system design, as well as later precise analysis. The tool permits modeling of component modes and processes both qualitatively and quantitatively. The tool permits graphical modeling of component configurations. The tool is built on a discrete event simulator, and propagates discrete change events through device configurations. The CONFIG project prototyping environment provides flexible object-oriented modeling capabilities and a language constructor that supports experimenting with various qualitative and quantitative representations of device information.

COMPONENT LIBRARIES AND GRAPHICAL MODELING

The CONFIG tool should be designed for ease of use by engineers and operators. As in the work of Towne et al [2], the tool supports the development of graphical libraries of component models, and permits an author to build device models graphically. A model is built by using component objects from a library, and connecting them at ports with graphical relations that define data flow between components. Libraries also define classes of processes and the process language for the author. This aspect of the CONFIG tool uses the Simkit discrete event simulation tool. An example of a thermal model

and some component models and relations from its associated library are shown in Figure 1.

COMPONENT DEFINITION

Computation and specification requirements are reduced by focussing the level of component description on operating modes and failure modes, and specifying qualitative ranges of component variables relative to mode transition boundaries.

A component model can be viewed both as a composite of modes and as a composite of ports. The core of a component model is its state-transition diagram, which graphically specifies modes of operation (both normal and failed) and the transitions among them. State transitions and within-state variable transformations are specified as processes. The other decomposition of a component model is into its ports. Ports designate component inputs and outputs. Relations connect ports for data propagation between components. Examples of both types of decomposition are shown in Figure 2. The boxed area on the right of the figure is a decomposition of the boxed area in Figure 1.

DISCRETE EVENT SIMULATION APPROACH

Using an approach similar to Pan [3], discrete events are defined at the level of changes in operating modes. A time-step approach is avoided, and simulation processing need occur only when modes change or variables cross qualitative boundaries.

Rather than constraint propagation, discrete event processes determine the consequences of component changes. The event structure controls the propagation of behavior changes among components. Scheduled events pass data between ports, change variable values, and make state transitions. The primary event is the update of a component, which is triggered by a change in an input variable, local variable, or component state. In such an event, appropriate processes are inspected, and the effects of invoked processes are scheduled with corresponding delays. Updates originating from many components can be scheduled at the same time on the discrete event clock.

PROCESSES AND PROCESS LANGUAGE CONSTRUCTOR

There are three kinds of processes, mode independent processes, mode dependent processes, and mode transition processes (which are actually also mode dependent processes). Processes consist of three parts: invocations (preconditions for effects execution), effects (executed if all invocations are satisfied), and delays corresponding to each effect (effect completions scheduled at increments to the current time). Invocations and effects are defined in terms of variables, modes, and processes.

A key capability is a process language constructor and interpreter that permits process statements to be written with an array of qualitative and quantitative syntaxes, including table lookups. A process language interprets statements that define invocations and effects of processes. The language constructor supports experimenting with representations, by permitting the definition of data-structure types and operators.

Component variables can be specified quantitatively or qualitatively, but a small number of qualitative ranges is desirable (e.g., abnormal-low, low, medium, high, abnormal-high). Continuous behavior is partitioned into trends and breakpoints. The durations of trends can be represented "qualitatively" as an approximate order of magnitude, which is translated into an interval on the discrete event clock. Trend effects are represented by scheduling a process for the end of a trend, at which time the duration of the trend is confirmed before executing the effect.

Examples of processes, data structures and operators from the thermal library are shown in Figures 3 and 4.

CONCLUSION

CONFIG has been used to build device models in two domains, digital circuits and thermal systems. The CONFIG project is ongoing, with plans for additional capabilities beyond the current modeling capabilities. These additional capabilities will include support for analysis of failure effects, and support for development of differential diagnostic measures and tests. An additional goal is to support incremental development of quantitative process simulation algorithms.

ACKNOWLEDGEMENT

The authors wish to thank Jodi Seaborn for developing the thermal library and thermal bus model using CONFIG.

REFERENCES

1. Malin, J. T., and Lance, N., "Processes in Construction of Failure Management Expert Systems from Device Design Information," IEEE TRANS. SYSTEMS, MAN, AND CYBERNETICS, in press.

2. Towne, D. M., Munro, A., Pizzini, Q. A., and Surmon, D. S., "Representing System Behaviors and Expert Behaviors for Intelligent Tutoring," TECH. REPORT NO. 108, Univ. So. Calif. Behavioral Technology Laboratories, Redondo Beach, CA, February, 1987.

3. Pan, J. Y., "Qualitative Reasoning With Deep-level Mechanism Models for Diagnoses of Mechanism Failures," PROC. FIRST CONFERENCE ART. INT. APPLICATIONS, Denver, CO, December, 1984, pp. 295-301.

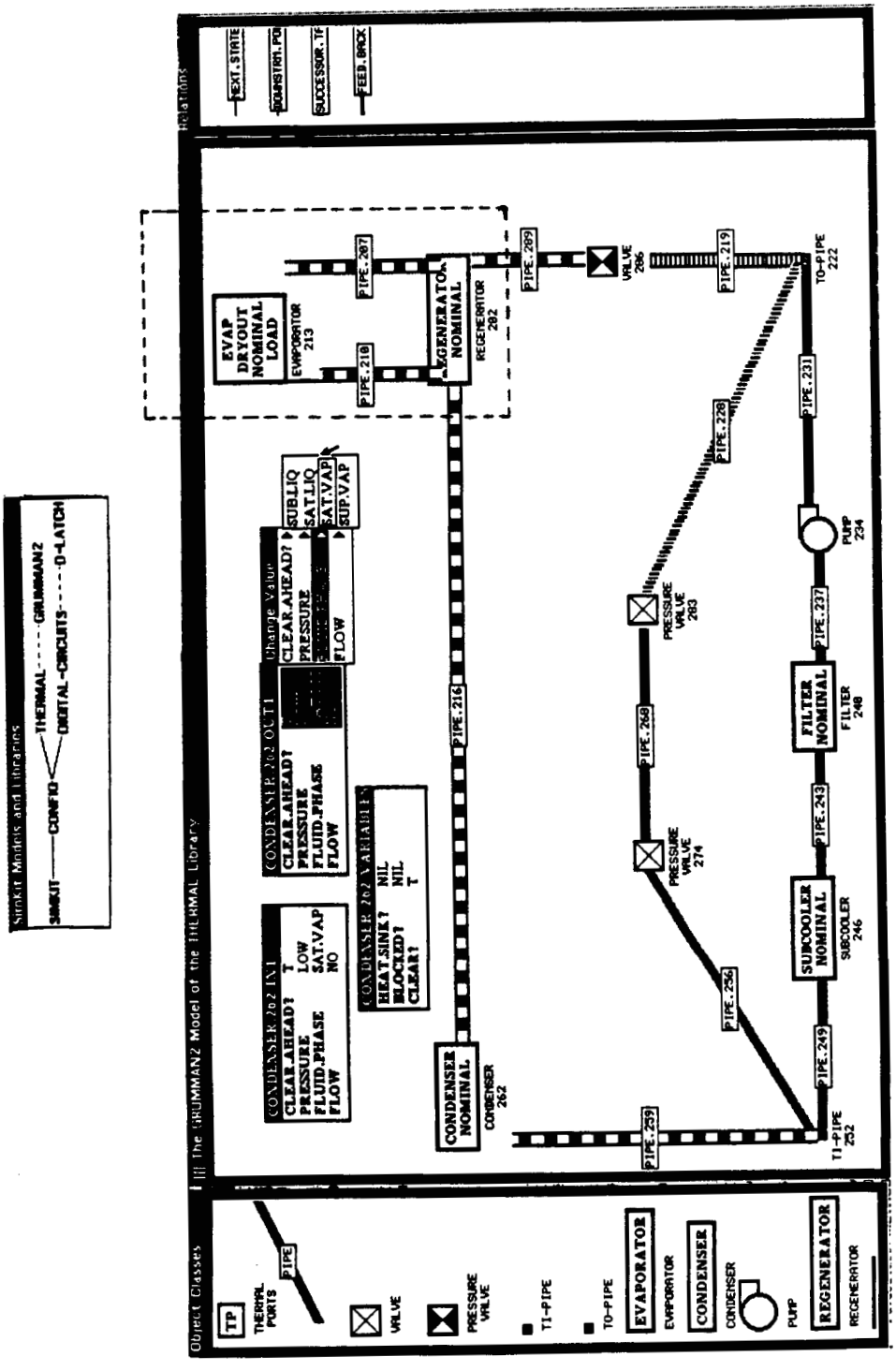
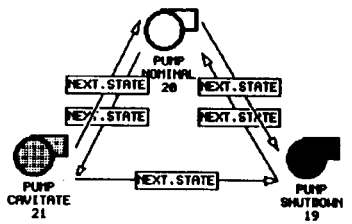
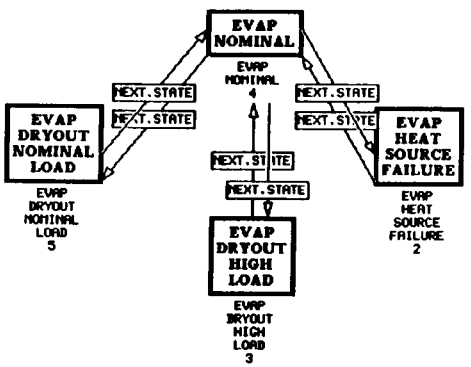


Figure 1 - Thermal Library and Grumman Bus Model

State Diagram for Pumps



State Diagram for Evaporators



Decomposition of Evaporator.213, Pipe.207 Regenerator.202, and Pipe.209

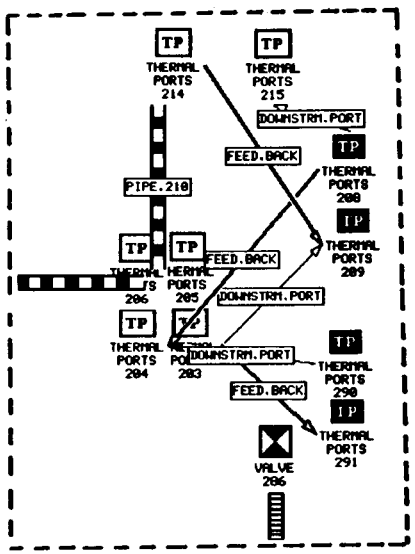
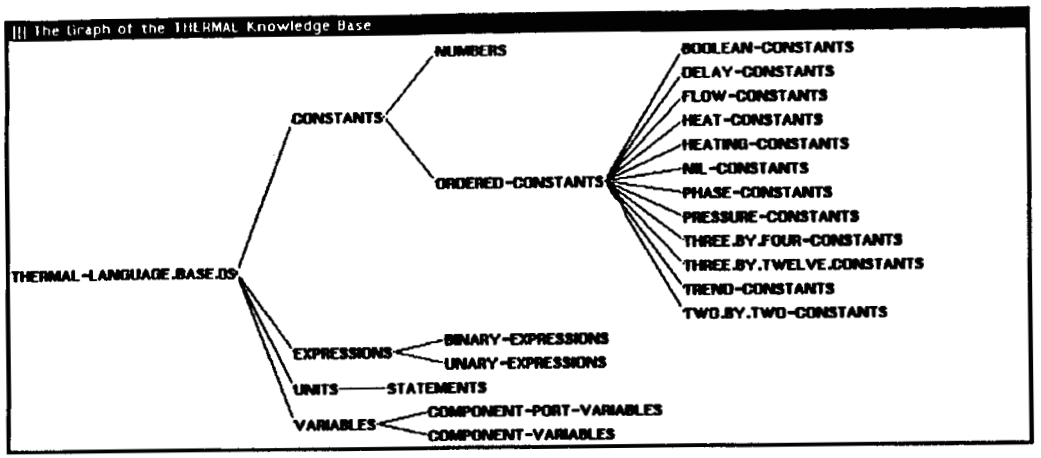


Figure 2 - Thermal Component Decompositions



Process with Delays

||| (Output) the CAVITATE.TO.SHUTDOWN.TREND.FAST Unit in THERMA

Member slot: **DELAYS** from **CAVITATE.TO.SHUTDOWN.TREND.FAST**
Inheritance: **OVERRIDE.VALUES**
ValueClass: (UNION **NUMBER SYMBOL**)
Values: SECS, 0

Member slot: **EFFECTS** from **CAVITATE.TO.SHUTDOWN.TREND.FAST**
Inheritance: **OVERRIDE.VALUES**
ValueClass: **LANGUAGE-CLASSES** in **PROCESS-LANGUAGE**
Avunits: (**CREATE-STATEMENTS** in **PROCESS-LANGUAGE** ALL NIL)
Values: **CAVITATE.TO.SHUTDOWN.TREND.EFFECT.PROCESS**,
CAVITATE.TO.SHUTDOWN.TREND.FAST.EFFECT

Member slot: **EVALUATE!** from **PROCESSES-CLASS**
Inheritance: **METHOD**
ValueClass: **METHOD**
Comment: This method evaluates this process by calling !EXECUTE if !INVOKE.P is satisfied.
Values: PROCESSES-EVALUATE

Member slot: **INVOCATIONS** from **CAVITATE.TO.SHUTDOWN.TREND.FAST**
Inheritance: **OVERRIDE.VALUES**
ValueClass: **STATEMENTS-CLASS** in **PROCESS-LANGUAGE**
Avunits: (**CREATE-STATEMENTS** in **PROCESS-LANGUAGE** ALL NIL)
Values: UNKNOWN

Delay Constants

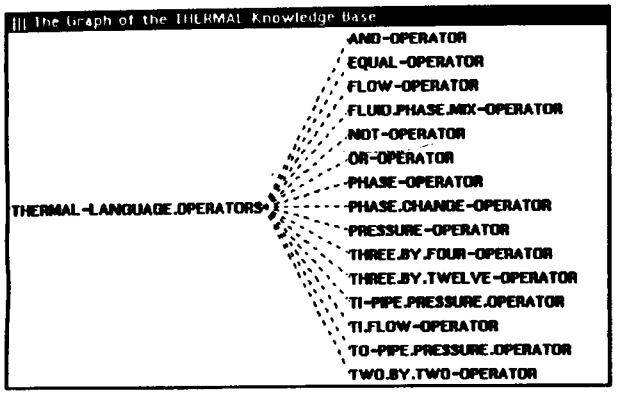
||| (Output) the DELAY-CONSTANTS Unit in THERMAL Knowledge Bas

Member slot: **CONSTANTS.LIST** from **DELAY-CONSTANTS**
Inheritance: **OVERRIDE.VALUES**
Comment: This is the list of all symbol constants defined by this unit.
Values: MSECS, SECS, MINS, HRS, DAYS

Member slot: **CONSTANTS.VS.LISP-VALUES** from **DELAY-CONSTANTS**
Inheritance: **OVERRIDE.VALUES**
Comment: This is an AList of symbol constants versus their value in the Lisp environment (if any).
Values: UNKNOWN

Member slot: **CONSTANTS.VS.VALUES** from **DELAY-CONSTANTS**
Inheritance: **OVERRIDE.VALUES**
Comment: This is the AList of all symbol constants versus their value defined by this unit.
Values: (MSECS . 1),
(SECS . 10),
(MINS . 100),
(HRS . 1000),
(DAYS . 10000)

Figure 3 - Thermal Language Data Structures



Phase Operator Table

Heat.Grad (component variable)

IN Fluid Phase (port variable)	neg.high	neg.med	no.change	pos.med	pos.high
sub.liq	sub.liq	sub.liq	sub.liq	sat.liq	sat.vap
sat.liq	sub.liq	sub.liq	sat.liq	sat.vap	sup.vap
sat.vap	sub.liq	sat.liq	sat.vap	sup.vap	sup.vap
sup.vap	sat.liq	sat.vap	sup.vap	sup.vap	sup.vap

Process Using Table

(Output) The FLUID.PHASE.TABLE.PROCESS Unit in THERMAL Knowl

Member slot: **DELAYS** from **PROCESSES-CLASS**
Inheritance: OVERRIDE.VALUES
ValueClass: (UNION NUMBER SYMBOL)
Values: UNKNOWN

Member slot: **EFFECTS** from **FLUID.PHASE.TABLE.PROCESS**
Inheritance: OVERRIDE.VALUES
ValueClass: LANGUAGE-CLASSES in PROCESS-LANGUAGE
Avunits: (CREATE-STATEMENTS in PROCESS-LANGUAGE ALL NIL)
Values: FLUID.PHASE.TABLE.EFFECT

Member slot: **EVALUATE!** from **PROCESSES-CLASS**
Inheritance: METHOD
ValueClass: METHOD
Comment: This method evaluates this process by calling !EXECUTE if !INVOKE.P is satisfied.
Values: PROCESSES-EVALUATE

Member slot: **INVOCATIONS** from **FLUID.PHASE.TABLE.PROCESS**
Inheritance: OVERRIDE.VALUES
ValueClass: STATEMENTS-CLASS in PROCESS-LANGUAGE
Avunits: (CREATE-STATEMENTS in PROCESS-LANGUAGE ALL NIL)
Values: FLUID.PHASE.TABLE.INVOCATION

Effect Statement

(Output) The FLUID.PHASE.TABLE.EFFECT Unit in THERMAL Knowledge Base

Own slot: **STATEMENT** from **FLUID.PHASE.TABLE.EFFECT**
Inheritance: OVERRIDE.VALUES
ValueClass: LIST
PutTranslation: TRANSLATION
Avunits: (LANGUAGE.TRANSLATE-STATEMENT in PROCESS-LANGUAGE ALL NIL)
Cardinality.Max: 1
Comment: This is an actual statement in the Process Language.
Values: ((OUT1 FLUID.PHASE) <- ((IN1 FLUID.PHASE) PHASE.OP HEAT.GRAD))

Own slot: **TRANSLATION** from **FLUID.PHASE.TABLE.EFFECT**
Inheritance: OVERRIDE.VALUES
Cardinality.Min: 1
Cardinality.Max: 1
Comment: This is the machine translation of the process code.
Values: (ASSIGNMENT-OPERATOR in PROCESS-LANGUAGE (OUT1 FLUID.PHASE) (PHASE-OPERATOR (IN1 FLUID.PHASE) HEAT.GRAD))

Figure 4 - Thermal Language Operators

A SITUATION-RESPONSE MODEL FOR INTELLIGENT PILOT AIDING

Robert Schudy
Kevin Corker

BBN Laboratories Incorporated
10 Moulton Street
Cambridge, Massachusetts 02238

ABSTRACT

An intelligent pilot aiding system needs models of the pilot information processing to provide the computational basis for successful cooperation between the pilot and the aiding system. By combining artificial intelligence concepts with the human information processing model of Rasmussen, we have developed an abstraction hierarchy of states of knowledge, processing functions, and shortcuts, which is useful for characterizing the information processing both of the pilot and of the aiding system. We are using this approach in the conceptual design of a real-time intelligent aiding system for flight crews of transport aircraft. One promising result from this work has been the tentative identification of a particular class of information processing shortcuts, from situation characterizations to appropriate responses, as the most important reliable pathway for dealing with complex time-critical situations. Situation-response models can be acquired from specialists, such as test pilots and systems engineers, and encoded in a situation-response pilot aiding system. The aiding system can then utilize that specialized expertise to assist flight crews dealing with novel situations, by characterizing the different aspects of the situation, and the appropriate pilot responses, in terms of a finite set of situation types and associated response procedures. There is promise that this approach to aiding will maintain the appropriate level of pilot situational awareness, while maintaining the peak cognitive workloads at levels more characteristic of situation recognition than of problem solving.

1. INTRODUCTION

The information available to the pilot of advanced commercial air-transport aircraft is becoming increasingly abstract from the physical parameters of the aircraft that are directly measured and monitored. This is true both for control such as flight controls, and for system monitoring and failure detection mechanisms such as engine monitoring and diagnosis. For example, automatic diagnostic systems have begun to reason about symptoms and situations of fault and failure rather than simply displaying monitored variable values. These trends, are changing the character of the interface between the pilot and the aircraft systems. We have concentrated on the structure of intelligent pilot aiding and pilot interface systems that:

1. Respond to situations such as diagnosis of engine failure;
2. Inform the pilot of these situations (at an adaptable level of detail); and
3. Advises the pilot of actions to be taken in response to the situation.

The architecture of the interface is designed to be quite general in the sense that it will support interactions between a broad range of expert-systems and pilots in a number of types of flight situations; our test cases and examples focus on the interaction and interface management of an engine-fault diagnosis system in commercial air-transports.

One promising result from this work has been the tentative identification of a particular class of shortcuts, from situation characterizations to appropriate responses, as the most important reliable pathway for dealing with complex time-critical situations. Situation-response models can be acquired from specialists, such as test pilots and systems engineers, and encoded in a situation-response pilot aiding system. The aiding system can then utilize that specialized expertise to assist flight crews dealing with novel situations, by characterizing the different aspects of the situation, and the appropriate pilot responses, in terms of a finite set of situation types and response procedures. There is promise that this approach to aiding will maintain the appropriate level of pilot situational awareness, while maintaining the peak cognitive workloads at levels more characteristic of situation recognition than of problem solving. The paper will describe the requirements for intelligent interface management (including the requirements for an explicit model of the pilot information processing functions), and then will outline the implementation architecture designed to meet those requirements.

The example problem being developed in this modeling effort is how the flight-crew and the automatic aiding systems together identify/classify and initiate appropriate response to an engine problem or failure during any portion of a commercial airline flight. Two interactive components of this problem are immediately apparent:

1. The engine diagnosis process (which is being researched at NASA-Langley Research Center); and
2. The selection, communication and execution of appropriate responses for the identified failure in the current context.

We will address the issues of the interface between diagnostic expert systems and the flight-crew, i.e., the selection, communication and initiation of situation information and appropriate responses. The use of the term "appropriate" conveys our concern for the evaluation of the full situation in which engine failure takes place as a necessary condition for response selection and advice. In addition to a full description of the "situation" of engine failure, the fact of cooperation between automatic expert system and human pilots necessitates careful consideration of the processes of human information processing and response selection to assure coordination in cooperation.

2. SITUATION-RESPONSE BEHAVIOR

While development of a model for the full repertoire of pilot information processing and flight control behaviors is a task that far exceeds our current state-of-knowledge and technology, we have developed a representation that we feel is appropriate for those behaviors associated with critical time-constrained situations. Our discussions with those responsible for pilot training, and our analysis of National Transportation Safety Board (NTSB) accident reports both lead us to identify a particular human information processing paradigm as predominant and highly preferred for airline pilots when dealing with time-constrained situations. We term this class of information processing *situation-response* behavior. The basic assumptions of the situation-response model are:

- That pilot situation-response information processing involves a situation assessment step in which the current situation is recognized in terms of a finite number of generic situation types; and
- That behavior in response to the situation is driven by procedures previously associated with those situation types.

Before elaborating the specific mechanisms for implementation of this model it is useful to consider the context from which it was derived.

The analysis of systems through description by multi-level abstraction hierarchies is a well established technique (Alexander, 1964, Asimow, 1962). Increasing levels of abstraction provide reduction of physical detail and an increase in functional or goal-oriented specification. It should be noted that the reduction of physical detail as one moves "up" in an abstraction hierarchy is matched by an increase in scope and system-oriented concern for context. More recently, Rasmussen (1983, 1984) has pioneered the description of humans in man/machine systems using the notion of abstraction hierarchies. Specifically, the functions associated with human perception through assessment and response selections and execution have been represented.

Movement through the "perceive/think/act" path (and various shortcuts and heuristics) are presented in Figure 1, which was derived by expressing the abstraction hierarchy in [Rasmussen, 1984] from an artificial intelligence perspective. Figure 1 is based on the description of processing in terms of an abstraction hierarchy of states of knowledge and processing functions which connect those states of knowledge. The states of knowledge are organized along a horizontal dimension which corresponds to the extent to which the concepts are expressed in terms of the system inputs or in terms of the system response, and along a vertical abstraction dimension. Thus organized, the useful states form a generally triangular shape with the sensors and effectors forming the lower two vertices and the full evaluated set of courses of action the apex. If the representations and processing steps in the sides of this triangle are correct and complete, the the processing sequence from inputs to outputs, following the sides of this triangle, is generally complete and correct. Unfortunately, this path is generally too computationally expensive to be performed in real time, either by natural or artificially intelligent systems. Within the boundaries of the triangle are numerous processing paths which shortcut the detailed processing, by connecting incomplete levels of analysis to partially defined responses. Example shortcuts at different levels of abstraction include reflexes, sensory-motor control, situation-response behavior, and satisficing [Simon, 1969]. The correctness of shortcuts depends on whether the response inferred on the processing shortcut is consistent with the responses which would have been inferred by the computations which are being shortcut. Additional information is provided in the companion paper *A conceptual framework for intelligent real time information processing*, in this volume.

In general, and in the situation-response model, the response for a particular situation is initiated at the lowest level of abstraction which has sufficient scope to select and execute the appropriate response. The situation attributes used to select any one response may span a range of abstraction. For example, the selection of the takeoff abort procedure depends on many higher level attributes such as engine diagnoses, but it also depends critically on the (primitive) air speed attribute. The kind and amount of human information processing required to accomplish a particular behavior is at least as great as that required to generate the highest level abstractions which select that behavior and the most difficult inferences in selecting and executing that behavior. Thus, it makes sense to talk about the level of abstraction of a behavior as a whole. Rasmussen (1983) identifies three general levels of behavioral abstraction:

- "Knowledge-based behavior" in which judgment and decision making and operator models of the system process, contribute to the identification and accomplishment of an operator's goals,
- "Rule-based behavior" by which the characteristics of situation are identified as belonging to a set of stored "situations" for which actions and responses are known, but for which procedures need to be tailored to the specific attributes of the situation, and
- "Skill-based behavior" in which limited packets, or sets of behavior are applied to specific stimuli in the environmental situation, with little or no reasoning effort applied to their generation or modification. The approximate ranges of abstraction of these three classes are indicated along the left margin of Figure 1.

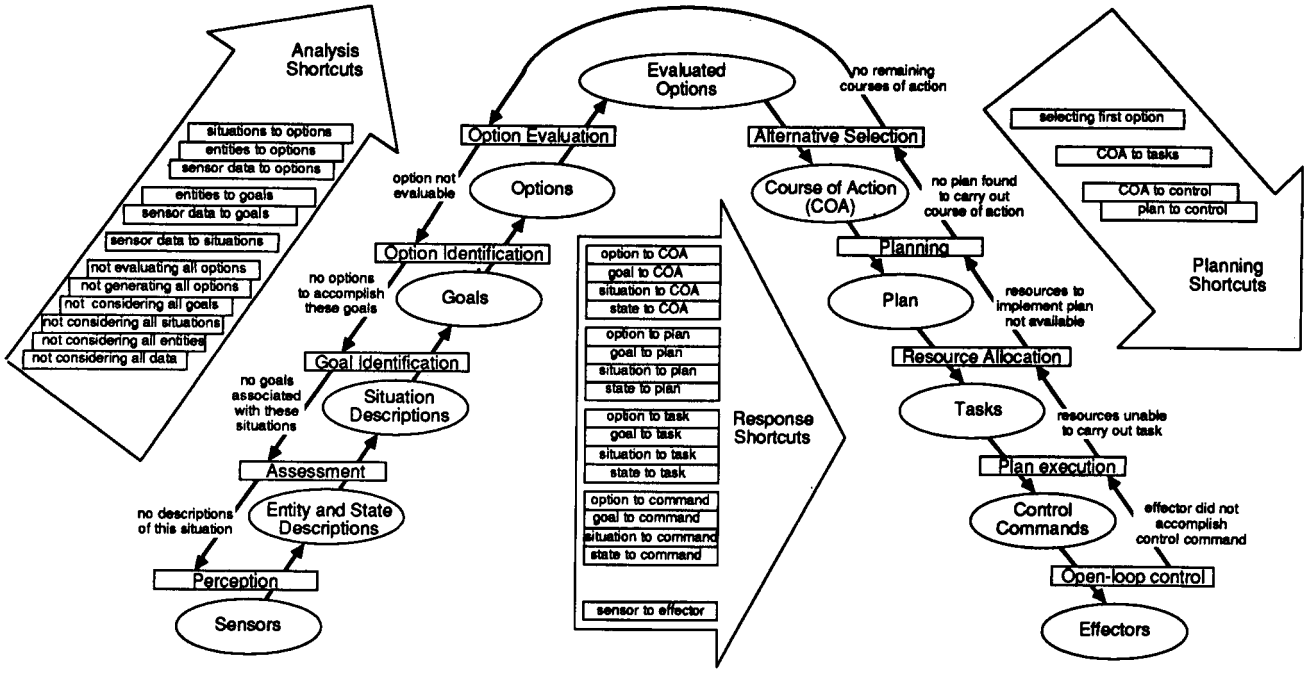


Figure 1. Real Time Processing Abstraction Hierarchy

The most efficient method for identifying an activity to execute is the skill-based strategy, which in the extreme, can be represented as involving no conscious decision-making activity at all, and might even be likened to an automatic reaction to a single stimulus. The correct and efficient enactment of skill-based behavior is expected to take place only after considerable training and/or experience, so that, in some sense, the cost for this efficiency can be thought of as having been borne at a previous time.

The association of behavior with situation attributes directly, without going through the situation assessment process, is a common and useful information processing shortcut. The establishment of such skill-based shortcuts reduces workload and reduces processing delay by uncoupling the situation assessment process from the process of adapting to changing situational parameters. In this model the activation and management of skill-based behavior (e.g., skill-based components of a response procedure) is one of the normal functions of rule-based behavior. The situation assessment function then assumes the role of enabling the execution of the skill-based behavior. Rule-based response selection is represented as taking a greater amount of time to complete, and therefore to tie up the cognitive resources of the pilot for a longer time. The information applied to these types of decisions, like that applied to skill-based decisions, takes the form of a production system. Rule-based decisions differ from skill-based decisions in terms of the number and level of abstraction of the situation attributes which select the behavior response. Rule-based decisions are considered to be more difficult because the enabling conditions are more difficult to compute.

Knowledge-based action selection requires a full analysis of the situation and an assessment of goals before particular courses of action can be selected and evaluated. Knowledge-based selection typically involves symbolic reasoning processes such as case analysis, projective evaluation, and search. Mental models play a large role in knowledge-based response selection.

Situation-response behavior is the class of rule-based behaviors in which there is a rapid assignment of a response schema to a set of stimuli that have been assembled (through training) into a trigger for the response. Situation-response behaviors are assembled and stored for rapid access and activation without requiring deep or novel reasoning. The links between situation characterization and response initiation are established by processes such as planning, rehearsal, evaluation, trial and error, training and practice. One advantage of situation-response behavior is the efficiency, in terms of time and cognitive resources expended, with which some correct response can be initiated. A second advantage is the ease with which correct situation-response behavioral models can be derived from experiments, experience, and engineering. The disadvantages of situation response behavior lie in the potential for inappropriate situation classification, and in the cost for development and storage of a sufficiently large set of situation types and associated response procedures to adequately deal with a complex and performance-critical task environment.

The focussing of our research on situation-response behavior is motivated by evidence that the need to resort to deep reasoning by aircrews in time-critical flight situations contributes to air transport accidents. Accident analyses suggest that in-flight abstract reasoning may shift attention from flight-critical tasks, and that deep reasoning under stress from potentially incomplete information and incomplete abstract models can produce results which are significantly and sometimes fatally inferior to those derivable from engineering studies, experience, and experiment.

The objective is for intelligent aiding systems to provide the flight crew with analyses of the situation and appropriate expert responses for the situation, to assist the pilot in correctly assessing and responding to the situation. For example, according to this model the behavior of a skilled transport pilot during takeoff may be determined almost entirely by his perception of the situation as a standard takeoff from that airport. For the pilot to implement behavior different than from established takeoff procedures depends on the pilot's recognizing that the situation is no longer solely, or best, described as a standard takeoff situation. The role of the intelligent aiding system is identifying the critical characterizations of the situation, and helping the pilot recognize these and implement the appropriate responses.

An example taken from the well-known United Airlines Flight 191 crash at O'Hare Airport in May, 1979 may serve to illustrate this concept. In that accident, an engine separated from the left wing of the DC-10 at approximately the time of aircraft rotation and lift off. In separating, the engine tore off the leading edge slats, which increased the minimum flight speeds necessary to prevent stall. The damage also rendered primary and secondary slat controls, slat disagreement, and stall warning systems inoperative. The flight crew reduced aircraft speed and climb angle, as per the standard company procedures for climbout with a failed engine. The loss of slat disagreement and stall warning indicators prevented the crew from realizing that by following the prescribed procedure they were inducing asymmetrical stall of the left wing, which resulted in a roll which was uncontrollable at the low flight speed.

The relevant situation types for this example are sketched in Figure 2.

Before engine separation the situation was described in terms of the normal takeoff situation types. At engine separation the engine-loss-climbout situation type also correctly described the situation. The engine-loss-climbout procedures require flight crew attention to airspeed, bearing, climb-rate, thrust-compensation, and crew behaviors designed to compensate for the engine loss and bring the aircraft to a safe altitude and flight path. However, the situation type of engine-loss-climbout did not fully describe the situation. Retraction of the left wing outboard slats placed the flight in a critical stall-regime situation. In low-speed flight any such flight control problem is an emergency of the highest priority, requiring immediate action. The procedures for such low speed

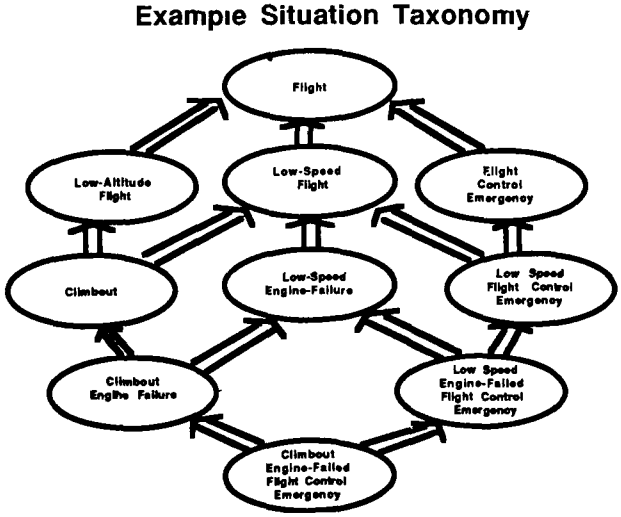


Figure 2. Situation Types in the DC-10 Crash

flight control emergencies are directed toward increasing air speed in order to increase control effectiveness, stall margin, and maneuverability. The appropriate response to this higher priority aspect of the situation would have been to sharply decrease climb angle to gain air speed. To quote the investigatory report:

> Each [of these causes: engine loss, slat retraction, warning loss] by itself would not have caused a qualified flight crew to lose control of its aircraft, but together during a critical portion of flight, they created a situation which afforded the flight crew an inadequate opportunity to recognize and prevent the ensuing stall of the aircraft. [NTSB Report #NTSB-A-AAR-79-17]

The basic problem was that in accepting the initial engine loss situational model, and following the established procedures for situations of that type, the flight crew did not recognize the other more critical aspects of the situation. The challenge for intelligent aiding systems is being helpful in such emergency situations, when the flight crew doesn't have any resources to spare. Ideally, the aiding system could have prevented the flight crew from accepting the situation as completely described by engine failure on climbout. It is reasonable to assume that a situation assessment system in the aircraft could have quickly detected the anomalous roll by monitoring the flight control and inertial systems. An aiding system could have given behavioral advice, such as maintaining at least V2 air speed, but such unmotivated paradoxical advice might confuse the flight crew. Thus advice such as "Roll Emergency" or "Flight Control Emergency", which implied both the situation and the response, would probably be better. Note that the effectiveness of such communication depends on the flight crew's having models of the emergency situation and associated response procedure, and on the effectiveness of the aiding system in stimulating the appropriate pilot situational awareness and response behavior.

3. SITUATION-RESPONSE AIDING

We now describe an approach to aiding the pilot in situation-response behaviors, including the functional requirements for such aiding and an approach to implementing a situation-response aiding system.

3.1 Aiding System Function

Figure 3 illustrates the parallelism between pilot situation-response information processing and an intelligent pilot aiding system which is helping the pilot with that processing. The flow at the top of the figure represents the processing steps which an aiding system might go through as it follows the situation-response pathway. The parallel flow at the bottom represents the pilot situation-response processing pathway. The four vertical arrows between these two horizontal flows represent the four main information flows between the pilot and aiding system. The figure shows how aiding systems could assist the pilot in assessing the situation, forming intentions, and executing those intentions. It also illustrates the flow of intentions from the pilot to the aiding system. The following paragraphs describe aiding system functional requirements to support the various phases in the situation-response information processing model.

Situation type. Situation types in the aiding system and pilot models should be different for distinguishable situations which require different types of responses. Thus, if two situations which are distinguishable by observable situation attributes have different responses, then those two situation types should be distinct. Conversely, if two situation types are never distinguishable based on their attributes, then the two types should be combined, and behavior associated with the combined type should be appropriate for a situation which could be of either type. Similarly, if two situations may or may not be indistinguishable, based on situation attributes, and those situations have different behaviors, then it is usually appropriate to generate an additional more general situation type which models the uncertainty by spanning the set of situation types which cannot be distinguished, based on situation attributes. The behavior associated with this more general situation type should be appropriate for the state of knowledge of the situation. When more specific situation attribute knowledge is available, then one of the more specific types should be used.

Pilot mental model. Effective high-level communication between intelligent aiding systems and the pilot requires compatibility between the conceptual model implemented by the intelligent aiding system at the pilot interface and the conceptual model held by the pilot. That is, the system image must be compatible with the pilot mental models. Because effective communication about complex topics depends on coherent relationships between topics at multiple levels of abstraction, severe requirements are imposed on the compatibilities between the images presented by intelligent pilot aiding systems and the pilot's own mental models. Indeed, it may be necessary, or at least desirable, to base the system image on an explicit representation of prototype pilot mental models.

Situation Attributes. The human interface should provide sufficient information on the situation attributes, at the appropriate level of detail, and in the appropriate structure, so that the pilot can correctly classify the situation in terms of his mental situation models. Different values of situation attributes which are significant in terms of situation assessment and response should be clearly distinguishable to the pilot who is to base his situation model and response on those attributes. The systems should provide the situation attributes at a level of detail which matches the human information processing input requirements. For situation-response processing this level of detail is the level necessary to unambiguously identify the situation type and to refine the attributes of that type at the level of detail needed to support the response. It is clearly critically important that situation descriptions be refined to the level of detail required to support the correct pilot response. It can also be very important that superfluous detail not overload and distract the pilot. The system should therefore provide different levels of detail and different foci of detail to support changing human information requirements.

Assessment aiding. The human computational burden of situation assessment and situation classification can be reduced by automating some of the assessment functions. The processing to support this assessment involves the fusion of different situation attributes into more abstract attributes which support a simpler form of the situation assessment processing. For example, a pilot may be presented with N1, N2, temperature, pressure and other engine information. He may also be presented with airspeed, altitude, throttle and other information. What the pilot needs to know is how the engine is performing now and how it will perform in the rest of the flight. An engine diagnosis system should combine this wealth of data from the sensor systems, to produce a model of the engine status which can be more easily matched to situation types which serve as the basis for pilot action. Intelligent pilot interface and aiding systems should also support the pilot in giving priority to the assessment of critical aspects of the situation.

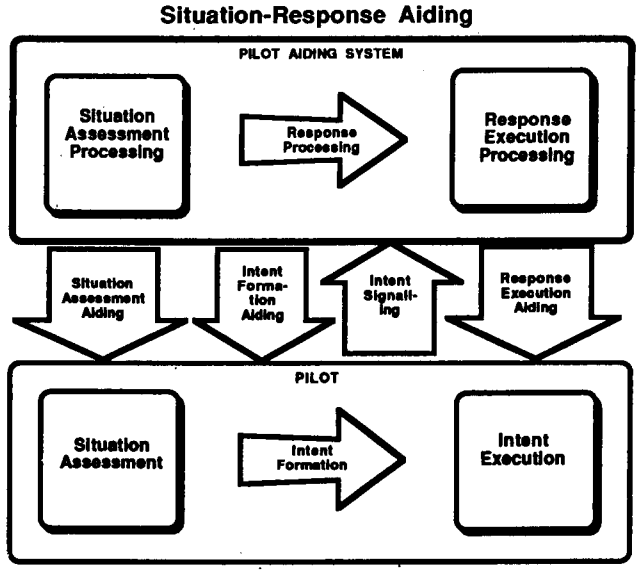


Figure 3. Situation-Response Aiding

Response Resolution. When different aspects of the situation require different procedures, then aiding systems can support the pilot's resolution of this behavioral dissonance, by constructing or identifying appropriate response procedures. These procedures can generally be derived from procedures for the different aspects of the situation. For example, an aiding system may be able to determine that a particular reduced throttle schedule on a damaged engine would provide the most favorable compromise between the conflicting behaviors of obtaining maximum thrust and protecting against total engine failure.

Knowledge-Based Reasoning. Experience with air crashes and other accidents indicates that pilot deep reasoning in dangerous situations with real-time response requirements has some undesirable properties. For example, successful knowledge-based reasoning normally requires case analysis, comparison with examples, mental simulation, and other cognitive activities which consume most or all of the human cognitive capabilities for significant time intervals. Further, knowledge-based reasoning under stress in real-time situations often leads to erroneous shortcuts in analysis. Thus automation of the knowledge-based reasoning to support situation-response behavior of pilots is highly desirable. It is also highly desirable that the aiding system not impose the requirement for knowledge-based reasoning, either through not fully supporting operator situation-response behavior, or through requiring knowledge-based behavior to use the aiding system.

Feedback. Humans or automatic controllers often require some information on the state of whatever it is they are effecting in order to monitor the progress of the response, and to provide the feedback necessary to implement any control laws in the response. The nature of the feedback can be determined using models of human performance and through experiment. Considerable benefit can be gained by adapting the feedback to the situation, response, and control laws. Our pilot information processing model suggests, through its hierarchical structure, that both control and feedback be available at multiple levels of the abstraction hierarchy. The particular form of the feedback or control should be geared to the level at which the pilot is interacting with the system.

Goal Monitoring. Responses to situations, either by automated systems or by the pilot, usually have as their focus attainment of some goal which terminates the tasks (or subtasks) required for that goal's achievement. If an intelligent aiding system is aware of tasks focused on a particular goal, and can determine when the goal state has been reached, then it may be of considerable value to the flight crew to announce the attainment of those goal states. Alternatively, a task may be one in which the goal state is to be maintained until some expected event occurs. Intelligent aiding systems can provide valuable assistance by inferring the values and ranges of those goal states, signalling their initial attainments, monitoring for their maintenance, and responding appropriately when the termination event occurs.

Context Change Monitoring. A task may become inappropriate because the context in which the task was being performed has changed sufficiently to make the task impossible to perform or to make the goals no longer of interest. The pilot information processing model provides two general pathways through which this can be discovered. One pathway follows from situation assessment. The second pathway follows from the monitoring of the ongoing task execution, when task execution requirements are no longer met, task performance expectations are not met, or other unexpected conditions are discovered. Intelligent aiding systems can assist in these cases by assessing the situation and monitoring for conflicts, and by monitoring task execution and appropriately alerting the flight crew of tasks which must be modified, abandoned or replaced.

3.2 Aiding System Implementation Approach

A computer implementation of the situation-response information processing model, using machine intelligence techniques, is illustrated in Figure 4.

The upper three boxes hold examples of the *a priori* knowledge structures for situation attributes, situation types, and response procedures. The lower three boxes hold examples of the runtime instances of those situation attributes, situations, and procedures. Arcs in the figure illustrate the explicit relations between the representations, and the large arrows illustrate the runtime processing steps. In our baseline implementation approach all six of these knowledge structures are represented by frames. The following paragraphs describe the representations and processing steps in Figure 4.

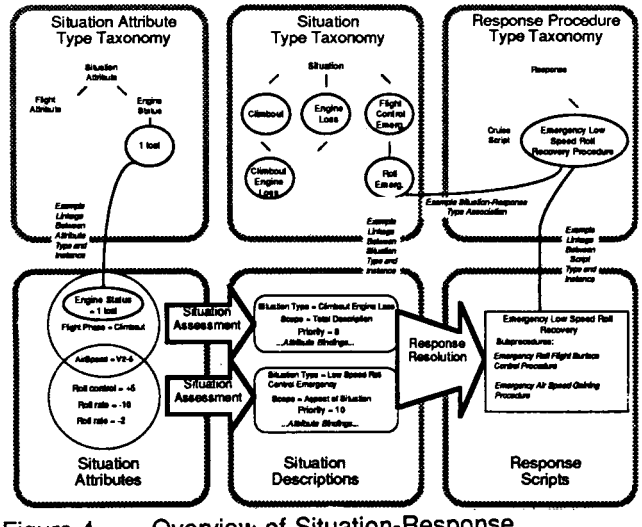


Figure 4. Overview of Situation-Response Implementation

Situation Types. Situation types are attempts to represent the pilot's mental models of generic classes of situations. For example, preflight, cruise, and landing roll are different types of situations in a typical flight. A situation type is represented in our computer model as a frame in a frame-based knowledge representation system. (In a frame-based representation, the frames consist of sets of ordered pairs of slot descriptions and slot fillers. The slot descriptions specify the relationship of the slot filler to whatever is being described by the frame; the slot descriptions are therefore often termed "relations" or "roles.") In situation type frames the slot is filled with a description of the class of things which could fill this slot in an instance of this type of situation. Situation types are sets of descriptions of relevant attributes of the generic situation and the generic relationships between those attributes. In the situation-response model each situation type may have associated with it a description of the behavior to be performed in that situation. For example, an engine-failed-climbout situation type might be described by the frame:

```
engine-failed-climbout
     SuperC          climbout-situation
     SuperC          single-engine-failure-situation
     response        engine-failed-climbout-procedure
```

The SuperC relations indicate that this situation is a specialization of both the climbout situation and the situation in which a single engine has failed. If the engine-failed-climbout situation were described as a specialization of climbout-situation and of single-engine-failure-situation all of the slots of those situations (and the SuperCs of those situations) would be *inherited* by the engine-failed-climbout situation.

Situation Response Procedures. Associated with each situation type is one description of the procedure to be performed in situations of that type. These procedures use the actual values of situation attributes in much the way that a computer software procedure uses the formal parameters. For example, the final approach procedure may key specific actions to specific values of the altitude attribute of the final approach situation.

Situation Assessment. Situation assessment can be modelled computationally by a matching or classification process in which the perceived situational attributes form a pattern, and the goal is finding all the situation types which can fit that pattern. Note that situation assessment does not attempt to resolve the ambiguities and inconsistencies due to the lack of information. Indeed, it cannot do so reliably. Rather, situation assessment provides a description of the *possible* interpretations of the current situation, together with the assumptions underlying those interpretations.

Perceived Situation Attributes. Perceived situation attributes are the attributes of the situation which are computed in real time from sensor data and models of the things perceived. This processing can be hierarchically structured, as sensor information is combined into abstractions with successively larger scope. For example, perceptual processing may include diagnosis of an incipient engine failure.

Situation Descriptions. The situation assessment process produces a situation description from an appropriate situation type, by replacing the attribute descriptions of the situation type with their refined values, to produce an instance of the situation type tailored to the actual current situation. For example, a description of a single engine failure situation might be derived from the single-engine-failure situation type, in part, by replacing the failed-engine ID attribute of the type by the ID (e.g., left, right) of the failed engine.

Ideally all situations could be described in terms of situation types which accounted in detail for all of the attributes of the situation. Such a well-fitting situation type would result in an equally appropriate situation description, and a very well focused response procedure. In some domains the number of different kinds of situations, and the number of combinations of different situation attributes, may be sufficiently small to permit the tailoring of situation types to each of the combinations of situation attributes. However, in complex domains, the number of different combinations of situation attributes precludes unique association of a situation type with each possible combination of attributes. There are at least four ways to obtain reasonable situation descriptions without having an unmanageable number of situation types: by refining attributes using perceived values, by using more general situation types, by describing the situation in terms of its different aspects, and by describing situations with subsumption hierarchies of descriptions of different levels of detail.

4. Conclusions

When the information processing pathways of a pilot or intelligent aiding system is laid out in an abstraction hierarchy stretching from inputs to actions, a particular subset of those pathways is found to describe the most important and desirable for pilots pilots others engaged in critical time-constrained system operation tasks. A model of these situation-response behaviors forms a sound basis for pilot training and for systems which aid the pilot in correctly assessing and responding to situations. Major research tasks remaining include verifying the scope of the model relative to the full range of pilot aiding requirements, implementing a situation-response aiding system, and testing with pilots in realistic real-time situations.

5. REFERENCES

Alexander, C., Notes on the Synthesis of Form. Harvard Press, Cambridge, MA, 1964.

Asimow, M., Introduction to Design Prentice-Hall, Englewood Cliffs, NJ, 1962.

Rasmussen, J., Skills, Rules, Knowledge; Signals, Signs and Symbols; and other Distinctions in Human Performance Models. IEEE Transactions of Systems, Man, and Cybernetics SMC-13 (3), 1983.

Rasmussen, J., Strategies for State Identification and Diagnosis in Supervisory Control Tasks, and Design for Computer-based Support Systems. In (Rouse) Advances in Man-Machine Systems Research, Vol. 1 pp. 139-193, 1984.

Simon, H., The Sciences of the Artificial. M.I.T Press, 1969.

A HUMAN PERFORMANCE MODELLING APPROACH TO INTELLIGENT DECISION SUPPORT SYSTEMS

MICHAEL S. MCCOY
DESIGN ENGINEERING - HUMAN FACTORS
MCDONNELL AIRCRAFT COMPANY
ST. LOUIS, MO 63166

RANDY M. BOYS
ARTIFICIAL INTELLIGENCE LABORATORY
DEFENSE SYSTEMS AND ELECTRONICS GROUP
TEXAS INSTRUMENTS, INCORPORATED
DALLAS, TEXAS 75266

ABSTRACT

Manned space operations require that the many automated subsystems of a space platform be controllable by a limited number of personnel. To minimize the interaction required of these operators, artificial intelligence techniques may be applied to embed a human performance model within the automated, or semi-automated, systems, thereby allowing the derivation of operator intent. A similar application has previously been proposed in the domain of fighter piloting, where the demand for pilot intent derivation is primarily a function of limited time and high workload rather than limited operators. The derivation and propogation of pilot intent is presented as it might be applied to the Darpa/AFWAL "Pilots Associate" or AAMRL "Super Cockpit" programs.

INTRODUCTION

Traditionally, a Human Performance Model is used as a design tool to predict various man-machine system designs. Such a model could be incorporated into an intelligent computer system for two additional applications. First, the model could be one element of a decision and task support system. It could predict the operator's forthcoming tasks, anticipate upcoming decisions, and formulate any necessary decision or execution aids. Second, the model could serve as a performance monitoring tool by analyzing the differences (and causes of differences) between expected and actual operator actions. These differences may indicate that: (1) modeled operator activities and goals were incorrect and require updating; (2) steps in a procedure were omitted by the operator; or (3) that critical information has not been presented to the operator. This data will be factored into the decison/execution support system, thus enabling the computer system to recommend or perform corrective actions.

The Pilot's Associate (PA) system is a decision and task support system which will use artificial intelligence (AI) technology to aid post-1995 fighter aircraft pilots. One of the expert systems within PA is the Pilot/Vehicle Interface (PVI) system. This system, employing machine intelligence and enhanced control and display devices, is being designed to improve the pilot's situation awareness. The primary goal of the PVI expert is to manage information flow between the pilot and the other PA expert systems (Systems Status, Mission Planning, Situation Assessment, Tactical Planning, and the System Executive) (McCoy & Boys, 1987).

Another example of the developing need for an "Electronic Crewmember" is that proposed for the Super Cockpit (SC) program. The SC system will immerse fighter pilots in a visual, aural, and tactile world that is a combination of real-world and computer-generated events. Compared with the PA program, SC is more of a technology pull for these "virtual world" subsystems than expert system technology itself. The PA system would be a subsystem available to the pilot through the SC interface.

Critical to the performance of both programs is the ability to derive and reason from Pilot Intent (PI). This capacity is provided by the Pilot Intent subsystem of PA and the Pilot Intent Inference Engine of Super Cockpit (McCoy & Boys, 1987 and Martin 1986). PI helps shift the pilot's role from operator to system manager so that the pilot can specify that something should be done, without precisely specifying how it should be done. Thus, PI must be able to identify the activities in which the pilot is involved, their relative priorities and execution constraints, and the intent of any non-deterministic pilot commands. This knowledge is then merged with any new tasks identified by the various Electronic Crewmember's (EC) subsystems, resulting in EC's tailoring of displays and man-machine interactions.

To identify pilot intent, the PI subsystem must assume that the pilot exhibits purposive behavior. By observing the behavior of the pilot, the system can derive the purpose or intent of the pilot while performing a particular task. The pilot's intent is derived from three sources: (1) discrete commands; (2) complex commands (requiring EC decomposition); and (3) inferred intent (derived from observing pilot behavior as a function of the current context). It will be shown that this derivation of purpose from behavior is a reverse application of the Human Performance Model (HPM) method of predicting behavior based on a known purpose.

*Portions of this document have been previously published (McCoy & Boys, 1987)

Traditionally, HPM's are used to predict consequences of specific situation-response mechanisms. These models typically predict operator behavior, performance, and workload within a given situation. By modeling the operator's performance of known tasks, the consequences of assigning new tasks and increasing operator duties may be predicted. These models can also predict the effect of relieving the pilot of specific tasks through automation. The projective nature of EC mandates these applications of HPM's, but this discussion extends their utility to the real-time EC system itself.

The Electronic Crewmember EC system must provide information in a timely manner and maintain the pilot's situation awareness. Since the situation is dynamic the system must be capable of knowing the goals of the pilot and of reasoning about the decisions it must make, as well as actions it must perform, to meet those goals. One method of providing the system with this ability involves incorporating a model of the human within the EC system. This model would allow for predictions of the pilot's decisions, actions, and information needs, thus providing the system with a means of not only anticipating pilot requirements, but also using these predictions to supplement the pilot and improve overall system performance.

The efforts put into developing the traditional HPM can, therefore, apply to the embedded HPM as well, particularly if the context for this PI performance model is supplemented by the other Electronic Crewmember experts. The correspondence of these two models is presented in the pilot intent interpretation section. Once the relationship between the HPM and PI has been established, the analysis procedure, applicable to the development of both, will be presented. This analysis will be shown to provide all of the information necessary to develop the purpose-behavior relationship required for both models. Finally, applications of the combined human performance/pilot intent model are described.

HUMAN PERFORMANCE MODELS

Although the title "Human Performance Model" implies a concentration on the human's role in a man-machine system, the only way to truly model the human's performance in a system is to include a model of the machine and of the environment that imposes demands on the human. Therefore, a closed-loop system model is developed, with statistics being collected (or predicted) on the human's participation in the total system performance. Several purposes for HPM's are: (1) explanation of the system being studied; (2) analysis of the system being studied; (3) assessment of the design of a new system; and (4) prediction of performance, or operator workload, within an existing system.

As an explanatory device, HPM's can be used to: define a system or problems within the system; isolate ambiguous relationships between inputs and outputs to the system; and enhance the analyst's understanding of the underlying dynamics of the system. The resulting networks often capture the functional intent of the system in a dynamic context, and can serve as input to training or development aids. These 'explanations' almost invariably exist at more than one level of abstraction, hence they are applicable to a number of different purposes (users, trainers, funding sources, analysts, etc.).

A second purpose for HPM's is as an analysis vehicle. When developing the model, the analyst is required to investigate and determine critical elements, components, and issues within the system being studied. In addition, the model can be used to investigate hypothetical relationships between various components of the system and new components to be added to the system. By using this method, much insight can be obtained about system characteristics without physically interfering with an operational system.

The HPM can also be used during design to assess, and aid in planning, new systems. In this way, required or suggested changes to the system design can be identified early in the design process, thus minimizing the risk of delays to the project during development. The advantages to life-cycle costs provided by 'up-front' work have been well documented.

A final application of HPM's is to predict the effects of proposed solutions to the problems being studied on the existing system (be it physically or hypothetically existent). By creating a model reflecting the proposed solution and then comparing the results to the simulation of the system containing the problem, an analysis can be made of the relative improvement on performance and workload based on this solution.

There are several critical components of the HPM which interact to reflect total system performance. They consist of: (1) system demands; (2) cognitive situation assessment; (3) decision making or task selection; and (4) task/procedure execution. Each of these components of the model will be explained in detail.

System Demands

Demands on the system are generated from several sources: (1) system dynamics; system malfunctions; (2) environmental factors; (3) situation contingencies; and (4) mission status. Each of these sources must be modeled separately. The first, system dynamics, represents a model of the machine, in this case the vehicle. This model represents the dynamics of the system, including the spatial position of the aircraft, the state of its various weapons, etc. When the aircraft takes off, one demand on the system is to retract the landing gear. Another demand is to raise flaps. These are examples of demands imposed on the operator by the system dynamics model.

Another model to be developed for the system is the system malfunction model. This model simulates any malfunctions, such as oil pressure problems, that can occur during flight and are important enough to be included in the system. In this way, the potential malfunctions to be considered by the operator can be represented and accounted for in generating demands on the operator.

The environment imposes specific demands that must be accounted for in the system. These demands can be weather related, such as wind gusts influencing the aircraft's state, or threat related, from either air or surface threats. In any event, they must be modeled as potential demands imposed on the system. Finally, the mission status can impose demands.

Any combination of the above demands may impose delays in the mission, requiring extra activities to be performed by the pilot to compensate for these delays. These activities can include minor adjustments to airspeed or major revisions of the mission route. All of these demands must be maintained as inputs to the pilot's cognitive situation assessment.

Cognitive Situation Assessment

When a new demand is generated, or an existing demand is eliminated, the situation must be reassessed. This assessment consists of two major tasks: 1) reprioritizing existing demands on the system and 2) developing a plan to eliminate one or more of the demands imposed on the system. Therefore a queue of demands exists (short-term memory) which must be ranked in order of importance. The prioritization can be represented as an algorithm in which a relative significance is assigned to each demand based on the situation (mission segment, current threats, relative altitude, etc.). Another method of prioritizing demands is through a set of production rules. This method would allow for determining the demand priority from the situation (as above), but would also allow for inferring the situation from the demand priority. Because of this dual applicability, the production rule approach may be more suitable for the PI portion of this model.

When the demands have been prioritized into their relative importance, an elimination of demands, or focusing on high priority demands for consideration, must be performed. This could be modeled using a heuristic process of elimination. When a seemingly managable subset of all demands has been developed, plans are then developed for satisfying demands. This stage of plan development examines the high priority demands and selects candidate tasks/procedures which will contribute to satisfying one or more demands. It is likely that the same task/procedure could be selected to meet several pending demands. Likewise, one demand may likely be satisfied by several tasks/procedures. In any event, all candidate task/procedures must be examined. This process is largely one of pattern matching (demands are matched to a plan library) and does not necessarily require production rules in either the HPM or PI subfunction functions which perform this task.

Decision Making (Activity Selection)

As mentioned, any particular demand could be met by alternative tasks/procedures. Alternatively, a specific task/procedure could meet many of the demands. Therefore, an assessment of the candidate tasks/procedures must be made. A list of candidate task/procedures that lend themselves to one or more of the demands must be generated. This list will provide the basis for selecting the next activity to be performed by the operator.

A ranking of tasks/procedures must then be performed. This ranking must take into consideration the number of demands that will be met by each task/procedure, the time required to perform the task/procedure, the resources required for performance of the task/procedure, and any other metrics of task desirability. This ranking can be performed by development of an expected net gain (Baron et. al., 1980) or production rules could be applied. Unlike the assessment stage, all alternatives will, most likely, not be investigated.

The final step in the decision making routine is to select the task/procedure that "best" meets demands or improves the situation. This selection criterion could be algorithmic, such as maximizing the expected net gain, or it could be heuristic. If it is heuristic, then a set of production rules must be determined. In addition, production rules may include task ranking. After the task/procedure has been selected, the effect on the situation, including the passage of time, must be simulated.

Task/Procedure Execution

HPM's traditionally incorporate a simulation of system activities which accounts for resource utilization as well as for the advance of time. Often, this simulation can be represented in the form of a network of activities, each activity seizing resources and consuming time. In this way, the potential bottlenecks and time delays, due to resource limitations (which includes the operator's attention, perception, cognitive and motor availability) can be incorporated into the model. In addition to accounting for time delays and resource utilization, the activities will satisfy the designated demand. When the task/procedure has been completed, the cognitive situation assessment portion of the model must be invoked.

Summary

HPM's can be created using many different conventions, but if the model is to reflect the operator's cognitive processing, the style chosen must support these generic stages. The model must also include any variability (usually stated as the appropriate statistical distribution, applied Monte-Carlo) associated with not only the situational demands, but with the human operator himself. Once a model is completed, it can be used for determining inherent faults in the system as well as optimizing system execution along any of the simulated parameters. These parameters include proposed system enhancements (improvements or new capabilities) or factors associated with operator performance (load leveling, response time, precision, etc.).

PILOT INTENT INTERPRETATION

The recognition of pilot plans and goals has not been a problem in the development of contemporary fighter aircraft. The role of the man-machine interface was simple and direct: receive and respond to pilot commands (switch or stick

inputs) and occasionally provide simple alerting, landing, targeting, or other cues. These cues and other rudimentary automated systems comprise the entirety of contemporary avionics autonomy.

The notion of an "Electronic Crewmember", a computer system which interacts with the pilot in a highly dynamic fashion through the incorporation of artificial intelligence and advanced input/output technologies, will change the nature of the man-machine interface. This change has been mandated by an explosion of information and activity in the cockpit as well as by the changing nature of ownship and threat capabilities. There is more to do and less time in which to do it. There is also little time for the pilot to train a computer system into an acceptable cognizance of his needs to the point where it can begin to assist him in his tasks. The EC must maintain the information flow bandwidth required for this task, which must be looked upon as reciprocal.

The reciprocal nature of communication acts has long been recognized, but only recently has the advance of computer system technology allowed for intelligence on both sides of the interface. The dynamic and independent nature of both the human and computer conversants requires that each party maintain a model of the other's behavior, as well as a model of the other's model of ones self! This is an area where psychology and sociology have done much research, but only recently have the designers of computer systems attempted to apply these theories to a man-machine interface (Baron, et. al., 1980; Wellens & McNeese, 1987; Martin, 1986).

An analogy to syntax and semantics as applied to linguistics can be made: only certain actions can follow other actions (syntax) as opposed to knowing what actions make sense right now (semantics). An example, as applied to the piloting domain, is knowledge that the pilot is requesting a EC subfunction because he has selected the 'Systems' menu on the display (syntax) as opposed to knowledge that he is requesting the detailed version of the engine status display because there have been indications of engine problems of which the pilot would have been aware (semantic). The identification of intent, regardless of the detail at which it is determined, will allow the PVI and other system components not only to model the pilot's upcoming activity in more detail, but also to provide pilot assistance in increasingly appropriate ways.

Of concern here are those EC functions which involve: the derivation of pilot goals; the identification of active pilot tasks and procedures; and the transmission of these goals and activities to the entirety of the EC system. These tasks are done both explicitly, using pilot commands (or 'expanded' commands), or implicitly, inferring pilot concerns and plans from the pilot's activity or inactivity in light of the current or anticipated situation as reported by the various EC subfunctions. The method for this derivation of 'unstated' intent is related to the already forwarded portrayal of HPM's.

The HPM methodology portrayed a model of the mechanism that a human employs to: (1) examine a situation; (2) infer the demands generated from the situation; (3) develop plans for meeting demands and improving the situation; (4) selecting a desired plan; and (5) executing the plan, which then feeds back to effect the situation. As presented in Figure 1, PI subfunction essentially performs the reverse process. Once a specific behavior is observed, this model must infer the demands and situation that must have caused this specific behavior. Therefore, an examination of this reverse process must be made.

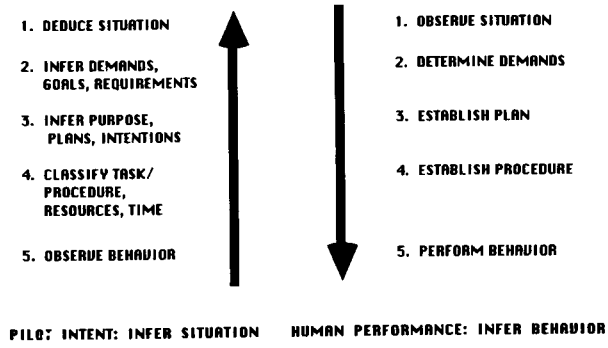


Figure 1. Operational Relationship of Pilot Intent and Human Performance Models

Behavior Observation

As the pilot operates the aircraft, he exhibits behavior that can be observed through the Hands On Throttle and Stick (HOTAS) system, for control of airframe and specific switches; the touch panel, for display change input, target designation, etc.; miscellaneous switches, for control of subsystems; Helmet Mounted Sight (HMS), for target designation, etc.; and the Voice Interactive System (VIS), for verbal inputs to the system. In addition to these command activities, other behavioral and physiological indices will be available to EC from the assumed pilot state monitoring system (used to monitor G-LOC, workload, etc. derivation). These indices includ eye movements, direction of gaze, etc. When behavior has been observed it can be assumed to be purposive unless erroneous. While there is a one-to-many mapping problem at this level of analysis, the following stages successively refine the estimate of the purpose of any observed behavior.

Task/Procedure Definition

Once the input has been detected, several questions must be answered. First, does the observed behavior contribute to an existing hypothesized) procedure with some reasonable confidence? If so, this helps confirm the belief that that procedure is being executed. This belief, in itself, may well be expressed in terms

of a probability. If the observed behavior does not confirm the belief, then there are two potential explanations. First, the believed procedure (intent) is false or second, a new procedure has been initiated in addition to those currently active.

If the task being initiated does not correspond to current expectations, yet does not conflict with them, then the task may be either an isolated task or the initiation of a new procedure to be performed in parallel with the current procedure. Knowledge must be gathered to distinguish which situation is occuring.

Purpose Inference

Given that a new task or procedure is inferred the purpose of the task or procedure must be determined. As with the HPM, a number of purposes could be met with many tasks, therefore a mechanism must be devised to distinguish when a task/procedure applies to one or another purposive behavior. The type of distinguishing characteristics necessary to determine the intention of the pilot would be current mission segment, other tasks/procedures being performed or recently completed, current threats, etc. All of this information must be maintained by the system to help infer the purpose of the behavior and may lend itself to a set of production rules. PI subsystems would be assisted by the expertise of the various other Electronic Crewmember systems in making this inference.

Demand Inference

Once intent or purpose has been inferred, the demands or goals being addressed (mapped) to the purpose can be determined. The identification of these goals is necessary if the system is to anticipate the information requirements of the pilot during execution of the procedure. In addition, given the purpose or goal for execution of a procedure, steps of the procedure are identified. While a step may be inadvertantly skipped by the operator, if the step is detected by the pilot intent model, these steps can be performed for the pilot or brought to the pilot's attention depending on the level of automation. As a final case, EC may have been authorized to respond autonomously to the identified demand.

If the demands and goals have been determined, they can be used to infer the situation confronting the pilot. This inference can be used as a check in which the existing goals and demands can be compared with inferred ones, verifying current belief of the pilot's intentions. If this belief is confirmed, then the system proceeds to operate as a decision support system. If this belief is not supported, an investigation must establish a proper belief in pilot intent.

PILOT/VEHICLE INTERFACE ANALYSIS AND DESIGN APPROACH

The general methodology being followed by the MCAIR/TI team in the development of the PVI is presented in Figure 2 (McCoy, Boys, 1987). The "System Analysis" effort has only recently been completed, with current efforts being directed to various "System Design" activities. The PVI analysis process consists of: Function Decomposition and Task Analysis. The PVI design process consists of: Task Analysis; Information Requirements Analysis; Static Display Definition; and the Pilot Intent/Human Performance Model sections (i.e., Human Performance Analysis, Dynamic Time Line Analysis, Pilot Intent Model Development, Timing Requirement Analysis, and Development of Automation Requirements). The final step in the process is to use the HPM in evaluating such procedures as: dynamic allocation of tasks; levels of autonomy (pilot/system operational relationship); load leveling of information; and those more traditional applications discussed earlier. The following paragraphs contain descriptions of each activity. It is important to remember that this analysis is iterative. Subsequent progress will be documented as necessary.

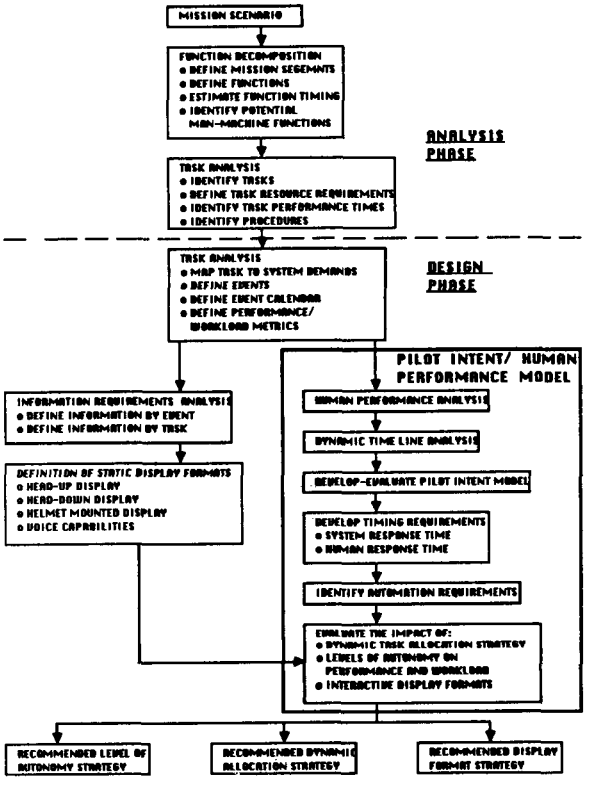


Figure 2. PVI Analysis and Design Approach

Analysis Process

Function Decomposition - The top-level mission functions in which the EC is involved (which include no less than all piloting activities) have been identified. These functions have been decomposed into subfunctions necessary to meet mission goals of survivability and effectiveness, estimated in terms of function timing requirements.

Tasks are initially classified as belonging to one of five categories: 1) tasks that can be performed only by the pilot, 2) tasks that can be performed by the pilot or a computer (the EC or conventional) but which the pilot does better, 3) tasks that can be performed by either the pilot or the computer, 4) tasks which either can perform but which the computer does better, and 5) tasks which only the computer can perform. It is sometimes necessary to make these assignments at the task level.

With the continued increase in computer capabilities (especially in the AI technology area), these latter categories continue to grow. Therefore, assumptions must be made about the level of potential automation available in the future aircraft. Completion of function decomposition leads directly to task analysis.

Task Analysis - In task analysis, individual tasks are defined to perform the functions. These tasks are performed throughout the mission and can be separated into three groups: perceptual, cognitive, and motor. The current task list is based on an analysis of a traditional high performance fighter aircraft (F-15, F/A-18) and does not reflect the task analysis or task allocation projected for an advanced combat aircraft equipped with the EC system. Many of the motor portions of tasks could be performed by various entities of the EC system. This leads to the next task required in task analysis, definition of resource requirements.

Once tasks have been defined, the resources necessary to perform the task must be determined. The term "resources" refers to a human, a machine, or a combination of the two. The definition of resource requirements is a logical extension of the function classification process. As resources are assigned to a task, the timing of that task can be determined. The timing of the task may be different given different resources. Analyses can be performed to develop the tradeoffs associated with automating tasks, varying degrees of pilot/system interaction, task allocations, etc.

Design Process

Task Analysis - The next set of activities occur simultaneously. First, a definition of events in representative missions must be made. An example of an event is detection of a threat. Given this event, what tasks are necessary to survive the threat? By answering this question, a link is established between demand for tasks and the tasks necessary to meet that demand. In addition, alternative tasks can be determined, leading to an analysis of what conditions lead to which tasks.

As tasks are defined, situations will arise in which specific tasks occur sequentially. This sequential occurence of tasks is called a procedure. Procedures must be defined to include the conditions under which they will occur. Defining procedures allows for the summarization of tasks under specific conditions. These specific conditions will aid in determining the procedure being performed when the behavior is observed. Of course, variability of procedures (particularly when the EC system attempts to 'internalize' these systems for dynamic use) must also be represented. The variability of procedures is represented by alternative task orderings within a procedure.

The final major activity to be addressed in the task analysis is to determine the performance and workload metrics to be used in analyzing various operating practices. There are two aspects to performance and workload metrics. First, metrics are needed to predict performance under different operating conditions. Second, metrics are used to measure performance and workload during experiments and during flight. These metrics should be developed for each task or procedure. Once this activity is complete, the task analysis is complete for both analysis and design and parallel activities related to the Information Requirements Analysis and the development of the Pilot Intent/Human Performance Models can begin.

Information Requirements Analysis - When the task analysis has progressed sufficiently, an information requirements analysis must be initiated. Preliminary versions of this analysis were conducted for the analysis stage task analysis, but they must now be mapped to the emerging EC task analysis. In addition, this information analysis should be performed for each event. This information requirements analysis will allow the EC to anticipate the information needs of the pilot throughout the mission, thereby contributing to the knowledge engineering effort. A major consideration is the timing requirements for this information. The timing requirements must be discussed in conjunction with the Pilot Intent/Human Performance Analysis.

Human Performance Analysis - The Human Performance analysis produces the HPM. The HPM must have a representation of the response characteristics of the airframe and subsystems, including the Electronic Crewmember. This representation will dictate demand for tasks to be performed during the course of the simulated scenarios. The aircraft model need only be detailed enough to account for behavior characteristics and response times given a specific control input.

The second portion of the HPM is a library of procedures and tasks. This library must be developed using the task analysis, and can be viewed as knowledge of the pilot's activities or potential activities during the mission. Alternative representations of tasks may be developed if a task can be performed by different resources, human and computer.

The final element of the HPM is the control system. This control system will scan the demands imposed by the aircraft and subsystem model, determine the highest priority demand and select the appropriate tasks or procedures to be executed. When the task or procedure is executed, it will eliminate or satisfy the appropriate task demand and allow the control structure to choose the next task, given the next highest priority demand. Thus the control model helps close the loop, representing a truly reactive system. The output of this model will reflect levels of the metrics developed in the task analysis. Examples

of the output would be resource utilization and delay time due to resource scarcity. These metrics will be used for all subsequent analysis.

Pilot Intent Model Development/Evaluation - As described earlier, the pilot intent model can be viewed as a reverse application of the HPM. The HPM must be developed to predict human activity under various circumstances throughout the mission. When this model is validated, it will be capable of predicting information demand through its identification of required tasks and procedures. By experimenting with various pilot intent strategies, this model will allow an evaluation of the appropriate strategy to be employed in the Pilot/Vehicle Interface. This design stage would be optional if it were not for the stated overlap between the HPM and PI. It is not necessarily a requirement for the next activity to be initiated.

Response Time Requirements Development - Once the human performance model has been developed, it can be used to investigate the necessary response time of the system. Given an event, such as detection of a threat, and the reaction times for the pilot, the response time of the system can be modeled. Knowing what the average allowable time for detection of a threat to its imminent impact, Monte-carlo studies can be performed to determine whether the system response time is adequate under all conditions.

Identify Automation Requirements - As a consequence of the system performance parameters gathered from the modeling efforts, suggested man-machine interactions will be identified as optimal. While this can be seen as identifying automation requirements, it actually only places lower bounds on the automation requirements. At this point, it is possible to evaluate the stabilized man-machine system in terms of acceptable variations in system configuration, man-machine interface, and performance. Referring to Figure 1, this includes an analysis of dynamic task allocation strategies, interactive modifications to the fixed display formats, and the tone of the man-machine operational relationship (level of autonomy, command modes, etc.).

APPLICATIONS OF PILOT INTENT/HUMAN PERFORMANCE MODEL

Applications Outside an Operational PVI

As was described in the discussion of HPM's, there are four major applications for the HPM. These applications include: developing response time requirements for the system to formulate information and present it to the pilot, investigating operational relationship strategies between the pilot and EC, investigating Dynamic Allocation strategies within each operational relationship, and developing strategies for load leveling the information flow to the pilot in order to prevent perceptual and cognitive saturation. By predicting the performance and workload of the pilot under various man-machine interaction strategies, an acceptable design can be chosen.

Applications Within the Operational PVI

Throughout a mission, two categories of pilot intent dictate the pilot's activities. The first includes goals that are applied continuously. These goals include mission specific purpose derived during mission planning, such as the intention to deliver ordnance. In addition, this category includes more general goals which apply to all missions, such as the intention to maintain a safe flight vector, to optimize the use of expendables, etc. The second category includes situation specific goals.

There are two ways for the EC system to detect situation specific goals. First, there are overt commands to the system, such as HOTAS input or high-level commands such as 'fence check', which cues the EC to perform a sequence of activities. The second method, a more covert means of detecting intent, is performed by inferring intent from observed behavior. We have presented a method for inferring intent, and the current situation, from observed behavior. Once these intentions, overt and covert, are made known to the EC system, appropriate decision and task aids can be formulated.

The 'associate' paradigm is the motivation for the EC's requirement to be cognizant of pilot intentions. An associate can notify, advise, assist, or execute activities with, or even for, the pilot. The degree of activity or autonomy with which the EC performs these various tasks will be the product of the system's capabilities as well as the pilot's momentary desires, expectations, and workload. A robust, contextual intent interpreter can direct the the EC system in executing the associate paradigm.

An integral component of the associate paradigm is adaptive aiding, which is the system's response to changes in the mission environment, pilot workload, and pilot intentions. The dynamic allocation of tasks to the pilot or EC, along with other aspects of adaptive aiding as implemented through the man-machine interface (shared activities, sequential behavior, etc.), will ensure that the EC system is always in step with the pilot's expectations and activities. This involves much more than the storage and identification of procedures. Operator-machine interactions will be a unique product of the current situation and the operator's current desires. At its most extreme, EC's knowledge of the pilot's goals allows the EC to anticipate the activity required of it, thereby making the EC operate in a more proactive, rather than reactive, manner.

The embedded HPM within PI subsystem will provide much of the knowledge necessary to maintaining the EC's awareness of the pilot's information requirements and task assistance expectations. In addition, it provides a method for applying this knowledge toward responding to the pilot and satisfying his goals. Figure 3 describes the use of the PI/HPM model to coordinate known pilot goals, pilot behavior, and mission/environment changes reported by the various Electronic Crewmember components.

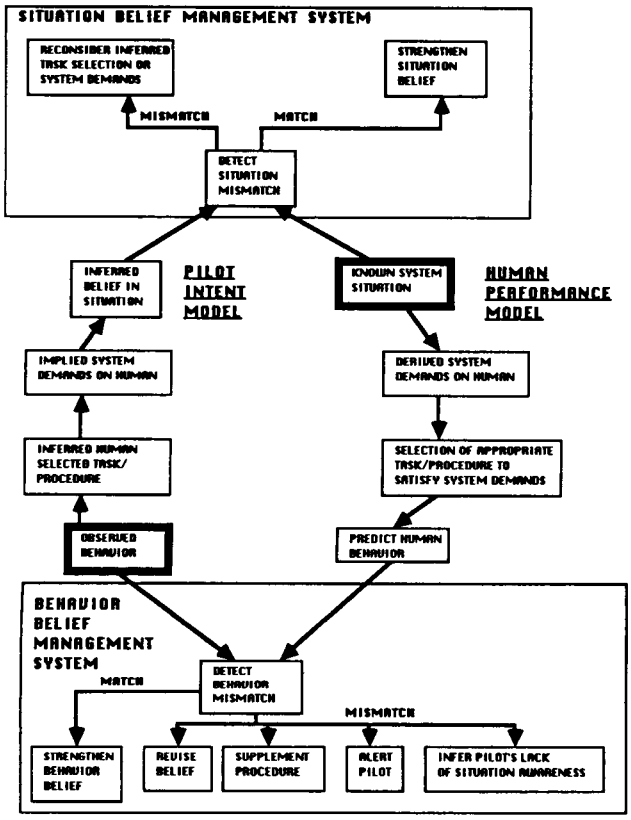


Figure 3. Combined Pilot Intent/Human Performance Model Logic Flow

The application of the model is triggered by two external events. The first event is observed operator behavior. If the observed behavior matches the expected behavior, then the behavior belief is strengthened (and therefore the system's model of the pilot). If the expected behavior does not match the observed behavior, then there are several possible responses to include: revision of behavior belief; supplementation of pilot's errors of omission in a procedure; alerting the pilot of an error of commission (procedure definitely does not meet known situation); or infer a lack of pilot situation awareness. Observed behavior is also used by PI subsystem to infer any changes in the current situation, which, as shown below, also triggers subsequent processing.

Changes in the known situation are the second trigger for system activity. As changes in the situation are reported by the various EC expert systems, matching is again done, this time against the PI subsystem generated estimate of the situation derived from activity. A match between the estimate and reality strengthens the PI subsystem's belief in the estimated situation. A mismatch indicates a need to reconsider any inferred pilot task selection or system demands. Situation changes reported by the EC subsystems are also propagated through the PI/HPM subsystem to update expected pilot behavior.

While 'observed behavior', and 'known system situation' are the only external inputs that cause PI subsystem activity, the changes that these inputs can cause, through the reasoning capabilities of the PI/HPM subsystems, to 'inferred belief in situation' or 'predict human behavior' can also trigger the matching process.

CONCLUSION

The Electronic Crewmember system will apply new technologies to the man-machine interface of future fighter aircraft. Within the man-machine interface, a requirement has been demonstrated for the system to be cognizant of the pilot's behavior and intentions. The Pilot Intent Subsystem satisfies this requirement by applying a combined pilot intent/human performance model. This machine cognizance of the pilot provides an enhanced decision and task support capability to the EC system. This creates an 'associate' paradigm approximating that of a human associate.

REFERENCES

Baron, S., Zacharias, G., Muralidharan, R., and Lancraft, R., "PROCRU: A model for analyzing flight crew procedures in approach to landing," Proceedings of Eight IFAC Work Congress, Tokyo, Japan, Vol. XV, pp. 71-76. (1980). (Also, NASA Report No. CR152397).

Martin, W. L., An Assessment of Artificial Intelligence and Expert Systems Technology for Application to the Management of Cockpit Systems, AAMRL-TR-87-040. (1986)

McCoy, M. S., Boys, R. M., "Human Performance Models Applied to Intelligent Decision Support Systems," Proceeding of NAECON '87. (1987)

Super Cockpit Industry Days Briefing sponsored by the Air Force Systems Command, Human Systems Division, AAMRL. (March 31-April 1, 1987)

Wellens, A. R., McNeese, M. D., "A Research Agenda for the Social Psychology of Intelligent Machines", Proceeding of NAECON '87. (1987)

THE EMPIRICAL ACCURACY OF UNCERTAIN INFERENCE MODELS

David S. Vaughan
Robert M. Yadrick
Bruce M. Perrin
McDonnell Douglas Astronautics Company
P. O. Box 516
St. Louis, MO 63166

Ben P. Wise
Thayer School of Engineering
Dartmouth College
Hanover, NH 03755

ABSTRACT

Uncertainty is a pervasive feature of the domains in which expert systems are designed to function. Several methods have been used for handling uncertainty in expert systems, including probability-based methods, heuristics such as those implemented in MYCIN, methods based on fuzzy set theory and Dempster Shafer theory, and various other schemes. This paper reviews research designed to test uncertain inference methods for accuracy and robustness, in accordance with standard engineering practice. We have conducted several studies to assess how well various methods perform on problems constructed so that correct answers are known, and to find out what underlying features of a problem cause strong or weak performance. For each method studied, we have identified situations in which performance is very good, but also situations in which performance deteriorates dramatically. Over a broad range of problems, some well-known methods do only about as well as a simple linear regression model, and often much worse than a simple independence probability model. Our results indicate that some commercially available expert system shells should be used with caution, because the uncertain inference models that they implement can yield rather inaccurate results.

INTRODUCTION

Uncertainty is a pervasive feature of many domains in which artificially intelligent expert systems are intended to function. Researchers in artificial intelligence have proposed a variety of approaches to uncertain reasoning. Some (e.g., 1, 2, 3) have developed methods that are explicitly based on probability theory. Other approaches, such as those used in MYCIN (4, 5), PROSPECTOR (6), and AL/X (7), use heuristics designed to approximate probability theory. Yet other methods involve adaptations of fuzzy set theory (8), Dempster-Shafer theory (9), and other ideas not based on probability. Unfortunately, there is no wide consensus concerning which approach is best or even suitable for any particular application.

Some researchers have attempted to compare these various approaches through theoretical analysis. For example, Heckerman (10) has shown that the equations which define MYCIN's certainty factors can be translated into probabilistic terms. As another example, Hunter (11) has investigated conditions under which probability theory and Dempster-Shafer theory agree.

Although theoretical analyses can provide useful insights, they also become exceedingly complex and their usefulness for the average practitioner can decrease, particularly when heuristics which have no particular theoretical justification are being considered. Furthermore, these analyses typically focus on the formal assumptions of the various uncertainty models, which are seldom met in practice. Perhaps the most important questions concern how models behave when assumptions are not met.

The present authors and various additional coauthors have taken a different, empirical approach to examining the accuracy of uncertain inference models in a series of studies. We started working independently, but eventually realized the commonalities in our work and began to collaborate.

It should be made clear that we are examining the basic inference models used by systems such as MYCIN. We are not evaluating any particular implementation that uses any given model. In our general approach, answers provided by probability theory are used as a norm against which the accuracy of other uncertain inference models may be measured. These studies differ in details, but all use the same basic research paradigm. First, example inference networks are constructed so that all relevant parameters are known. Next, new values are assigned to the evidence nodes, as though additional information in the form of updated estimates is being supplied by a user during a consultation session. Conclusion node certainty values are calculated which reflect the new information according to the model under consideration. Finally, these answers are compared to results obtained from a probability-based method which provides the minimum cross-entropy solution (12). This approach parallels methods used in various scientific and engineering disciplines, such as sensitivity analysis and "Monte Carlo"

simulations, for investigating the behavior of complex systems when assumptions are violated.

We have completed studies which evaluate the MYCIN model, the PROSPECTOR model, probability-based models which contain simplifying assumptions (e.g., independence) and a simple linear model. The objectives of the present paper are to review and summarize these studies, describe the major objectives and findings of each, and discuss the overall implications of these findings for expert system construction and future research.

Table I summarizes the studies that will be reviewed. All of these studies used the general method described above. However, they differed in certain important respects as well. Some focused on only a single uncertain inference model, while others looked at several models simultaneously. Some used many small, randomly-created inference nets, while others used larger, selected nets. Finally, some of these studies derived model parameters by using published theoretical definitions to translate the nets directly, while others "tuned" the parameter values. These issues are all explained and discussed in more detail below.

STUDIES USING THEORETICAL PARAMETERS

In our methodoloy, inference nets are created, solved by a minimum cross-entropy extension of probability theory, and also solved by another uncertain inference model. A key part of this process involved translating between parameters suitable for the probability calculations and parameters required by the other model. For example, the MYCIN model expresses rule strengths (relationships between evidence and conclusions) in measures of believe (MBs) and measures of disbelief (MDs). The developers of MYCIN provided theoretical definitions of these parameters in probability terms. In the first three studies shown in Table I, such theoretical definitions were used for the necessary translations. Consider these studies:

STUDY	UNCERTAIN INFERENCE MODELS	INFERENCE NETS	MODEL PARAMETER ESTIMATION
Wise (13)* Wise & Henrion(14) Wise (15)	MYCIN probability with assumptions	large, selected	theoretical definition
Perrin, Vaughan, Yadrick, Holden, & Kempf (16)	MYCIN	many, small	theoretical definition
Yadrick, Perrin, Vaughan, Holden, & Kempf (17)	PROSPECTOR	many, small	theoretical definition
Wise, Perrin, Vaughan & Yadrick (18)	MYCIN PROSPECTOR probability with assumptions linear regression	many, small	tuned
Wise (19)	PROSPECTOR probability with assumptions linear regression	many, small, selected	tuned

* Wise & Henrion (14) and Wise (15) both contain summaries of results which are presented in more detail in Wise (13).

Table I Evaluation Studies Reviewed

o Wise

Wise (13) presented a detailed theoretical analysis of the MYCIN model, as well as several other models. He also discussed in detail the rationale for accepting the minimum cross-entropy probability solution as an appropriate criterion for evaluating other uncertain inference models. Highlights of this work appear in Wise & Henrion (14), which presents the methodology and some preliminary results, and in Wise (15), which summarizes results for the MYCIN model.

The MYCIN model (5) was one of the first to be used for handling uncertainty in an expert system. It was designed to solve some problems that the developers believed made probability theory unsuitable for their application. The model was also designed to approximate probability calculations while being modular, computationally efficient, and more natural for their subject matter experts to use. The original concerns the developers had with probability theory are probably not valid (e.g., 20). However, the model and several variants are widely cited and used today, particularly in several commercial "shells". Thus, information about the accuracy of the MYCIN model continues to have practical relevence for a large community.

Based on his detailed theoretical analyses and on critical examples cited in the literature, Wise constructed some sets of inference nets with associated rule strengths (defined as probabilities) for which the MYCIN model was predicted to be reasonably accurate, and some for which large errors were predicted. The resulting 30 nets ranged in size from the simplest (two evidence nodes and one conclusion node) to nets with three evidence nodes, multiple conclusion nodes and an intermediate node level. One net comprised nine evidence nodes, four intermediate nodes, and four conclusion nodes. Correlations between pieces of evidence were also varied systematically between strong positive and strong negative associations. With two exceptions, rules in the nets were conjunctive ("AND") rules. To generate test problems, he systematically varied "updated" or input evidence probabilities over four values for each evidence node. This means, for example, that a net with three evidence nodes yielded 64 (4x4x4) problems. Each problem was solved using the probability model, the MYCIN model, and several models based on probability theory with simplfying assumptions. For each inference net he computed $\underline{e}$, the mean squared difference between the maximum entropy probability answers and the inference model answers across the set of problems.

The MYCIN model was most accurate for cases in which there was very little difference between the base rate (prior probability) of the conclusion and the conditional probability of the conclusion when both pieces of evidence were false (or absent). For example, the value of $\underline{e}$ .0004 in one such case and .0005 in another. Conversely, MYCIN was most inaccurate when there was a large difference between the conclusion base rate and the conditional probability of the conclusion when both pieces of evidence were false. The value of $\underline{e}$ was .03, .09, .03, and .04 in four such cases. These results are attributable to two features of the MYCIN model. First, based on theoretical definitions, MYCIN ignores negative evidence. That is, if the updated probability for a piece of evidence is greater than the prior probability (base rate) for that evidence, MYCIN updates conclusion probabilities associated with the evidence. However, if the updated probability for a piece of evidence is below the base rate, MYCIN ignores that information and conclusion probabilities are not updated. The other feature concerns the method used to combine evidence for conjunctive (AND) rules. This method "pays attention to" only one of the pieces of evidence involved. As a consequence of these features, MYCIN provides accurate answers when the impact of ignoring negative evidence is minimized, i.e., when the conditional probability of the conclusion is high given that the evidence is absent.

The robustness of an uncertain inference model can be assessed by examining reasons for its worst performance. In this light, Wise compared the MYCIN model to a simple probability model which assumes conditional independence. Across the set of nets he studied, the conditional probability model was considerably more robust (largest $\underline{e}$ = .04) than the MYCIN model (largest $\underline{e}$ = .09). The MYCIN model was very accurate on some nets, but very inaccurate on other nets.

o Perrin, et al.

Perrin, Vaughan, Yadrick, Holden, and Kempf (16) also studied the MYCIN model. However, the inference nets were constructed in a somewhat different way. In this study, only the simplest sorts of nets were studied, i.e., those comprising two pieces of evidence and one conclusion. These networks are the basic building blocks of larger networks; inference in these nets requires both evidence combining and propagation. In this study, many nets were constructed by random sampling from the universe of three-node nets. In particular, 200 nets were compiled in which the pieces of evidence were independent and 200 were compiled in which the pieces of evidence were statistically associated. Problems were generated by independently varying the updated evidence probabilities over five values. Since all nets had two evidence nodes, this created 25 problems for each net. As in (13), each problem was also solved using the minimum cross-entropy probability model, Next, each net was translated into MYCIN parameters using the theoretical definitions, and was solved using all three of MYCIN's combining functions (conjuctive, disjunctive, and incremental).

Networks were classified according to which combining function provided the lowest error for the network. The incremental function was the most accurate for about 60% of the networks, the conjuctive function was the most accurate for about 35%, and the disjuctive function was most accurate for the remaining 5%. The mean absolute error across all nets was about .07, while the average maximum error per net was about .22. Further analysis indicated that much of this error was due to MYCIN's ignoring negative evidence. For only problems in which MYCIN updated conclusion probabilities, the mean error across problems and nets was about .02, and the average maximum error was about .05. Further analysis indicated that MYCIN error was greatest in these problems when evidence base rates were low and evidence-conclusion associations were strong. These attributes characterize the difficult diagnostic process; the results suggest that MYCIN will be least accurate in precisely the situations for which expert systems are likely to be most valuable.

o Yadrick, et al.

Yadrick, Perrin, Vaughan, Holden, & Kempf (17) studied the model used in the PROSPECTOR system (6). Like the MYCIN model, the PROSPECTOR model was developed to address perceived problems with probability theory for expert system applications. It was also intended to approximate probability calculations while being computationally efficient and modular. While this model has received less attention than the MYCIN model, it and several variants (e.g., AL/X) have also been implemented in commercially-available expert system shells.

Yadrick, et al. used the same inference net and problem generation methods as Perrin, et al. All networks contained two evidence nodes and one conclusion node. A total of 400 networks were sampled which contained independent evidence and 400 nets were sampled which contained associated evidence. Again, 25 problems were generated for each net and solved using maximum entropy probability calculations. The problems were translated into PROSPECTOR parameters using theoretical definitions and the problems were solved using PROSPECTOR conjunctive, disjunctive, and independent rule combining functions. For each net, the mean squared error was computed and the maximum error for a single problem was recorded.

PROSPECTOR error was quite large (often greater than .5) for many nets. Extremely large errors were found mainly for nets in which the probability of the conclusion was high if one piece of evidence was true and one was false, but was not as high if both pieces of evidence were either true or false We concluded that the PROSPECTOR model is fundamentally incapable of handling these "counterintuitive" nets, and excluded them from futher analysis. This left 66 independent and 73 associated evidence nets for additional consideration.

The independence combining function was most accurate for about 90% of the remaining independent nets and about 80% of the remaining associated nets. The overall average error was about .014 for independent nets and about .022 for associated nets; overall maximum error was about .055 for independent nets and about .083 for associated nets. Further analysis indicated that error was greatest when the evidence is most strongly associate with the conclusion. Moreover, the error can be compounded or mitigated by the values of updated evidence probabilities. In summary, the PROSPECTOR model was quite accurate for some problems and networks, but very inaccurate for others over a wide range of new evidence probabilities. Like MYCIN, it appears to be least accurate in the typical situations to which it would likely be applied.

TUNED MODEL PARAMETERS

The studies described above used published formal definitions to translate between probability model parameters and uncertainty model parameters. The hope was to determine the absolute degree of error and provide a theoretical explanation for sources of error produced by the uncertainty models. Despite some success in this, practical applications of the findings are limited, precisely because we used formal uncertainty model parameter definitions. These theoretical definitions have little relevence to knowledge engineers building real expert systems, because parameters are typically estimated by experts based on an intuitive rather than a formal understanding. Then the parameters are "tuned", or adjusted interactively by the experts and knowledge engineers to obtain the most accurate results on the data used for system development. The relationship between parameters estimated in this way, the formal definitions of the parameters, and probability theory is not clear. Furthermore, the tuning process may correct some or all of the errors observed in the studies described above.

This tuning issue lead us to do two additional studies (18, 19). The objective was to study the errors made by uncertain inference models empirically after their parameters have been tuned. As before sample networks were created, and problems were run by systematically varying updated evidence probabilities. Problem solutions produced by uncertain inference models were compared to the same minimum cross-entropy probability norm. This time, however, the model parameters were optimized for each net ("tuned") so that the model's answers were as close to the probability answers as possible, on the average. These solutions, therefore, represent the best performance that could be achieved by each model.

o Wise, et al.

Four different inference methods were examined by Wise, Perrin, Vaughan, & Yadrick (18). These included the MYCIN and PROSPECTOR models as before. We also included a linear regression model, given by the following equation:

$$P'(C) = a + b1*P'(E1) + b2*P'(E2). \quad (1)$$

In this equation and below, P'(x) is the updated probability for the event x, and a, b1, and b2 are constant parameters, which were optimized. Linear models have received little attention from artificial intelligence researchers, although they have been used successfully to model a variety of human judgments (21). We included this model to provide a baseline against which to compare the other models.

Finally, a probability theory-based independence model was also included. This model is described by the following equation:

$$\begin{aligned} P'(C) = {} & P'(\sim E1) * P'(\sim E2) * P(C|\sim E1\&\sim E2) + \\ & P'(E1) * P'(\sim E2) * P(C|E1\&\sim E2) + \\ & P'(\sim E1) * P'(E2) * P(C|\sim E1\&E2) + \\ & P'(E1) * P'(E2) * P(C|E1\&E2). \end{aligned} \quad (2)$$

The model reflects normal probability calculations under the assumption that the pieces of evidence are independent. The four conditional probabilities are the model parameters (which were optimized). After parameter optimization, this model is equivalent to a linear regresion model with an interaction term.

A total of 109 two-evidence, one-conclusion networks were sampled using procedures similar to those of (16) and (17). For each piece of evidence, the updated probabilities varied over five values so that 25 problems were run for each network. For each model, parameter values were obtained which minimized, across the 25 problems, the sum of squared differences between the model solutions and the minimum cross-entropy answers. This optimization was done using a deflected gradient search algorithm (22) with appropriate precautions to avoid local minima and round-off error problems. Table II summarizes the performance of the four models.

The main finding was that the MYCIN (5 parameters), PROSPECTOR (7 parameters), and linear (3 parameters) models performed equally well (for all practical purposes), while the independence (4 parameters) model was significantly more accurate (according to an analysis of variance test). Furthermore, the errors for the MYCIN, PROSPECTOR, and linear models were highly correlated (Pearson product-moment coefficient >.95). This shows

INFERENCE METHOD	AVERAGE RMSE	HIGH RMSE	LOW RMSE
MYCIN	.048	.152	.001
PROSPECTOR	.047	.148	.001
Linear eq.	.048	.152	.001
Independence	.006	.036	.000

Note: This table was taken from (18). RMSE is root mean squared error.

Table II Tuned Parameter Errors

that the models all performed well or poorly on the same problems. They were behaving almost identically for the networks studied here, although the linear model requires estimation of fewer parameters. A probability theory-based independence model performed better and required fewer parameters than MYCIN or PROSPECTOR.

o Wise

The objective of this study (19) was to determine the degree to which errors of the sort shown in Table I can be attributed to assumption violations in the networks. The study included the PROSPECTOR model, the linear and marginal independence models (equations 1 and 2), and a model that was linear on logarithms of odds ratios (i.e., it substituted logs of odds ratios for probabilities in equation 1). The general methodology, including network and problem generation and model parameter optimization, were the same as (18). Here, however, all networks were constructed to meet the PROSPECTOR model's conditional independence assumptions. Thus, all error for PROSPECTOR in these networks must be due to the approximate updating functions.

Table III summarizes results for the conditional independence networks. In this table, errors are expressed in terms of a

	INDEPENDENCE	PROSPECTOR	LOG-ODDS
Mean	.90	.52	-.36
Standardized error	.08	.35	.07

Note: This table summarized from (X). Cell entries are standardized error measure (see text).

Table III Errors for Conditionally Independent Networks

standardized measure, where 1.0 reflects no error and 0.0 reflects the same level of error as the linear model. Positive scores on this measure indicate better performance than the linear model, and negative scores indicate worse performance than the linear model. As

may be seen, the PROSPECTOR model performs better than the linear model when networks meet the model's assumptions. However, due to the updating procedure, the model still performs worse than the independence model and still makes substantial errors.

DISCUSSION

We think that these results have a number of implications for expert system construction.

First, it is clear that both the MYCIN and PROSPECTOR models are suitably accurate under some circumstances, but can make large errors under other circumstances. Exactly which model and what parameter values are used make a potentially important difference in the overall accuracy of an expert system. It may not be possible to tune the system to perform with reliable accuracy across a broad range of problems, users, and solutions. In short, under some circumstances one should probably not use the MYCIN or PROSPECTOR models. This conclusion is important, since these models are embedded in many commercial shells and are widely used. Indeed, neither these nor other models should be used uncritically, without investigations to determine appropriateness to the particular application under consideration.

Second, very simple models may work well for many problems. A simple linear model worked as well as the MYCIN and PROSPECTOR models, and a probability-based independence model worked much better. Elaborate models have been developed to handle uncertainty in expert systems, but the elaborations add little to accuracy and are very sensitive to differences in, for example, evidence-conclusion relationships.

All of the uncertain inference models made substantial errors under some circumstances. This suggests that for some difficult applications, custom-built uncertain inference models may still be required. The system builder should select or develop a method that is neither too simple nor too complex for the application at hand.

When an uncertain inference model is being considered, one need not focus entirely on the assumptions of the model and whether those assumptions are met in the application. We have found that some models work well even when assumptions are not met (e.g., the probability-based independence model and the linear model) and that others may work poorly even if assumptions are met. We believe that robustness is more important than theoretical elegance in practical expert system building.

Finally, we believe that the empirical approach to evaluating uncertain inference model accuracy and the general methodology we have developed is useful. The findings summarized above have shed new light on the performance of such models, which goes beyond theoretical analyses. However, many questions remain unanswered. Our studies have looked at only a few models and only at simple networks. While it seems likely to us that errors will tend to propagate and compound in many large networks, that other heuristic models will perform poorly in many circumstances, these issues should be settled empirically. We are presently investigating these and other issues.

Acknowledgment — Some of the research reviewed in this paper was conducted under the McDonnell Douglas Independent Research and Development program.

REFERENCES

1. Pearl, J., "How to do with probabilities what people say you can't", Proceedings of the IEEE Second Conference on Artificial Intelligence Applications, Miami, FL, 1985.

2. Cheeseman, P., "A method of computing generalized Bayesian probability values for expert systems", Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, 1983.

3. Vaughan, D.S., Perrin, B.M., Yadrick, R.M., Holden, P.D., and Kempf, K.G. "An odds ratio based inference engine," UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, Kanal, L.N. & Lemmer, J.F. (Eds.), North-Holland, Amsterdam, 1986. 383-389.

4. Shortliffe, E.H. & Buchanan, B.G., "A model of inexact reasoning in medicine," MATHEMATICAL BIOSCIENCES, 23, 351-379, 1975.

5. Shortliffe, E.H., COMPUTER-BASED MEDICAL CONSULTATION: MYCIN, American Elsevier, Amsterdam, 1976.

6. Duda, R.O., Hart, P.E., Konolige, K., & Reboh, R., "A computer-based consultant for mineral exploration, Final Report, Project 6415, SRI International, Menlo Park, CA, 1979.

7. Reiter, J., "AL/X: An expert system using plausible inference," Intelligent Terminals, Ltd., Oxford, 1980.

[8] Zadeh, L.A., "The role of fuzzy logic in the management of uncertainty in expert systems," MEMORANDUM # UCB/ERLM83/41, Electronics Research Laboratory, University of California, Berkeley, CA, 1983.

9. Shafer, G., "Probability judgment in artificial intelligence", UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, Kanal, L.N. & Lemmer, J.F. (Eds.), North-Holland, Amsterdam, 1986, 127-135.

10. Heckerman, D. "Probabilistic interpretation for MYCIN's certainty factors," UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, Kanal, L.N. & Lemmer, J.F. (Eds.), North-Holland, Amsterdam, 1986, 167-196.

11. Hunter, D., "Dempster-Shafer vs. Probabilistic Logic", Proceedings of the Third AAAI/Martin Marietta/ADS Workshop on Uncertainty in Artificial Intelligence, Seattle, WA, July, 1987.

12. Shore, J.E. & Johnson, R.W., "Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy," IEEE TRANSACTIONS ON INFORMATION THEORY, Vol. IT-26, 1980, 26-37.

13. Wise, B.P., "Experimental comparison of uncertain inference systems," Unpublished Ph.D. Dissertation, Department of Engineering and Public Policy, Carnegie-Mellon University, Pittsburgh, PA, 1986.

14. Wise, B.P. & Henrion, M., "A framework for comparing uncertain inference systems to probability," UNCERTAINTY IN ARTIFICIAL INTELLIGENCE, Kanal, L.N. & Lemmer, J.F. (Eds.), North-Holland, Amsterdam, 1986, 69-83.

15. Wise, B.P., "Experimentally comparing uncertain inference systems to probability," Proceedings of the Second RCA/AAAI Workshop on Uncertainty in Artificial Intelligence, Philadelphia, PA, August, 1986. To appear in a forthcoming book from North-Holland.

16. Perrin, B.M., Vaughan, D.S., Yadrick, R.M., Holden, P.D., & Kempf, K.G., "Evaluation of uncertain inference models II: MYCIN, MDC TECHNICAL REPORT # E3205, McDonnell Douglas Corporation, St. Louis, MO, July, 1987.

17. Yadrick, R.M., Perrin, B.M., Vaughan, D.S., Holden, P.D., & Kempf, K.G., "Evaluation of uncertain inference models I: PROSPECTOR," Proceedings of the Second RCA/AAAI Workshop on Uncertainty in Artificial Intelligence, Philadelphia, PA, August, 1986. To appear in a forthcoming book from North-Holland.

18. Wise, B.P., Perrin, B.M., Vaughan, D.S., & Yadrick, R.M., "The role of tuning uncertain inference systems," Proceedings of the Third AAAI/Martin Marietta/ADS Workshop on Uncertainty in Artificial Intelligence, Seattle, WA, July, 1987.

19. Wise, B.P., "Satisfaction of assumptions is a weak predictor of performance," Proceedings of the Third AAAI/Martin Marietta/ADS Workshop on Uncertainty if Artificial Intelligence, Seattle, WA, July, 1987.

20. Cheeseman, P., "In defense of probability," Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Los Angeles, CA, August, 1985.

21. Dawes, R.M., "The robust beauty of improper linear models in decision making," AMERICAN PSYCHOLOGIST, 34, 571-582.

22. Beightler, C.S., Phillips, D.T., & Wilde, D.J., "FOUNDATIONS OF OPTIMIZATION, Prentice-Hall, Englewood Cliffs, NJ, 1979.

Reasoning Under Uncertainty

(Paper not written)

Decision-Theoretic Control of Planning Under Uncertainty

(Paper not provided by publication date)

AUTONOMOUS CONTROL PROCEDURES FOR SHUTTLE RENDEZVOUS PROXIMITY OPERATIONS

Robert N. Lea and Eric V. Mitchell

NASA/Johnson Space Center, Houston, Texas

Mary Ann Goodwin

Rockwell Shuttle Operations Company

ABSTRACT

This paper presents the results of a study which uses fuzzy sets to model a Space Shuttle pilot's reasoning and actions while performing rendezvous proximity operation maneuvers. Use of conventional pilot models is limited since they often results in unrealistic overfirings and, therefore, excess fuel usage and unacceptable playload contamination and plume disturbances. In this model fuzzy sets are used to simulate smooth and continuous actions as would be expected from an experienced pilot and to simulate common sense reasoning in the decision process. The present model assumes visual information available to the Shuttle pilot from the Shuttle Crew Optical Alignment Sighting (COAS) device and the overhead window and rendezvous radar sensor information available to him from an onboard display.

This model will be used in a flight analysis simulator to perform studies requiring a large number of runs, each of which currently needs an engineer in the loop to supply the piloting decisions. A great deal of the engineer's time is required for this repetitious and somewhat boring function. This work has much broader implications in control of robots such as the Flight Telerobotic Servicer, in automated pilot control and attitude control, and in advisory and evaluation functions that could be used for flight data monitoring or for testing of various rule sets in flight preparation.

INTRODUCTION

The manual phase of Space Shuttle rendezvous begins several nautical miles from the target when the crew starts maneuvering the Shuttle using visual and sensory information. This portion of rendezvous is referred to as terminal phase.

Once terminal phase begins the crew can fly a number of scenarios, or paths, to approach the target. One sequence is illustrated in figure 1. The coordinate system is centered at the target, and has a v-bar axis tangential to the target's orbit and positive in its direction of motion. The r-bar axis is perpendicular to the v-bar axis with the positive direction towards the center of the earth. The Shuttle flies to the v-bar axis several hundred feet in front of the target. There the upward velocity is nulled to zero, and the vehicle begins "closing" toward the target by decreasing its velocity relative to the target's velocity so that the two will eventually rendezvous.

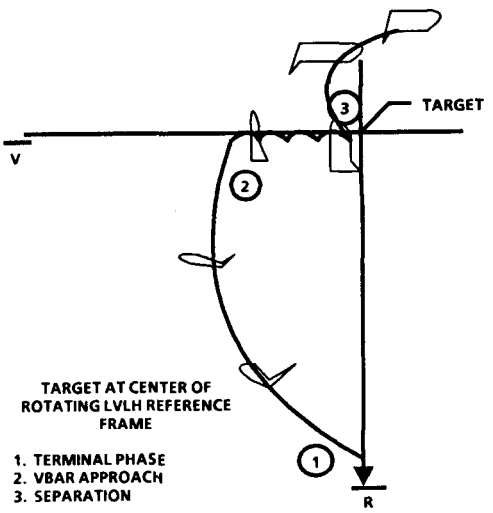


figure 1

One simulation available for these so called "proximity operations" is the Shuttle Engineering Simulator (SES), a high fidelity simulation complete with cockpit. It is expensive to operate, heavily scheduled and

must frequently be reserved for crew training. For certain engineering studies that do not require SES fidelity, a "desk top" simulator [1, 2] implemented on an HP 9000 is adequate. A "pilot" controls the "vehicle" by input through a keyboard or, sometimes, a joystick. Decisions to "fire jets" are made based on CRT graphical representations of the view through the Shuttle overhead window, the scenes from closed circuit television cameras mounted in the payload bay and the view through the Crew Optical Alinement Sight (COAS). Digital data on the CRT represents information provided the Shuttle crew through onboard CRT displays. Included in this data are rendezvous radar measurements of range and range rate, which are updated at the simulation integration rate.

There are several motives for wanting to program an automatic pilot that can "fly" this simulation. An automatic pilot that realistically represented human response could make it possible to obtain many executions in a batch mode that presently require an engineer in the loop. If a number of executions are needed to obtain statistics on various parameters, such as fuel usage, contamination, or plume impingement, a great deal of engineering time can be required. Run time could be reduced since creation of the graphic output at a realtime response rate could be eliminated. Further, once an engineer playing pilot has performed ten or so simulations of a proximity operations scenario, the effort of learning for that scenario is unrealistically high, which causes the results to be overly optimistic.

Existing automatic pilots for a batch-mode simulation were found to be almost as labor intensive as having an engineer in the loop. A given scenario might need to be executed repeatedly and "tuned" to determine when and what thrusters should fire before the desired results can be obtained. This effort was necessary to avoid the overfirings that frequently resulted in excess fuel usage and unaccecptable payload contamination and plume disturbances due to the "crispness" of the commands in the logic coded to do the piloting.

The need then is to create decision making logic that results in the same common sense decisions a pilot would make. Since the pilot uses his experience combined with the imprecise visual and digital information available to him, it appeared that fuzzy logic would provided a good basis for simulating his decision making. Furthermore, it was realized that this approach could be adapted to a number of areas of interest, such as development of translational and rotational digital autopilots, telerobotics, remote vehicle control, as well as ground or onboard advisory and evaluation functions.

A low level study effort has been underway for several months. The results and status to date are presented below.

DESIGN AND IMPLEMENTATION

Design and implementation of the pilot proceeded as follows. A set of rules of the kind observed and followed by Shuttle pilots flying a proximity operations scenario of the type illustrated in figure 1 was developed by observing and communicating with pilots of simulators used in Shuttle training and evaluation and testing at the Johnson Space Center. These rules were stated in natural language as they were related to us by the various pilots. The rules deal with the Shuttle keeping both the desired vertical distance and the desired closing velocity with respect to the target. The visuals that were used in this study were restricted to the COAS and the overhead window, which are illustrated in figure 2, and the digital data display of range and range rate from the rendezvous radar.

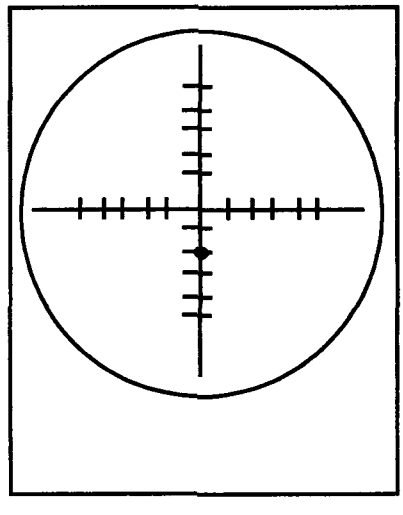


figure 2

Typical rules maintaining the desired vertical relationship are of the following form:

> If the target is located near the center of the field of view of the COAS , then no action is needed to raise or lower the shuttle.
>
> If the target is significantly above or below the center of the field of view of the COAS, then jets must be fired to lower or raise the shuttle.

The following rule is used to maintain the desired horizonal closing relationship:

> If the Shuttle is at a braking gate, e.g., 1000 ft, 500 ft, 400 ft, ..., from the target, then the closing velocity should be 1.0 fps, .5 fps, .4fps, ...

To implement these rules using fuzzy sets it was necessary to examine more closely what was meant by phrases such as "near" the center and "significantly" above or below. These terms are definitely descriptors of sets with fuzzy boundaries.

It was decided to use the π and S functions as given in [3, 4] since they can easily be adjusted for different degrees of "fuzziness" by varying the parameters that define their width and shape. The equations of the π and S functions are given below.

$$
\begin{aligned}
S(x,a,b,c) &= 0 && \text{for } x \leqq a \\
&= 2((x-a)/(c-a))^{**}2 && \text{for } a \leqq x \leqq b \\
&= 1 - 2((x-c)/(c-a))^{**}2 && \text{for } b \leqq x \leqq c \\
&= 1 && \text{for } x \geqq c
\end{aligned}
$$

$$
\begin{aligned}
\pi(x,b,c) &= S(x, c-b, c-b/2, c) && \text{for } x \leqq c \\
&= 1 - S(x, c, c+b/2, c+b) && \text{for } x \geqq c
\end{aligned}
$$

Graphs of the general S and π functions are given in figure 3.

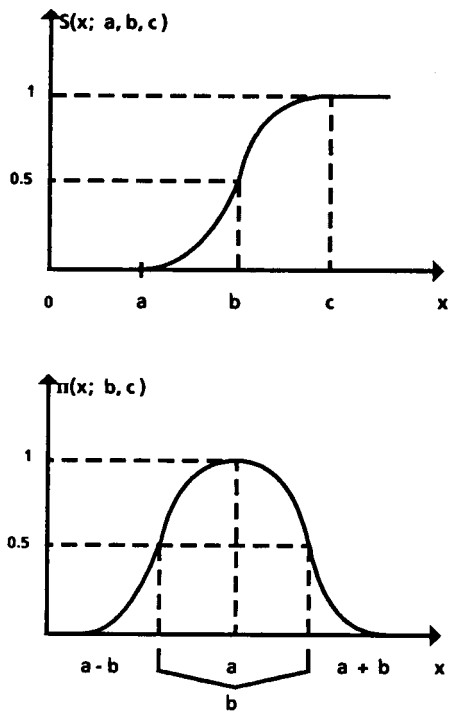


figure 3

As can be seen, one can effect a rapid or slow transisition from complete membership to complete non-membership by altering the parameters a, b, and c or b and c for the S or π function respectively. For example, the graphs of the S and π functions used for the maintainance of vertical position are given in figure 4. These functions allow flexibility in simulating different style pilots, for example, those who attempt to keep the target very close to the center of the field of view at all times, those who are more relaxed and are concerned mostly with keeping the target in view and using orbital mechanic effects to their advantage, and any type of pilot between these extremes.

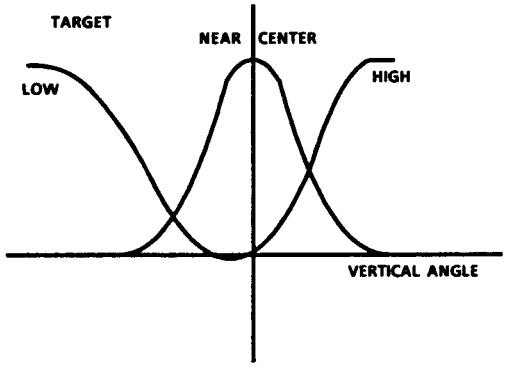


figure 4

The functions are used as follows. Fuzzy sets were defined for "somewhat greater than", "somewhat less than", and "approximately equal to" the desired closing rate. They were also defined for "high", "low", and "near the center" in the field of view. Approximately every twenty seconds, a time interval considered sufficient for a pilot to deal with the visual and displayed noisy range and range rate information, the fuzzy sets were evaluated and the maximum value recorded for each set of three functions corresponding respectively to closing rate and location in the field of view. Indices identifying the fuzzy function with the larger value for each set were recorded. If both indices corresponds to the no change fuzzy functions then no action is taken, if exactly one indicates no change then the action corresponding to the other maximum value is taken, and if both indicate action should be taken then the action corresponding to the larger of the two is taken.

The action to be taken, which is a certain number of jet firings, is determined as follows. The velocity change required to effect a vertical position change or to increase or decrease the closing rate is divided by the proper setting of the digital autopilot (DAP). The DAP has two settings that are preloaded with values that control the magnitude of the jet firings. Typical values are 0.02 and 0.05 which translates into 0.02 or 0.05 feet per second change in velocity per pulse depending on which value has been selected. The nominal DAP setting is the larger of the two and is the proper setting if it is smaller than the required velocity change. If this setting exceeds the required velocity change then the proper setting is the smaller value. The selected number, which could be considered the appropriate number of pulses under ideal conditions, is then weighted by multiplying by the fuzzy set evaluation that has been saved. This is the number of pulses that is commanded to the jets in the required direction.

To illustrate the number of pulses computation consider figure 5.

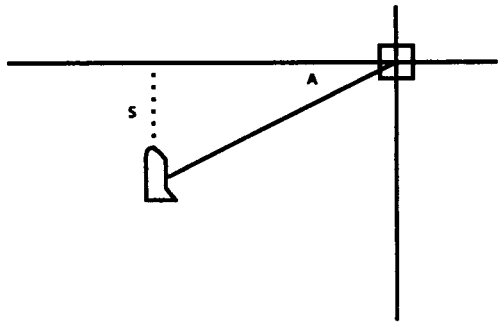


figure 5

In this case the target is high in the field of view. By using the shuttle-target range the angle A can be used to compute s, the distance below the target. Using the equation $\Delta h = .56\ \Delta v$, which relates Δh in nautical miles to Δv in feet per second one can estimate the required Δv to move the shuttle up to the v-bar. This estimate of Δv is adjusted according to whether the shuttle is currently moving up or down relative the target. This Δv can then be converted into an "ideal" number of pulses by dividing by the DAP setting, d. If f(A) is the evaluation of the fuzzy function corresponding to target high in the field of view then the number of pulses to be applied is given by

$$N = (\Delta v/d)^* f(A)$$

Closing rate control is easier in theory since one only has to difference actual closing rate and desired closing rate and divide by the DAP setting. In practice however it is considerably harder since range rate obtained from the radar is corrupted by noise and bias which at close ranges can be on the order of magnitude of the closing rates that need to be maintained. This problem, which is essentially one of modeling a human method of visually monitoring a stream of data and extracting the center of the data, is currently being considered. Some initial efforts look promising but have not been tested sufficiently at this time.

RESULTS

The automatic pilot can presently perform a terminal phase approach to the v-bar and a closing approach along the v-bar and maintain the desired line of sight and closing rates in a manner that compares reasonably well to the response of a "nominal" pilot. A "nominal" pilot is considered one that performs maneuvers in a manner recognized as appropriate in flight planning material. "Nominal" means the pilot does not allow the Shuttle's position and velocity to deviate greatly from the planned scenario.

The following table shows propellant consumption data tabulated from flying five dispersed proximity operation profiles of the type illustrated in figure 1. Both manually controlled and automated pilot controlled are given and it can be seen that the automated pilot controller compares favorably to the manually flown profile. It is at least within ten percent for each case. Dispersed means that the beginning states are randomly selected and that the noise and bias in the rendezvous radar are randomly varied within radar specification limitations.

ORBITER RCS PROPELLANT CONSUMPTION

MIL - MAN - IN - A - LOOP

ORBITER V-BAR APPROACH FROM 500 TO 60 FT FLIGHT PROFILE SCENARIOS	** PROPELLANT CONSUMPTION TOTAL (LBS)
(MIL) APPROACH DESPERSED CASE 1	101.0
(MIL) APPROACH DESPERSED CASE 2	107.0
(MIL) APPROACH DESPERSED CASE 3	126.0
(MIL) APPROACH DESPERSED CASE 4	119.0
(MIL) APPROACH DESPERSED CASE 5	116.0
AUTO - PILOT APPROACH DESPERSED CASE 1	110.0
AUTO - PILOT APPROACH DESPERSED CASE 2	115.0
AUTO - PILOT APPROACH DESPERSED CASE 3	120.0
AUTO - PILOT APPROACH DESPERSED CASE 4	127.0
AUTO - PILOT APPROACH DESPERSED CASE 5	122.0

** PROPELLANT NUMBERS ARE NOT A STATISTICAL MEAN BUT ARE REPRESENTATIVE OF THE NUMBER OF FLIGHT PROFILES FLOWN

Figure 6 illustrates a typical man-in-the-loop trajectory as contrasted with a profile generated by the automated pilot. Note that the trajectories are similar for the two cases..

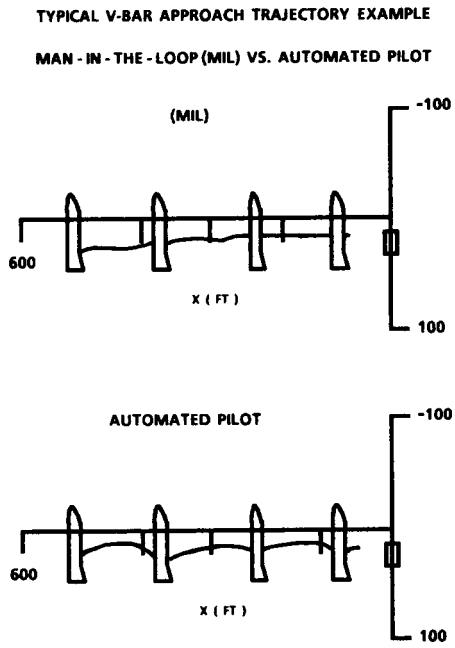


figure 6

STATUS AND CONCLUSIONS

The preliminary results of an automatic pilot for a low-fidelity Shuttle proximity operations simulator that maneuvers based on fuzzy decision functions indicate the goals of the study is achieveable. That is, it appears to be possible to simulate the common sense reasoning of a pilot using fuzzy decision functions to express rules obtained from experienced pilots.

A great deal of work must be completed to recognize the full potential of the concept, however. First, we realize improvements can be nade to the existing rules. These improvements will give more versatility to the reasoning and will incorporate additional fuzzy decision functions. An example is to give more emphasis to the vertical rate of the Shuttle to decide if the Shuttle needs to be raised or lowered. The present version does not deal with out-of-plane errors. In addition there are many more scenarios to address.

For expediency, the automatic pilot was implemented in Fortran, but it is apparent that a rule based expert system shell would provide a better implementation language and a conversion will be made as soon as possible.

While the parameters chosen for the curves used in the fuzzy functions result in appropriate decisions for a wide range of input for a given scenario, it may be desireable to allow the user to determine the piloting characteristics he would like simulated, and have the software select the parameters based on this description. For example, does the pilot respond quickly to deviations from the desired path, or does he allow errors larger than considered "average" to accumulate before he reacts. Results indicate a method could be devised to accomplish this.

As the approach is extended to other applications, or possibly to speed up use of the present application, it is realized that a fuzzy function chip could be used to offload a great deal of the computation. This would be especially appropriate to study the application of the concept to realtime rotational and translational digital autopilots.

ACKNOWLEDGEMENTS

The authors wish to acknowledge Sam Wilson, TRW Systems, and John Whynott, McDonnell Douglas Corporation, for their various contributions which range from dealing with the man-in-the-loop software simulation to augmenting our understanding of Shuttle proximity operations.

REFERENCES

1. Whynott, J.J., Hewlett Packard 9000 Desk Top Flight Simulator version 1.0 Preliminary Documentation, October, 1985.

2, Wilson, Sam, Delivery of Software for HP-9825A Desk Top Flight Simulator (DTFS) Version 04F, TRW Letter No. 84:W482.1-5, February, 1984.

3. Zadeh, L.A., Fu, K.S., Tanaka, K, Shimura, M. (eds.), Fuzzy Sets and Their Applications to Cognitive and Decision Processes, New York-London, 1975.

4. Zimmerman, H.J., Fuzzy Set Theory and its Applications, Kluwer-Nijhoff, Boston-Dordrecht-Lancaster Publishing 1985.

ARTIFICIAL INTELLIGENCE (AI), OPERATIONS RESEARCH (OR), AND DECISION SUPPORT SYSTEMS (DSS): A CONCEPTUAL FRAMEWORK

GREGORY S. PARNELL WILLIAM F. ROWELL JOHN R. VALUSEK

AIR FORCE INSTITUTE OF TECHNOLOGY
School of Engineering
Department of Operational Sciences
Wright-Patterson AFB, Ohio 45433-6583
(513) 255-2549

ABSTRACT

In recent years there has been increasing interest in applying the computer-based problem-solving techniques of Artificial Intelligence (AI), Operations Research (OR), and Decision Support Systems (DSS) to analyze extremely complex problems. The purpose of this paper is to develop a conceptual framework for successfully integrating these three techniques. First, the fields of AI, OR, and DSS are defined and the relationships among the three fields are explored. Next, a comprehensive adaptive design methodology for AI and OR modeling within the context of a DSS is described. The paper concludes with four major observations: (1) The solution of extremely complex knowledge problems with ill-defined, changing requirements can benefit greatly from the use of the adaptive design process, (2) the field of DSS provides the focus on the decision making process essential for tailoring solutions to these complex problems, (3) the characteristics of AI, OR, and DSS tools appears to be converging rapidly, (4) there is a growing need for an interdisciplinary AI/OR/DSS education.

INTRODUCTION

In recent years there has been increasing interest in applying the computer-based problem-solving techniques of Artificial Intelligence (AI), Operations Research (OR), and Decision Support Systems (DSS) to analyze extremely complex problems. The purpose of this paper is to develop a conceptual framework for successfully integrating these three techniques. First, the fields of AI, OR, and DSS are defined and the relationships among the three fields are explored. Next, a comprehensive adaptive design methodology for AI and OR modeling within the context of a DSS is described. Finally, the paper presents four major observations about the use of AI, OR, and DSS techniques to analyze the increasingly complex problems of the future.

AI, OR, AND DSS

This section briefly characterizes the fields of AI, OR, and DSS and examines their fundamental similarities and differences.

AI can be defined as "the study of ideas which enable computers to do things that make people seem intelligent." [1] AI means different things to different people. Natural language processing, robotics, and expert systems are the three major areas of AI. For the kinds of problems addressed in this paper expert systems (ES) are most applicable. An expert system is "an intelligent computer program that uses knowledge and inference procedures to solve problems that are difficult enough to require significant human expertise for their solution." [2]

Expert systems attempt to capture highly specialized human expertise in limited problem domains. Unlike conventional computer programs, ES separate the deduction mechanism (inference engine) from the knowledge base, which contains both the facts and rules. ES also provide a friendly user interface as well as the capability to explain their reasoning and recommendations. ES can be viewed as a special class of models which assist with a variety of tasks including interpretation, prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, and control. [3]

OR can be defined as "the application of the methods of science to complex

problems arising in the direction and management of large systems of men, machines, materials and money in industry, business, overnment, and defence. The distinctive approach is to develop a scientific model of the system, incorporating measurement of factors such as chance and risk, with which to predict and compare the outcomes of alternative decisions, strategies or controls." [4]

Operations research analysts traditionally use a wide range of mathematical models to help solve problems including mathematical programming, stochastic, simulation, and network models. For large problems the analyst traditionally works with a model providing input to the model and analyzing the resulting output.

DSS can be viewed as a evolutionary advancement beyond Electronic Data Processing (EDP) and Management Information Systems (MIS). EDP focuses on the generation, storage, processing, and flow of data at the operational level within the organization. MIS places its emphasis on the information flow of middle management. The key idea behind DSSs is their focus on supporting the decision process. The DSS builder views a DSS as consisting of three major components--a data base, a model base and a dialogue component which integrates the other two components and the user.

COMPARISON OF COMPUTER-BASED PROBLEM SOLVING APPROACHES

This section compares the relationship of the various computer-based problem solving approaches with the decision maker (summarized in Figure 1).

	DECISION-MAKER'S RELATIONSHIP				
	VALUES	FOCUS	SYSTEM INTERFACE	HUMAN INTERFACE	DEVELOPMENT APPROACH
ELECTRONIC DATA PROCESSING (EDP)	Clear	Efficiency	Limited	Programmer	Traditional
MANAGEMENT INFORMATION SYSTEMS (MIS)	Clear	Efficiency	- Information Summaries - Decision Queries	Systems	Traditional
OPERATIONS RESEARCH/ MANAGEMENT SCIENCE (OR/MS)	Credibly Model	Efficiency Insight	Summary Briefings	OR/MS Analyst	Modeling
DECISION SUPPORT SYSTEM (DSS)	Can Not Credibly Model	Effectiveness	Interactive - Information - Analysis Models	DSS Designer	Adaptive Design
EXPERT SYSTEM (ES)	Clear	Efficiency Effectiveness	Interactive - Recommendations - Explanations	Knowledge Engineer	Rapid Prototyping

Figure 1 Summary Comparison of Computer-based Problem Solving Approaches

Unlike the other approaches, DSS is usually applied to relatively unstructured or underspecified problems where it is not easy to directly model the values of the decision maker using an objective or value function. Instead, the decision maker's values are incorporated into the problem solution through the choices that the decision maker selects during operation of the DSS. Therefore, in the early stages of a DSS evolution, the system will likely take on a strong data base orientation.

EDP and MIS focus on efficiency, that is, accomplishing a specific task, such as processing a financial transaction, with a minimum amount of resources. Efficiency is an input-output measure. OR has a dual focus--allocating scarce resources efficiently and providing insight to the decision maker. DSS and ES provide the decision maker with new capabilities. The novice can use an ES to extend his capabilities. Experience has shown that because of the flexibility of a DSS a user often discovers that he can solve problems that he had never considered before or that could not be solved using other solution techniques.

The newer DSS and ES technologies allow the decision maker to interact directly with the system rather than relying on intermediaries such as a programmer or an operations research analyst. Particularly noteworthy is the capability of ES to make recommendations as well as furnish the decision maker with logical explanations to support these recommendations. This unique capability increases the credibility of the solutions generated.

The development approach taken by DSS and ES is significantly different from those of EDP, MIS, and OR. Both the adaptive design and rapid prototyping approaches involve initially selecting a small but significant problem. The design, development, and test phases are compressed into a few weeks and performed iteratively for a few months until a relatively stable system has emerged. Experience has shown that user requirements constantly change and, in reality, the system continues evolving until its retirement. The DSS adaptive design approach assumes that there exists an organizational commitment to field the system whereas it is not unusual to develop a 'throwaway' ES to demonstrate the feasibility of an ES technology.

ADAPTIVE DESIGN METHODOLOGY FOR AI AND OR MODELING IN DSS

This section describes our adaptive design methodology for AI and OR modeling in DSS. As mentioned earlier, a DSS has three components: the models, the data, and the dialogue (i.e., man machine interface). The three components can best be thought of as the three legs of a stool. Like the stool, the DSS can not withstand an ineffective leg.

The interrelationships of the three DSS components are shown in Figure 2. The role of OR/MS in DSS is well understood; however, the AI aspects require additional explanation. First, AI emphasizes new types of data (i.e., knowledge) and offers new knowledge representation approaches, e.g., semantic nets, frames, scripts, and rules. Second, AI offers new type of models, e.g., the cognitive models of human thought operationalized in an inference engine, for reasoning with the knowledge representation schemes. Third, AI shells and programming environments provide models for knowledge representation and MMI facilities.

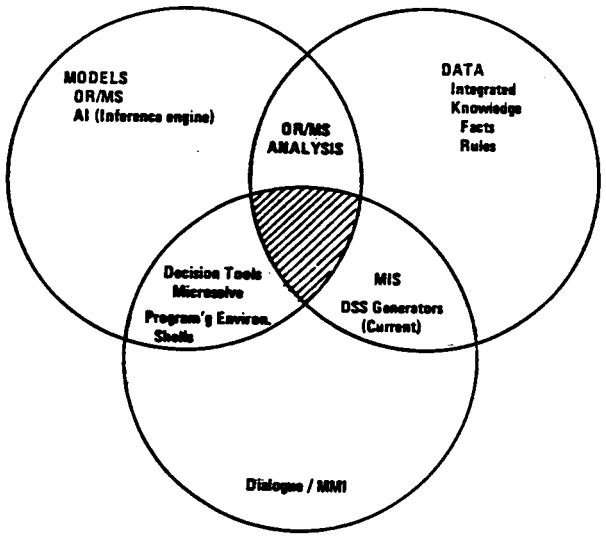


Figure 2. Decision Support System Components

Not all problems require a computer-based DSS using AI and OR techniques. Our experience has shown that the types of problem domains that require our approach (the cross-hatched area in the center of Figure 2) are complex, dynamic problem domains where specialized (procedural or heuristic) knowledge significantly improves the quality of recurring decisions. In these domains there may be many decision-makers.

The steps in our adaptive design methodology are summarized in Figure 3. These concepts have been developed and used by over 15 of our thesis students at AFIT over the past two years. We will focus on the first five steps, since these are unique to our approach. It is important to understand that our methodology is highly iterative; the results of any step may require redoing portions of one or more of the previous steps.

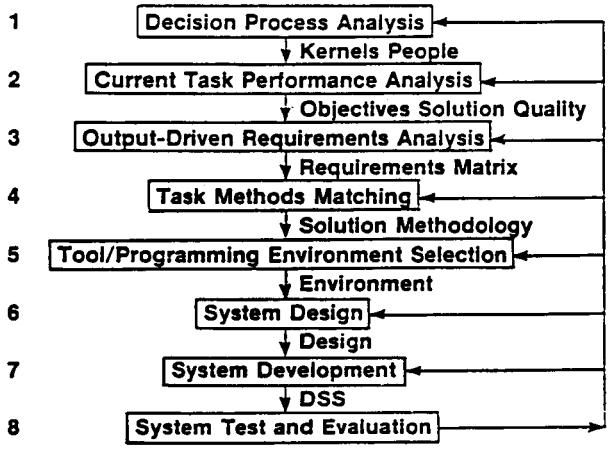


Figure 3. Adaptive Design Methodology for AI and OR Modeling in DSS

The first step in the process is the analyis of the decision process. This step is the most crucial in the entire methodology; fundamental errors in understanding of the decision process can easily result in solving the wrong problem. The most useful concepts have come from the DSS literature.

The major objective of the decision process analysis step is the identification of the kernel problems. We recommend focusing on the user's perspective and performing a technologically- unconstrained analysis of the decision process. A very useful technique is the concept map: a free-wheeling network, similar to a semantic net, that aids the analyst in capturing the major concepts and the cognitive processes of key decision-makers [5]. As an example, Figure 4 provides a concept map for determining the intent of an ICBM attack on the US [6]. Two additional components, the feature chart [7] and storyboarding [8], have also been used to capture the user's requirements using state-of-the-art graphics packages. This step concludes with the selection of the kernel problems in the decision process. The five kernals identified in Figure 4 are diplomatic, political, indications & warning, military, and economic.

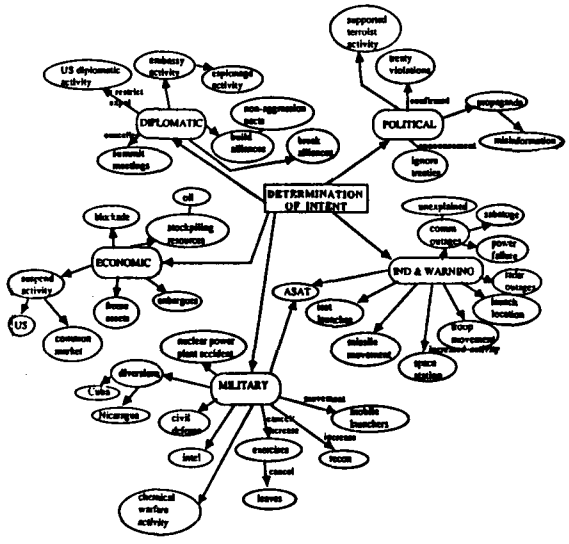


Figure 4. Concept Map

The second step in the methodology is the analysis of the current task performance for each of the kernel problems. We focus on the individuals involved, the objectives of each individual, and the desired quality of the solution. Figure 5 displays a matrix framework we have found useful. Tasks requiring an optimal solution suggest an OR model. Tasks where specialized knowledge is useful and a satisfactory solution is acceptable are candidates for AI techniques. Unstructured tasks with dynamic objectives are candidates for conventional data base query techniques. Finally, tasks with no feasible solutions are candidates for an OR analysis.

MAJOR TASK	EXPERT(S)		USER		MANAGER(S)		SENIOR DECISION MAKER	
	MIN COST		MIN COST		UNSPECIFIED		MIN COST	
TASK 1	S		F					
TASK 2	F		NF					
TASK 3					S		O	
TASK 4							NF	

O: optimal solution
S: satisfactory solution ⇒ specialized knowledge ⇒ AI
F: feasible solution ⇒ DSS candidate
NF: no feasible solution ⇒ OR analysis candidate

Figure 5. STEP 2: Current Task Performance Analysis

The third step in our adaptive design methodology is an output driven requirements analysis. The development of information systems requirements has been a major problem for MIS and DSS designers [9]. Users are unable to initially specify a complete set of the system requirements. Knowledge engineering focuses on capturing the knowledge of the experts but does not offer fundamentally new techniques for capturing system requirements. Like DSS, knowledge engineering makes extensive use of prototype knowledge systems to demonstrate the usefulness of AI in a problem domain. Our adaptive design approach synergistically combines AI and DSS concepts to use prototyping to capture the critical system requirements and provide a framework for the management of the adaptive design effort by focusing the prototype designer's efforts on the system requirements of the operational DSS.

Figure 6 provides our framework for recording DSS system requirements. This framework is used throughout the adaptive design effort. The current method column comes from the previous step. The second column identifies the requirements that the current prototype can successfully accomplish. The third column is only used when the goal of the design effort is to develop a prototype to establish the feasibility of an operational DSS. The fourth column identifies the desired requirements for an operational system. For the reasons discussed above, all four columns are iteratively developed during the adaptive design process.

	Current Method(s)	Prototype Status	Prototype Goal	Operational System
OUTPUTS Displays Graphics Knowledge Data Models				
Actions Recommendations Explanations				
PROCESSES Tasks				
INPUTS Static Inputs — DB — KB				
Dynamic Inputs				

Figure 6. STEP 3: Desired Output Driven Requirements Analysis

The left hand side of the matrix in Figure 6 identifies the types of knowledge or data the user would like displayed. We have found it useful to use storyboards to capture the users output requirements for each task. Many times we want our AI models to provide recommendations and explanations based on specialized knowledge in the DSS. The processes are the tasks analyzed in the previous step. In order for the processes to result in the desired outputs, static and dynamic inputs are required. The static inputs are the data and knowledge resident in the DSS. The dynamic inputs are provided interactively by the user or automatically by interfacing systems while the DSS is in use. Several AFIT thesis have successfully used this framework [10,11].

Step 4 in our methodology is the task/methods matching. Figure 7 provides an example of adaptive design step for a three task scheduling problem. Seven possible solution methodologies (paths) are identified. Three are pure paths: path 1 is knowledge engineering, path 3 is OR, and path 6 is data base browsing. The other four are mixed methodologies.

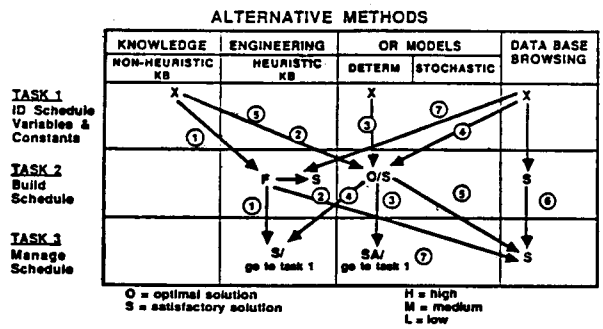


Figure 7. STEP 4: Task/Methods Matching

Step 5 is the analysis of the prototype tool/programming environments. Figure 8 provides a conceptual example for path 4 of the problem described in the previous paragraph. Three alternative approaches are identified. Again, each of these approaches can be evaluated against specific criteria and the selected approach can be implemented by the DSS designer. Two important trends are worth noting in this step. First, many AI tools increasingly allow the programmer to interface with database programs and conventional languages. Second, many conventional hardware and software vendors are seeking ways to incorporate AI programs.

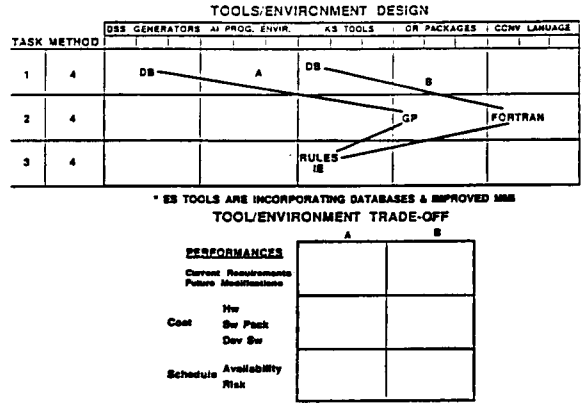


Figure 8. STEP 5: Prototype Tool/Programming Environment Selection

Once these steps are completed Step 6 (system design) and Step 7 (system development) proceed as in the mormal development of a conventional DSS. If AI techniques have been used, Step 8 (system test and evaluation) may require AI test and evaluation techniques, e.g., the cases method.

OBSERVATIONS

Finally, we make four observations about the use of AI, OR and DSS techniques to analyze complex decision problems. First, most complex, dynamic problem domains require the adaptive design process to capture the system requirements and demonstrate the usefulness of a computer-based decison aid. Second, DSS provides the most useful techniques for the crucial decision process analysis step in our adaptive design process. Third, AI/OR/DSS tools appear to be converging. Finally, the effective use of the adaptive design process described requires an interdisiplinary education in AI, OR, and DSS.

SUMMARY

This paper developed a conceptual framework for integrating AI, OR, and DSS techniques. First, the fields of AI, OR, and DSS were defined and compared. Next, a comprehensive adaptive design methodology for AI and OR modeling within the context of a DSS was described. Finally, the paper presented four major observations about the use of AI, OR, and DSS techniques to analyze the increasingly complex problems of the future.

REFERENCES

1. Headquarters United States Air Force Artificial Intelligence Research & Development Master Plan, Pentagon, Washington DC, April 30, 1984.

2. Harmon, Paul and King, David, Expert Systems: Artificial Intelligence in Business, Wiley Press, 1985, p. 5.

3. For a list of major applications see Donald A. Waterman, A Guide to Expert Systems, Addison-Wesley Publishing Company, 1986 and Bruce G. Buchanan, "Expert Systems: Working Systems and the Research Literature," Expert Systems, Vol. 3, No. 1, January 1986, pp. 37-51.

4. Operational Research Quarterly, Vol. 28, No. 3, i, 1977.

5. Seagle, John P., and Belardo, Salvatore, "The Feature Chart: A Tool for Communicating the Analysis of a Decision Support System", Information and Management, 1986, Vol 10, pp. 11-19.

6. Mc Farren, Michael, Using Concept Mapping to Define Problems and Identify Key Kernels During the Development of a Decision Support System, AFIT Masters Thesis, June 1987.

7. Bivins, Robert, Class Project in TACT 761, "Command and Control Decision Aiding", Winter Quarter, 1987.

8. Andriole, Stephen J., Akey, Mark, Dunkelberger, Kirk, and Millis, Phillip J., "Storyboarding for C2 Systems Design: A Combat Support System Case Study, Forthcoming paper to be presented at the 5th Annual Workshop on Command and Control Decision Aiding.

9. Valusek, John R., Information Requirements Determination: An Empirical Investigation of Obstacles within an Individual, Ph.D. Dissertation, University of Wisconsin-Madison, 1985.

10. Kennedy, Dale J., A Prototype Expert System Advisor for Satellite Support Scheduling, AFIT Masters Thesis, December, 1986.

11. Phillips, Barbara A., A Prototype Knowledge-Based System to Aid Space System Restoration Management, AFIT Masters Thesis, December, 1986.

N88-17246

ROBOTIC PLANNER EXPERT SYSTEM (RPLANES)

Ervin O'Neal Grice

Artificial Intelligence Section
Mission Planning And Analysis Division
Johnson Space Center
Houston, Texas 77058

ABSTRACT

The Artificial Intelligence Section of the Mission Planning and Analysis Division of the Johnson Space Center has developed a prototype of an expert system for robotic planning. A robot is given a high-level goal to perform an action (i.e. swap, adjust, or stow) on a component unit of an object such as a satellite and the Robotic Planner Expert System (RPLANES) generates the necessary goals for arm actions. RPLANES is designed using the Inference Corporation Automated Reasoning Tool (ART) development tool. It resides on a SYMBOLICS 3670. This paper describes RPLANES and its evolution.

1. INTRODUCTION

Robotic technology is an anticipated essential during the Space Station operational era. Robots will be given high-level goals to perform that will be accomplished by executing a series of lower-level goals. These lower-level goals and the order of performance are derived by a planner expert system. The initial development effort for the prototype detailed in this paper is attributed to the Sequence Automation Research Group of the Jet Propulsion Laboratory of the California Institute of Technology. The forty-eight rules expert system had a limited capability of planning the two arm actions of a robot to swap a component unit from a simple satellite structure (Figure 1) by dismantling all components of the structure. Enhancement of this expert system has evolved to eighty-two rules to:

a. Performing a swap, adjust, or stow goal.
b. Dismantling only those component units required to reach the target component.
c. Defining a more complex satellite structure (Figure 2).

2. INITIALIZATION

2.1 Define The Architectural Structure Of The Serviceable Object

Using the Inference Corporation Automated Reasoning Tool (ART) concept of schemata, the architectural structure of the serviceable object is defined. The relations "comes-before" and its inverse "comes-after" are defined and employed to define the architectural integrity of all components within the structure. Figure 2 is a diagram of the Laser Battlestation used in this prototype and Figure 3 details the definitions of the "door1" architecture using schemata.

2.2 Define The Serviceable Component Units

Each serviceable component unit of the object is defined with a collection of attribute slots (Figure 4). Specific actions that can be performed on the component and the action by the robot is defined. In addition to defining the component units of the object, replacement units used in a swap action are defined having a current location of a "tool-box" or "storage-bin" (Figure 5).

2.3 Define The Initial State Of The Robot

The initial state of the robot is defined using schemata and attribute slots (Figure 6). The left and right arms are statused "free" and the left and right arms are defined as empty by assigning tool values of "null-tool-1" and "null-tool-2" to each arm respectively.

2.4 Define The Action Scripts

An action on a component unit can be described as a sequence of sub-actions. For example a swap action of a component unit involves:

a. Rendezvousing with the component unit to be swapped.
b. Removing the component unit.
c. Stowing the component unit.
d. Unstowing the replacement component unit.
e. Replacing with the new component unit.

If a node is used to represent each sub-action, this sequence of nodes represents the action script. Each node in the script and its order within the script is defined using schemata and attribute slots (Figure 7).

3. USER INTERFACE

RPLANES employs mouse and menus for user interface (Figure 8). The serviceable object definition, type of action to perform, and the component unit(s) involved, are defined by the user. Any conflicts in these selections are determined by the expert system and redefinition is required.

4. GENERATING THE SUB-GOALS

4.1 Logically Dismantle The Required Component Units

Based upon the architectural structure of the object defined during initialization, all serviceable component units that "comes-before" the target component unit must be dismantled. A node representing each component unit along with attribute slots are placed in a knowledge base.

4.2 Retrieve The Action Script

Following the logical dismantling of the necessary component units, the specified action script is retrieved. A node and attribute slots representing the sequence of steps in the specified action script are placed in the knowledge base. A "comes-after" relationship attribute is associated with this set of nodes to denote the order of the action on the target component unit with respect to the dismantling of preceding units.

4.3 Logically Reassemble The Required Component Units

Once the action script has been logically performed on the target component unit, the object must be reassembled. Nodes representing each unit and the attribute slots detailing each node are also placed in the knowledge base. Reassembling the component units is performed in reverse with respect to dismantling.

4.4 Determine The Robot's Arm Actions

RPLANES assumes the robot has two arms that can work in parallel and/or serially. Progressing through the ordered nodes constructed in the knowledge base, the specific arm to be used by the robot, the tool required, and the component unit to service is determined.

5. OUTPUT

Using the nodes of attribute slots generated in the knowledge base above, the robot primitives (commands) are generated. The output is displayed to CRT (Figure 9).

6. CONCLUSION

Development of this prototype demonstrates the planner expert system functional capabilities. Applications employing graphics and hardware with the capability to input and execute these generated primitives are potential extensions.

7. ACKNOWLEDGEMENTS

I would like to thank Mark Rokey of the Sequence Automation Research Group of the Jet Propulsion Laboratory for the baseline concepts expanded within the expert system and Dr. Joseph Giarratano of the University of Houston-Clear Lake for the design of the laser battlestation used in the prototype expert system.

8. REFERENCE

1. Inference Corporation, "AUTOMATED REASONING TOOL REFERENCE MANUAL," 5300 W. Century Blvd., Los Angeles, Ca. 90045.

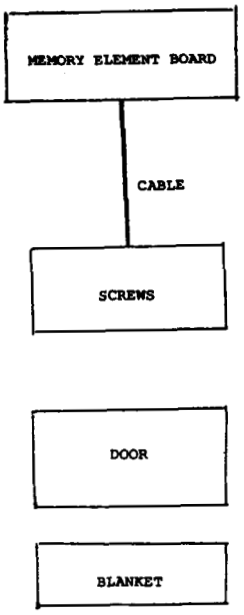


FIGURE - 1 SIMPLE SATELLITE

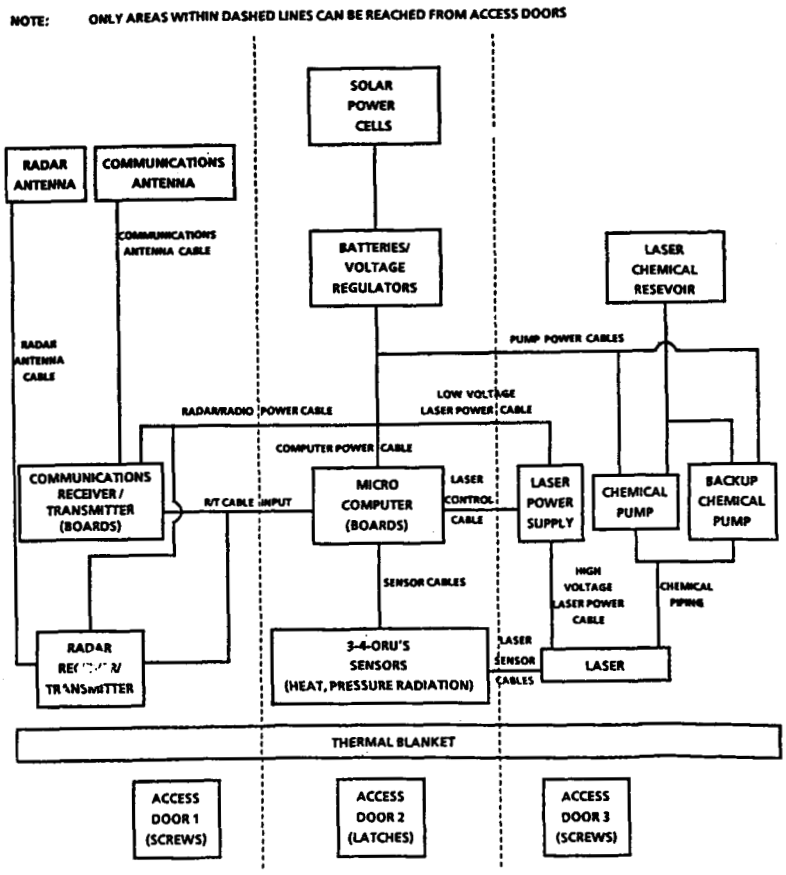


FIGURE - 2 LASER BATTLESTATION

```
;;;                                    DOOR1

(defschema laser-door1-part1 ""
  (structure-type laser-station)
  (part-type screw1)
  (comes-before laser-door1-part1)
  (instance-of part-item))

(defschema laser-door1-part1 ""
  (structure-type laser-station)
  (part-type screw2)
  (comes-before laser-door1-part1)
  (instance-of part-item))

(defschema laser-door1-part1 ""
  (structure-type laser-station)
  (part-type door1)
  (comes-before laser-door1-part2)
  (instance-of part-item))

(defschema laser-door1-part2 ""
  (structure-type laser-station)
  (part-type blanket)
  (comes-before laser-door1-part3)
  (comes-before laser-door1-part4)
  (comes-before laser-door1-part5)
  (instance-of part-item))

(defschema laser-door1-part3 ""
  (structure-type laser-station)
  (part-type radar-antenna-cable)
  (comes-before laser-door1-part4)
  (comes-before laser-door1-part6)
  (comes-before laser-door1-part7)
  (comes-before laser-door1-part8)
  (instance-of part-item))

(defschema laser-door1-part4 ""
  (structure-type laser-station)
  (part-type radar-power-cable)
  (comes-before laser-door1-part6)
  (comes-before laser-door1-part8)
  (comes-before laser-door1-part9)
  (instance-of part-item))

(defschema laser-door1-part5 ""
  (structure-type laser-station)
  (part-type radar-rt-cable)
  (comes-before laser-door1-part4)
  (comes-before laser-door1-part6)
  (comes-before laser-door1-part8)
  (instance-of part-item)).

(defschema laser-door1-part6 ""
  (structure-type laser-station)
  (part-type radar-receiver-transmitter)
  (comes-before laser-door1-part8)
  (comes-after laser-door1-part2)
  (instance-of part-item))

(defschema laser-door1-part7 ""
  (structure-type laser-station)
  (part-type radar-antenna)
  (comes-after laser-door1-part6)
  (comes-after laser-door1-part8)
  (instance-of part-item))

(defschema laser-door1-part8 ""
  (structure-type laser-station)
  (part-type comm-receiver-transmitter)
  (comes-after laser-door1-part6)
  (instance-of part-item))

(defschema laser-door1-part9 ""
  (structure-type laser-station)
  (part-type comm-antenna-cable)
  (comes-before laser-door1-part8)
  (comes-before laser-door1-part10)
  (instance-of part-item))

(defschema laser-door1-part10 ""
  (structure-type laser-station)
  (part-type comm-antenna)
  (comes-after laser-door1-part8)
  (instance-of part-item))
```

FIGURE - 3 LASER BATTLESTATION SCHEMATA DEFINITION

```
(defschema door1 ""
  (family door)
  (first-choice-arm right-arm)
  (second-choice-arm left-arm)
  (status attached)
  (current-location laser-station)
  (actions-by-this-object-slot (slot-how-many multiple-values)
              (detach open)
              (attach close)
              (adjust-in close)
              (adjust-out open)
              (stow stow)
              (unstow unstow)
              (replace attach)
              (remove detach))
  (state-or-location-after-action-slot (slot-how-many multiple-values)
               (detach laser-station)
               (attach laser-station)
               (adjust-in closed)
               (adjust-out opened)
               (stow rack)
               (unstow freespace)
               (replace laser-station)
               (remove rack))
  (tool (slot-how-many multiple-values) null-tool-1 null-tool-2))

(defschema screw-characteristics ""
  (family screw)
  (actions-by-this-object-slot (slot-how-many multiple-values)
              (detach unscrew)
              (remove unscrew)
              (adjust-in screw)
              (adjust-out unscrew)
              (stow slide-in)
              (unstow slide-out)
              (attach screw)
              (replace screw))
  (state-or-location-after-action-slot (slot-how-many multiple-values)
               (detach outset)
               (remove toolbox)
               (adjust-in inset)
               (adjust-out outset)
               (stow tool-box)
               (unstow freespace)
               (attach inset)
               (replace inset))
  (tool screwdriver))

(defschema screw-1 ""
  (first-choice-arm right-arm)
  (second-choice-arm left-arm)
  (instance-of screw-characteristics)
  (current-location inset)
  (status attached))
```

FIGURE - 4 SERVICEABLE COMPONENT DEFINITION

```
(defschema new-door ""
  (family door)
  (first-choice-arm right-arm)
  (second-choice-arm left-arm)
  (status attached)
  (current-location rack)
  (actions-by-this-object-slot (slot-how-many multiple-values)
              (replace attach)
              (adjust-in push)
              (adjust-out pull)
              (stow stow)
              (unstow unstow)
              (remove detach))
  (state-or-location-after-action-slot (slot-how-many multiple-values)
               (replace satellite)
               (adjust-in rack)
               (adjust-out rack)
               (stow rack)
               (unstow freespace)
               (remove satellite))
  (tool (slot-how-many multiple-values) null-tool-1 null-tool-2))
```

```
(defschema new-cable ""
  (family cable)
  (first-choice-arm left-arm)
  (second-choice-arm right-arm)
  (status attached)
  (current-location toolbox)
  (actions-by-this-object-slot (slot-how-many multiple-values)
              (replace plug-in)
              (remove slide-out)
              (stow fold)
              (unstow slide-out))
  (state-or-location-after-action-slot (slot-how-many multiple-values)
               (replace satellite)
               (remove satellite)
               (stow tool-box)
               (unstow freespace))
  (tool (slot-how-many multiple-values) null-tool-1 null-tool-2))
```

FIGURE - 5 REPLACEMENT UNITS DEFINITION

```
(defschema script "Base definition of a script schema"
  (script-type)
  (script-object)
  (script-state)
  (comes-before))

;;;*****************************************************************
;;;  Here is the definition of the network of nodes which describe
;;;  a SWAP action.  This is typical of what will be many such
;;;  "script-like ACTION networks" in the final system.
;;;*****************************************************************

(defschema swap-source ""
  (instance-of script)
  (script-type swap)
  (script-object source)
  (script-state rendezvous)
  (comes-before swap-1))

(defschema swap-1 ""
  (instance-of script)
  (script-object swappee)
  (script-type swap)
  (script-state remove)
  (comes-before swap-2))

(defschema swap-2 ""
  (instance-of script)
  (script-object stowee)
  (script-type swap)
  (script-state stow)
  (comes-before swap-3))

(defschema swap-3 ""
  (instance-of script)
  (script-object unstower)
  (script-type swap)
  (script-state unstow)
  (comes-before swap-4))

(defschema swap-4 ""
  (instance-of script)
  (script-object swapper)
  (script-type swap)
  (phase 3mark)
  (script-state replace)
  (comes-before swap-sink))

(defschema swap-sink ""
  (instance-of script)
  (script-type swap)
  (script-state rendezvous)
  (script-object sink))
```

FIGURE - 7 ACTION SCRIPT DEFINITION

```
(defschema left-arm ""
  (status free))

(defschema right-arm ""
  (status free))

(defschema screwdriver ""
  (status free))

(defschema null-tool-1 ""
  (held-by right-arm)
  (status free))

(defschema null-tool-2 ""
  (held-by left-arm)
  (status free))
```

FIGURE - 6 INITIAL ROBOT STATE

```
Primitives developed for robot

(TASK REQUESTED: STOW THE LASER-STATION RADAR-RT-CABLE)

(MOVE RIGHT-ARM TO TOOLBOX)
(REPLACE NULL-TOOL-1)
(ACQUIRE SCREWDRIVER)
(MOVE RIGHT-ARM TO SCREW1)
(UNSCREW SCREW1)
(MOVE RIGHT-ARM TO TOOL-BOX)
(SLIDE-IN SCREW1 IN TOOL-BOX)
(MOVE RIGHT-ARM TO SCREW2)
(UNSCREW SCREW2)
(MOVE RIGHT-ARM TO TOOL-BOX)
(SLIDE-IN SCREW2 IN TOOL-BOX)
(MOVE RIGHT-ARM TO TOOLBOX)
(REPLACE SCREWDRIVER)
(ACQUIRE NULL-TOOL-1)
(MOVE RIGHT-ARM TO DOOR1)
(OPEN DOOR1)
```

```
Primitives developed for robot

(TASK REQUESTED: STOW THE LASER-STATION RADAR-RT-CABLE)  page 2

(MOVE RIGHT-ARM TO BLANKET)
(REMOVE BLANKET)
(MOVE RIGHT-ARM TO TOOL-BOX)
(SLIDE-IN BLANKET IN TOOL-BOX)
(MOVE LEFT-ARM TO RADAR-RECEIVER-TRANSMITTER RADAR-RT-CABLE CONNECTION)
(UNPLUG RADAR-RT-CABLE)
(MOVE LEFT-ARM TO COMM-RECEIVER-TRANSMITTER RADAR-RT-CABLE CONNECTION)
(UNPLUG RADAR-RT-CABLE)
(MOVE LEFT-ARM TO RADAR-RT-CABLE)
(DETACH RADAR-RT-CABLE)
(MOVE LEFT-ARM TO TOOL-BOX)
(FOLD RADAR-RT-CABLE)
(MOVE RIGHT-ARM TO TOOL-BOX)
(SLIDE-OUT BLANKET)
(MOVE RIGHT-ARM TO SATELLITE)
(REPLACE BLANKET TO SATELLITE)
```

```
Primitives developed for robot

(TASK REQUESTED: STOW THE LASER-STATION RADAR-RT-CABLE)  page 3

(MOVE RIGHT-ARM TO DOOR1)
(CLOSE DOOR1)
(MOVE RIGHT-ARM TO TOOLBOX)
(REPLACE NULL-TOOL-1)
(ACQUIRE SCREWDRIVER)
(MOVE RIGHT-ARM TO TOOL-BOX)
(SLIDE-OUT SCREW1)
(MOVE RIGHT-ARM TO INSET)
(SCREW SCREW1 TO INSET)
(MOVE RIGHT-ARM TO TOOL-BOX)
(SLIDE-OUT SCREW2)
(MOVE RIGHT-ARM TO INSET)
(SCREW SCREW2 TO INSET)
```

FIGURE - 9 SAMPLE OUTPUT

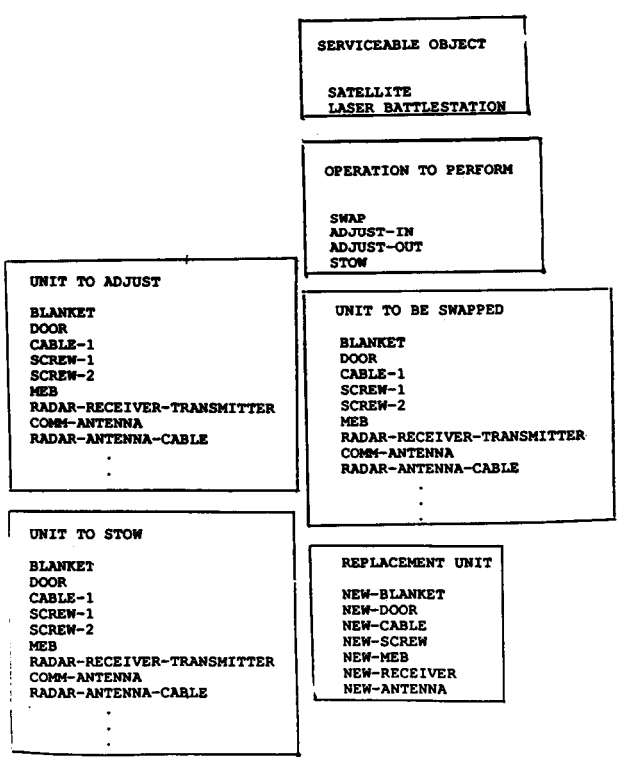


FIGURE - 8 USER INTERFACE MENUS

EXPERT MISSION PLANNING AND REPLANNING SCHEDULING SYSTEM FOR NASA KSC PAYLOAD OPERATIONS

Roger Pierce
Payload Operations
Information Systems Office (CS-ISO)
National Aeronautics and Space Administration
Kennedy Space Center, Florida 32899

ABSTRACT

EMPRESS (Expert Mission Planning and REplanning Scheduling System) is an expert system created to assist payload mission planners at the Kennedy Space Center (KSC) in the long range planning and scheduling of horizontal payloads for space shuttle flights. Using the current flight manifest, these planners develop mission and payload schedules detailing all processing to be performed in the Operations and Checkout (O&C) building at KSC. With the EMPRESS system, schedules are generated quickly using standard flows that represent the tasks and resources required to process a specific horizontal carrier. Resources can be tracked and resource conflicts can be determined and resolved interactively. Constraint relationships between tasks are maintained and can be enforced when a task is moved or rescheduled. EMPRESS became operational in March 1986. It was developed jointly by NASA at the Kennedy Space Center and by the MITRE Corporation of Bedford, Massachusetts. EMPRESS-II, currently under development by the MITRE Corporation, is scheduled to be delivered to KSC in September 1987. This paper will briefly descibe the domain, structure, and functionality of the EMPRESS system. It will describe some of the limitations of the EMPRESS system as well as improvements expected with the EMPRESS-II development. Finally, the future of the project will be discussed.

INTRODUCTION

As the primary launch and landing site of the Space Transportation System (STS), the Kennedy Space Center (KSC) is responsible for the final checkout, preparation, and installation of payloads into the space shuttle orbiter vehicle prior to a mission. Upon return from space, KSC is responsible for the deintegration of these payloads. A payload is an experiment or set of experiments attached to a carrier structure, which is then placed into the shuttle payload bay. A mission can be thought of as a set of payloads that are flown into lower Earth orbit using the STS.

Payloads and payload operations at KSC are divided into two primary categories, vertical and horizontal. Vertical payloads are installed into the orbiter vehicle at the launch pad and include payloads that use the Payload Assist Module (PAM) or the Inertial Upper Stage (IUS) as their carrier structure. Most satellites launched from the space shuttle are considered vertical payloads. Horizontal payloads, on the other hand, are usually installed into the orbiter payload bay at the Orbiter Processing Facility (OPF). Carrier structures for these payloads include the Spacelab long module (LM), short module (SM), and pallet. The mission peculiar experiment support structure (MPESS) is also considered a horizontal payload carrier. Various payload carrier combinations may be flown on the same mission which increases the complexity of the work performed.

Processing of horizontal payloads occurs primarily at the Operations and Checkout building (O&C) in the KSC industrial area. This processing includes all of the steps or tasks necessary to assemble and install the experiments onto the carrier structure as well as the steps needed to perform the required experiment and subsystem functional verifications prior to transport of the payload to the OPF and installation into the orbiter. To moniter and control this processing activity, NASA generates and maintains a hierarchy of schedules illustrated in Figure 1.

At the top of the NASA schedule hierarchy is the flight manifest. Released periodically from NASA Headquarters in Washington D.C., this document assigns launch dates, orbiter vehicles, and payloads to each STS

mission and lists this information for missions slated for the next five years. An example of the manifest in its telemail format is found in Figure 2. With this data KSC then creates long range plans and schedules that describe how KSC will support the launch date milestones. In the payload processing community, one of the long range support plans is called the Multiflow Assessment or MFA. This schedule was previously known as the Master Multiflow or MMF.

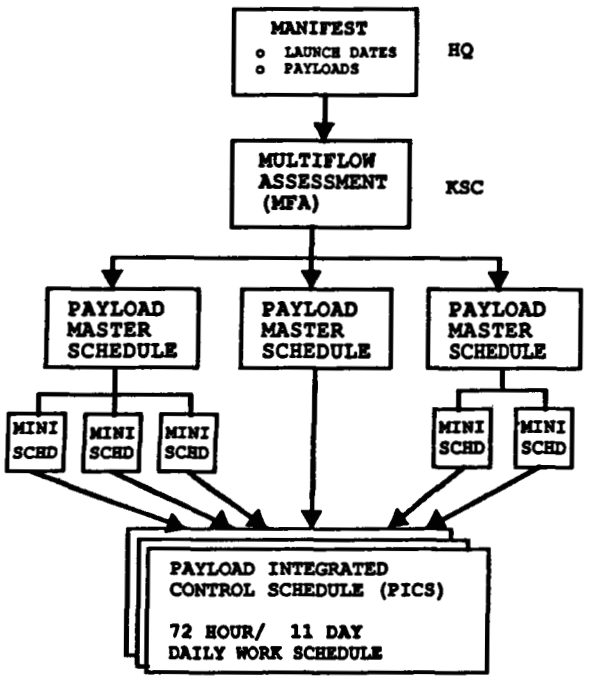


Figure 1 - KSC Schedule Hierarchy

The MFA consists of Gantt charts that illustrate the major processing activities needed for each payload listed in the manifest. More detailed information is given for payloads processed by the KSC Payload Directorate. For horizontal payloads, the major activities include experiment integration, carrier integration, and orbiter integration operations. These activities are referred to as Level IV, Level III/II, and Level I, respectively. An example of the MFA format is shown in Figure 3. In addition, the MFA contains information on some of the critical resource needs of these payloads. This enables early recognition of potential conflicts between limited resources. The MFA also serves as a baseline for the development of more detailed mission and payload schedules.

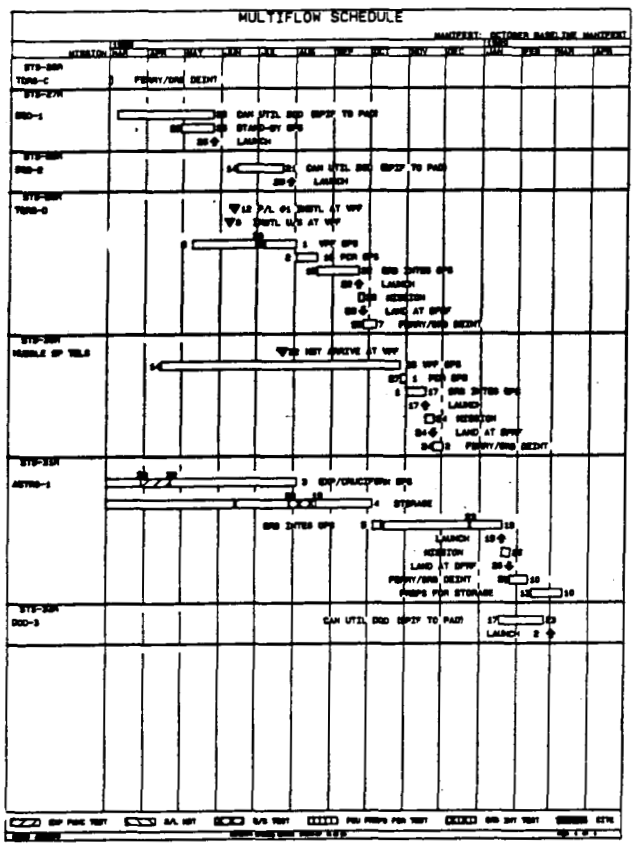
Because of the dynamic nature of shuttle operations, the need to respond quickly and effectively to changes in the process-

*** SHUTTLE FLIGHT ASSIGNMENTS FOR PAYLOADS ***
JUNE 1987 PRELIMINARY BASELINE
POP 87-2 MAX DOD OFFLOAD CP-APR 8-JUN-87

FLT	DATE ORBTR	INCL ALT	CRW DUR	PAYLOAD	CARRIER	OTHER PAYLOADS	UF
34	89 10 5 COLUMBIA	28.5 160	5 4	GPS-1 GPS-2 MSL-3	PAM-D2 PAM-D2 MPESS		
35	89 10 23 DISCOVERY	28.5 110	5 4	GALILEO	IUS-3 STA		
36	90 1 11 ATLANTIS	0.0 0	0 0	DOD			
37	90 3 1 COLUMBIA	28.5 160	5 5	GPS-3 GPS-4 MSL-4	PAM-D2 PAM-D2 MPESS		
38	90 4 9 DISCOVERY	0.0 0	0 0	STARLAB	LM+1P		
39	90 4 30 ATLANTIS	0.0 0	0 0	DOD			
40	90 6 7 COLUMBIA	28.5 160	5 4	GRO			
41	90 7 12 DISCOVERY	28.5 160	5 4	TDRS-E	IUS-2 STA		

Figure 2 - NASA Manifest in Telemail Format

ing schedule is important. Payload mission planners are often called upon to develop new MFA's quickly when the manifest is changed or to produce "what-if" schedules when examining unusual mission scenarios. This planning and replanning



7

Figure 3 - KSC Multiflow Assessment example using Artemis.

must be as accurate and timely as possible. Unfortunately, the manual methods for developing and producing these schedules do not fulfill these requirements.

In an effort to automate the process of producing the MFA, NASA at the Kennedy Space Center initiated a project to perform this function using artificial intelligence (AI) techniques. The project was named EMPRESS (Expert Mission Planning and REplanning Scheduling System) and was developed jointly between NASA at KSC and the MITRE Corporation out of Bedford, Massachusetts. The goal of the project was twofold. First, EMPRESS was to demonstrate the application of AI to a real KSC problem, namely planning and scheduling. Second, the project was to be instrumental in building a NASA in-house AI capability for payload operations. Both of these objectives have been met with the current EMPRESS system.

HISTORY

The initial study for EMPRESS was begun by NASA and the MITRE Corporation in May of 1984. Dual implementation began in January of 1985 and the prototype was completed in February 1986. EMPRESS became operational in March of 1986. The software was developed on Symbolics 3600 series LISP machines using Symbolics' 6.1 operating system. EMPRESS was written in ZetaLISP using the Symbolics' object oriented programming paradigm called Flavors. A Flavor is a data structure used for symbolic representation of an object. In November 1986, EMPRESS was updated by NASA to the new Symbolics Genera 7.0 operating system.

EMPRESS is currently used by the KSC payload operations community as an aid in developing quick "what-if" mission scenarios and resource utilization studies. EMPRESS is not used to produce the MFA. This is due in part to a change in the responsibility for the MFA and in limitations of the EMPRESS prototype. In January 1987, responsibility for the MFA was transferred from NASA to the Payloads Ground Operations Contract (PGOC), which was awarded to the McDonald Douglas Aerospace Corporation (MDAC). MDAC uses the commercial Artemis scheduling system supplied by Metier to produce the MFA.

DESCRIPTION

EMPRESS was designed to allow a payload mission planner the ability to develop schedules quickly for horizontal payloads using the space shuttle flight manifest. When creating a schedule for a payload on a mission, EMPRESS first looks to see if a current schedule already exists for that particular mission or payload. If not, EMPRESS creates a default schedule using a standard flow, which is simply a list of all of the tasks, task constraints and resources required to process a particular horizontal carrier. With the default schedule generated, the planner can then modify the tasks and resources as required. EMPRESS gives the planner the ability to verify that resource conflicts have not occurred between parallel operations and to revise resources and tasks automatically if conflicts do exist. Constraint relationships between tasks are maintained and can be enforced when tasks are moved or rescheduled. The user interface is quite robust and gives the planner an excellent graphical representation of the schedule and detailed histograms of the resource utilization.

DOMAIN

The domain knowledge base for EMPRESS can be divided into three major areas - tasks, resources, and system heuristics. Task data include the various activities required to process a payload and the temporal relationships between these tasks. The resource knowledge encompasses the people, hardware, and facilities required to process a payload. The heuristics control scheduling, resource allocation, and conflict resolution.

All tasks in EMPRESS are represented as Flavor objects. Each task may have a set of subtasks and each subtask may have subtasks resulting in an overall tree structure for the task knowledge base. At the top of the EMPRESS task tree is the manifest. A manifest may have any number of mission subtasks and each mission may have any number of payload subtasks. Each mission task has at least a launch date milestone and a fly-mission task associated with it. In addition, each payload task has a series of processing tasks. These processing tasks are divided into the various integration activities, which include the Level IV, Level III/II, and Level I functions. The integration steps are finally reduced into the lowest task level which may include activities like experiment staging, integration, power-up, or testing.

Relationships between tasks are also maintained by the EMPRESS domain. This knowledge is represented by variables in the task Flavor structure. Using these variables, constraints such as task-subtask or parent-child relationships, as

well as predecessor-successor or sibling relationships can be defined. EMPRESS refers to these relationships by the terms "contains", "contained-by", "before", or "after". A task may also have a set of milestones associated with it. Milestones are separate data structures in EMPRESS and signify one time events with no duration. Launch dates, as an example, are considered milestones. The relationships between a task and a milestone are referred to by "begins", "begun-by", "ends", or "ended-by". Using these eight task constraint mechanisms, EMPRESS has a powerful method for defining any task hierarchy.

The resource domain defines the facilities, hardware, and people available to process a payload. Table 1 outlines the resource class structure used in the development of EMPRESS. Like tasks, resources are stored as flavor objects in the working memory of the system. However, where task hierarchial relationships are maintained by variables, the resource relationships are maintained by the inheritence mechanism provided by the flavor paradigm. For example, a PCU is a type of Test Equipment, which is a type of Facility, which is an essential resource. Resource data includes the maximum number of individuals available, any possible alternative resources, and a list of the current users.

Along with the task and resource knowledge, the domain embodies the heuristics used to control the planning and scheduling. This includes the methods for scheduling the tasks, searching for existing schedules, assigning resource states, and maintaining task constraints. EMPRESS has a small set of rules used to resolve resource conflicts. These rules control how EMPRESS finds alternative resources, modifies resource utilization, or reschedules the resources within a task to correct a resource problem.

STRUCTURE

The EMPRESS architecture is divided into six primary modules. These modules are the Command Module, the Display Module, the Scheduling Module, the Resource Module or Resource Tracker, the Constraint Module, and the Data Module. Figure 4 illustrates this software structure. Each module performs a specific series of functions and the interaction between the modules provides for a flexible and powerful planning tool.

The Command Module contains most of the higher level system functions. It controls the system build and holds many of the system utility functions and application programs.

The functions for the user interface are performed by the Display Module. This includes the window processes, menus, mouse functions, and the graphics code seen by the user, including the calendar and histogram displays. The EMPRESS command loop and file system interface codes are also found here.

The Scheduler is responsible for building a schedule and maintaining the constraint relationships between tasks. The Scheduler pulls task information from the Data Module, calculates start and end times, and verifies that no temporal constraints have been violated. This module also defines the task flavor structure and controls many of the query and modification functions that can be performed on a task.

With a preliminary schedule created, the Resource Module allocates resources and maintains resource accountability. The resource flavor structure and class hierarchy are stored here. When a user decides to commit a resource to a task,

TABLE 1 - EMPRESS Resource Class Summary

Facility	Non-Flight Hardware
GSE	Alignment Equipment
EGSE	Brackets
MGSE	Cable
Offline Area	Cable Harness
Storage Area	Canister
Test Equipment	Covering
ATE	Crane
CITE	Fork Lift
HITS	Harness
HRDE	Long Trolley
PCU	Strong Back
PITS	Subsystem Hardware
PSTE	Support Structure
SPCDS	Transporter
Test Stand	Trunnion Support Fix
User Room	

Flight-Hardware	People
Carrier	Electrical Engineer
Experiment	Logistics
Flight Payload	Mechanical Engineer
Floor	Quality Control
GSA	Safety
IPS	Technician
Keel	Test Conductor
MPE	
Orbiter	
Rack	
Trunnion	

the Resource Tracker verifies the resource availability and modifies the resource usage if appropriate. If the resource is unavailable, the resource tracker contains the rules needed to resolve the resource conflict.

The Constraint Module defines the temporal relationships between tasks. It also has the capability to create precedence relationships between generic tasks. However, this capability has not been implemented.

Finally, the Data Module stores the majority of information used by the system. This includes the manifest, mission, payload, and standard flow files, as well as the resource and task knowledgebases. Any schedules saved by the user and the system files required to initialize EMPRESS are found here.

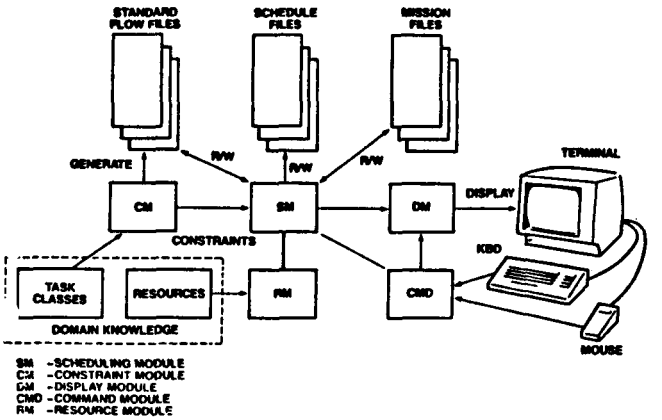


Figure 4 - EMPRESS Structure Summary

FUNCTIONALITY

EMPRESS contains a multitude of functions for creating a schedule. The best way to summarize these functions is to review a hypothetical session with an EMPRESS user when given a new flight manifest. In this example session, we will create a schedule from a manifest, describe some of the interface display options, explain resource handling, review conflict resolution, and finish by describing some useful tools. All functions in the EMPRESS system are menu or mouse driven.

When given a new manifest, usually in a telemail text format, the text file is first converted to a usable form by running an EMPRESS utility function. The operator then selects a "load manifest" menu option to begin the task of creating a new schedule. As each mission and payload on the manifest is read, EMPRESS searches to see if an existing schedule already exists. If so, then that schedule is used. Otherwise, EMPRESS creates a default schedule for the payload based on its carrier. These default carrier schedules are called standard flows and they contain the tasks, task constraints, and resources needed to process that type of carrier. In EMPRESS there are standard flows for the following horizontal carriers - Spacelab long module, pallet-igloo, MDM-pallet, and MPESS. For missions that do not contain horizontal carriers, EMPRESS creates a default fly-mission task using the manifest launch date.

After loading in each payload schedule, the payload tasks are scheduled to determine start and end times and to verify that no task constraints have been violated. This is done by making a backward CPM pass with the launch date as the starting point. The total process of converting a new manifest from the telemail format to creating a schedule for 50 missions takes less than 20 minutes. An example of the loaded manifest is given in Figure 5.

With schedules for all of the manifested missions loaded, the operator has many display options. He may keep the display in the manifest format or view an individual mission. The spreadsheet-like display can be moved in any direction and the calendar end times may be changed as required. In the "tree" function, the duration of the calendar is tied to a specific task or subtask and can display the activities on a weekly or daily level. The operator can also "open" or "close" any task to show any level of the task-subtask hierarchy. This is shown in Figure 6 for the planned STS-31 mission. If a task needs to be relocated, the operator may move the task graphically with the mouse, then reschedule the task. Resources are reallocated automatically and task constraints verified.

Using the standard flows, resource needs are identified for various tasks in the processing task tree. EMPRESS resources may be in one of three states: unspecified, requested, or committed. The operator can change the status of a task's resource needs to "requested" or "committed" and view the resource status in a histogram or utilization chart. A resource histogram is illustrated in Figure 7. When the resources are committed, EMPRESS will detect resource conflicts.

There are several ways to change a resource state. The operator has the option to request or commit all resources for all missions or to set the resource state at any level of the task tree. In addition,

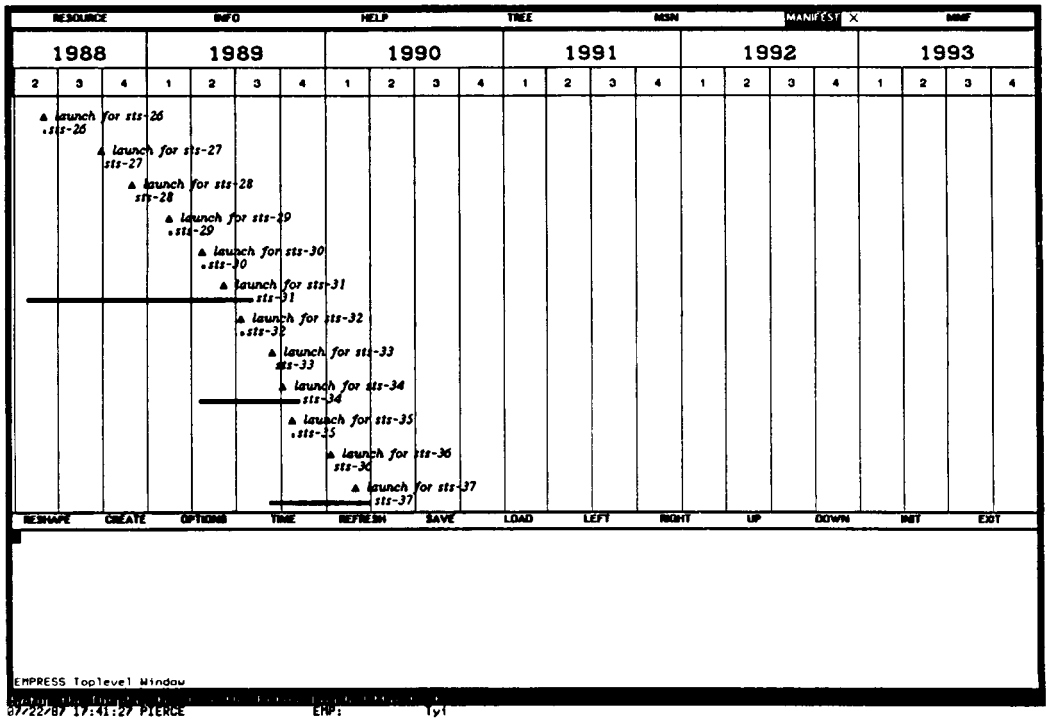


Figure 5 - Example of a flight manifest schedule using EMPRESS.

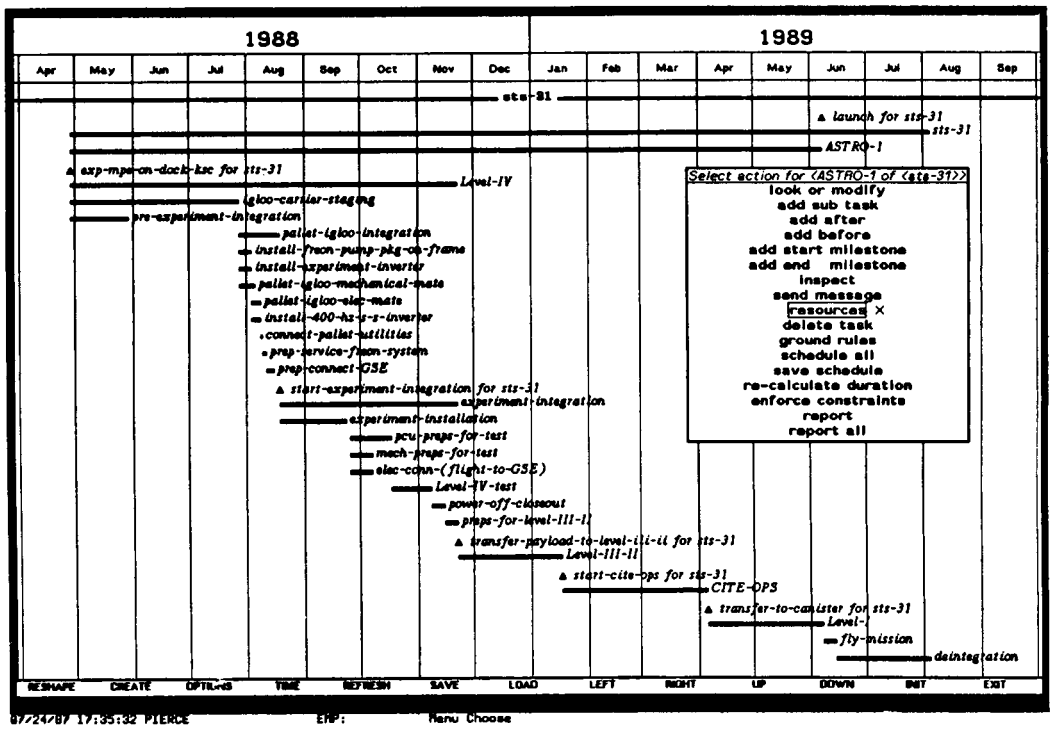


Figure 6 - Example of a task-subtask tree and task menu using EMPRESS.

the operator may add or delete resources for any individual task in the schedule.

With an EMPRESS schedule created and resources committed, the operator can now detect and resolve any conflicts that may have arisen. Conflicts occur during task scheduling or resource allocation. If a task constraint is violated (ie. a child task is moved outside of its parent task), the operator has two options. The constraint can be enforced and the child task will be repositioned under the parent task, or the constraint can be relaxed and the parent task's end times will be recalculated to remove the contention. In a resource allocation conflict, a set of rules are fired that allow the operator to resolve the conflict. These rules allow the operator to substitute an alternative resource, to increase the workload of the resource (ie. add more shifts), or to reschedule the task that caused the problem. The operator may also choose to let EMPRESS resolve all resource conflicts automatically without operator input.

After all conflicts are resolved, the operator can save the schedule into the EMPRESS data directory or into his or her personal directory. This feature allows each user to create and store individual "what-if" files that can be recalled and revised later. With the schedule saved, the operator can then reinitialize the system and start anew.

There are a set of optional functions that further enhance the EMPRESS planning and scheduling capability. A "create" option allows the operator to create a "what-if" mission that does not exist on the flight manifest. If the user is interested in working on only a small subset of the manifest, there are "mark" functions that will reduce the mission set. There is also a complete set of query functions for reviewing resource accountability and the system knowledgebase.

LIMITATIONS

While EMPRESS provides an effective tool for payload mission planning, it has limitations. The primary limitation with the system is the lack of output. When a schedule is completed, the operator must use screen prints to hardcopy the display. For lengthy schedules, this is quite bothersome and ineffective. EMPRESS does not match many of the Artemis graphics capabilities used in the current MFA. These limitations must be corrected in order for EMPRESS to produce this document.

There are other limitations with the current EMPRESS system. EMPRESS does not store its data in a relational format and can not access a relational database. There is no justification mechanism to explain scheduling, rule firing or conflict resolution. There are also small problems with the current implementations for deintegration tasks, standard flow flavor structures, and the scheduler.

EMPRESS II

After completion of the original EMPRESS prototype in February 1986, the MITRE Corporation continued development on a new project. This project, called EMPRESS-II, uses a different approach to the payload planning and scheduling problem. EMPRESS-II is built upon a new planning and scheduling architecture developed by MITRE for the Air Force called CAMPS (Core of A Meta Planning System). CAMPS provides a more robust foundation for planning and scheduling than the original EMPRESS system and addresses many of its limitations. CAMPS provides for a full declarative representation of the knowledge base. It supports external relational databases, has improved scheduler and resource tracking capabilities and implements an effective justification and truth maintenance system. CAMPS will use strategies and agendas to facilitate automatic planning and replanning operations.

A pre-release version of EMPRESS-II was delivered to KSC in December 1986. The current development project concludes in September 1987.

THE FUTURE

Despite the limitations of EMPRESS, the future of the project looks bright. Work is about to begin on the development of a new version of the EMPRESS system. This work will be performed by NASA using the KSC Payload Operations Artificial Intelligence Application Laboratory with assistance from its PGOC contractor. A user group is being started and a system for configuration control will be implemented. The redevelopment of the EMPRESS system will focus on creating graphical output, improving the user interface and scheduler, and in enhancing the conflict resolution and justification capabilities of the system. The new EMPRESS will also access schedules from the Artemis database currently in use. With a concerted effort, KSC's goal to implement an operational AI system for payload planning and scheduling will be achieved.

CONCLUSIONS

EMPRESS has provided KSC with an exceptional prototype planning and scheduling tool. Using artificial intelligence techniques, schedules for horizontal payloads are created quickly and contentions for limited resources can be determined and resolved interactively. Development of a new version will address many of the limitations with the initial system and bring the project to a more operational state.

REFERENCES

1. Dumoulin, J.M., "EMPRESS Expert Mission Planning and Replanning Scheduling System", software design presentation, NASA, Kennedy Space Center, Florida, May, 1985.

2. Dumoulin, J.M., "Payload Operations and Management AI/Expert Systems Projects Status", management briefing, NASA, Kennedy Space Center, Florida, June, 1986.

3. Dumoulin, J.M., "Payload Operations Artificial Intelligence Lab Activities and Facilities", management briefing, NASA, Kennedy Space Center, Florida, February, 1987.

4. Hankins, G.B., Jordan, J.W., Katz, J.L., Mulvehill, A.M., Dumoulin, J.M., and Ragusa, J.M., "EMPRESS Expert Mission Planning and REplanning Scheduling System", M8533, Mitre Corp., Bedford, Massachusetts, September, 1985.

5. Mulvehill, A.M., "Cargo Expert System Feasibility and Evaluation Phase", MTR-9746, Mitre Corp., Bedford, Massachusetts, September, 1985.

6. Mulvehill, A.M., "A User Interface for a Knowledgebased Planning and Scheduling System", MTR-10175, Mitre Corp., Bedford, Massachusetts, November, 1986.

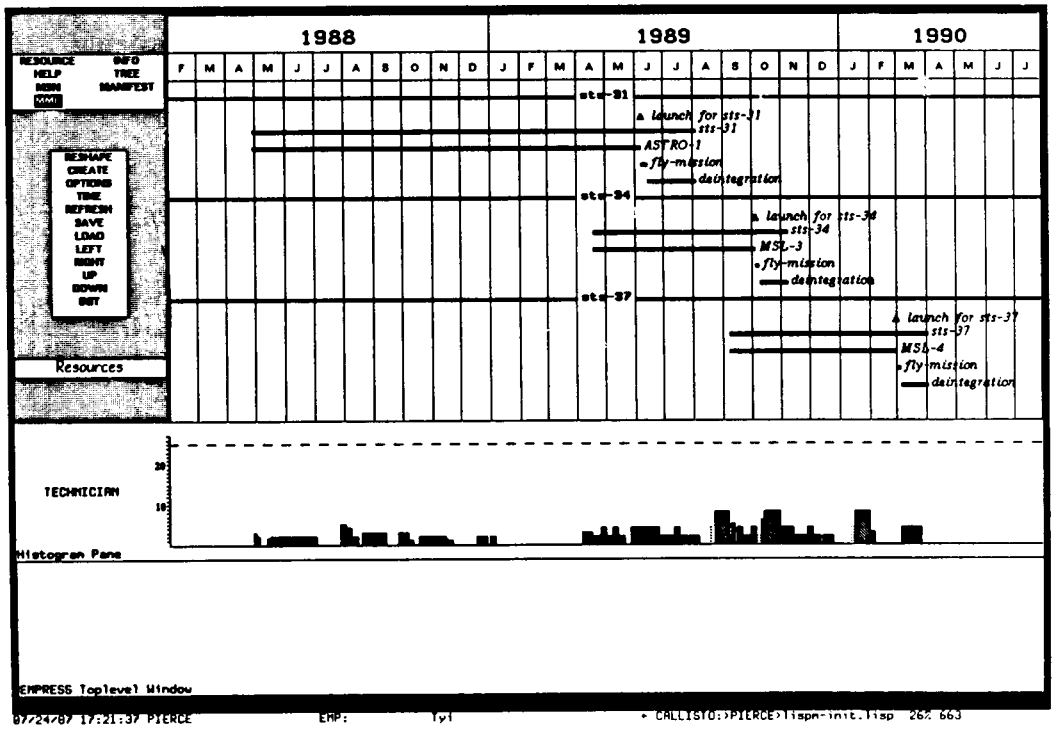


Figure 7 - Example of a resource histogram using EMPRESS.

Trimodal Interpretation of Constraints for Planning

by
David Krieger and Richard Brown
MITRE Corporation, Bedford, MA 01730

1. ABSTRACT

Constraints are used in the CAMPS[1] knowledge-based planning system to represent those propositions that must be true for a plan to be acceptable. One mode of interpreting a constraint determines its logical value. A second mode inverts a constraint to restrict the values of some set of planning variables. CAMPS introduces a third mode: the "*make*" mode. Given an unsatisfied constraint, *make* evaluation mode suggests planning actions which, if taken, would result in a modified plan in which the constraint in question may be satisfied.

These suggested planning actions -- termed *delta-tuples* -- are the raw material of intelligent plan repair. They are used both in "debugging" an almost right plan and in replanning due to changing situations. Given a defective plan in which some set of constraints are violated, a *problem-solving strategy* selects one or more constraints as a focus of attention. These selected constraints are evaluated in the *make* mode to produce *delta-tuples*. The problem-solving strategy then reviews the *delta-tuples* according to its application and problem-specific criteria to find the most acceptable change in terms of success likelihood and plan disruption. Finally, the problem-solving strategy makes the suggested alteration to the plan and then rechecks constraints to find any unexpected consequences.

2. Background

Modern planning systems usually distinguish the "what is" knowledge that captures the salient features and constraints of a plannings application from the strategic reasoning that effectively applies such knowledge to accomplish some goal.[Davis80] [Wilensky80] [Stefik81a]. The latter, typically termed meta-rules or meta-knowledge, provides an explicit and extensible representation of the control strategies required for intelligent planning. The CAMPS Metaplanning [Brown85] component provides a mechanism for posting goals to the system and utilizing a mix of declarative *meta-plans* and procedural *standard control flows* to accomplish these goals. In so doing, new subgoals are posted and new metaplans are instantiated to accomplish these subgoals. The resulting hierarchy of active problem-solving agents provides global control over local planning actions (e.g., filling in a slot or checking a constraint).

The problem-solving agents provide CAMPS with a high-level, top-down view of planning and resource allocation. However, problems with planning usually arise because some *detail* is out of place. This defect in a plan is signaled to the metaplanning component via a *constraint violation*. The third of CAMPS' three modes of constraint evaluation, **make-mode**, is intended to provide a low-level, bottom-up view of the planning problem by producing a structure called a **delta-tuple** that suggests some action the problem-solver might take to resolve the problem and eliminate the constraint violation.

Make-mode evaluation is not guaranteed or even intended to provide an exhaustive list of all possible corrective actions available to the planning system. The metaplanning component itself can use high-level strategies for resolving multiple constraint violations in a global manner. Rather, make-mode evaluations is an attempt to extend the role of the CAMPS primitive operators (CAMPS predicates), and through the delta-tuples, involve them in the plan repair process. This does not absolve the metaplanning component from its primary responsibilities, i.e., preventing destructive subgoal interaction/annihilation and limiting the combinatorics of the search through the space of possible plans.

The ability to respond to unforeseen conditions in the environment (replan) is a major design goal of CAMPS. In order to achieve replanning capability, problem-solving strategies and predicates must communicate in an orderly manner. Delta-tuples typically suggest ways of undoing some planning decision which, due to the limitations of CAMPS look-ahead mechanism or, more importantly, due to some unforeseeable change in the planning environment, that has made a suggested plan unacceptable.

2.1 CAMPS's Application-specific Knowledge

CAMPS organizes its *domain* knowledge around an AKO hierarchy of *plan elements*. Plan element instances represent specific tasks, resources, and other objects within a CAMPS application domain. Using the nomenclature of "frames," a plan element instance has associated with it a number of *slots*, some of which may contain instances of, or sets of instances of, other plan elements. In this way, the plan element hierarchy also represents some of the relationships that can exist between objects.

3. Defining Constraints

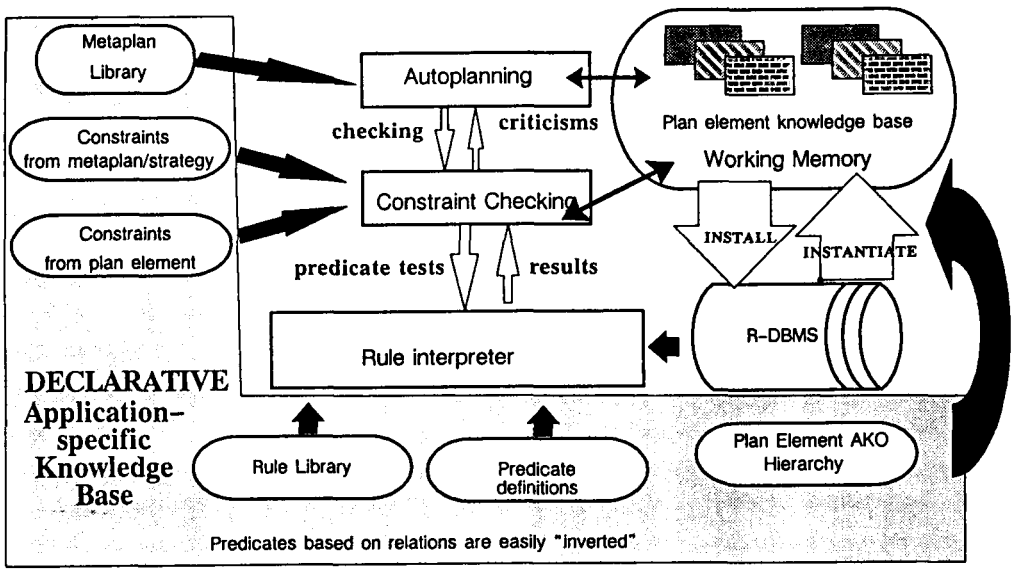
3.1 Planning as Constraint Satisfaction

CAMPS views mission planning primarily as a constraint satisfaction problem [Stefik81b] [Fox83]. Constraints describe relationships that must hold among the slots of plan elements. In its simplest form, a constraint simply says that some relationship should *always* be enforced ,i.e. there are no limiting conditions under which the constraint does not apply. This relationship is expressed as a **predicate** applied to constants and variables. Following a LISP-like syntax, a relationship may look like:

```
(PREDICATE-SYMBOL ARG1 ARG2 ...).
```

[1] This reports on work conducted by the MITRE-Bedford Artificial Intelligence Center under project 8400B, sponsored by NASA/KSC. Additional sponsorship for the implementation of the CAMPS architecture was provided by the Rome Air Development Center, COES.

Figure 1. The CAMPS architecture uses several types of declarative knowledge to support planning applications in different domains. Information about partially completed plans is kept in a relational database. When operating, information is read into working memory. Choices for filling slots in plan elements (made by the user or automaticallly in the user's behalf) are subjected to constraint checking. Finally, work is saved back into the database.



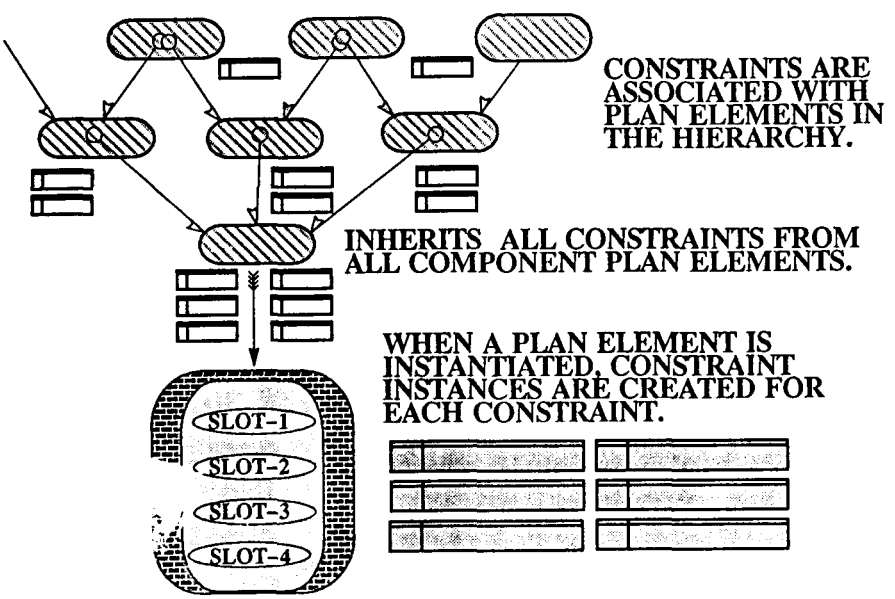


Figure 2. Planning -- solving a constraint satisfaction problem -- centers on filling slots in plan elements subject to constraints. Types ("capabilities of") plan elements are arranged into a specialization (AKO) hierarchy along which slots (but not slot values) and constraints are inherited.

For example, a constraint that enforces the condition that the start of a task must be no earlier than the task's earliest-start (as determined by some problem-solving strategy) might be of the form:

```
(*GREATER* ?START ?EARLIEST-START)
```

The variables, syntactically flagged by a leading "?", are usually associated with slots of a plan element, so that `?EARLIEST-START` is a variable associated with the `:EARLIEST-START` slot of the plan element instance whose class includes the essential-event capability.

3.2 Constraint Declarations

Constraints are central to the topic of this paper. We will show how the various modes of constraint interpretation interact during plan construction and repair. Each constraint declaration creates a data structure with the following attributes:

Plan Element. The plan element that is the focus of the constraint.

Involved Slots The slots for which the relationship is enforced.

Conditions This is a list of predicates, some of which may be negated. The constraint is applicable only if the conditions are satisfied.

Relationship (Predicate) A constraint defines a relationship between slots of a plan element. This relationship is expressed as single, possibly negated predicate that defines the constraint.

3.3 Constraint Consequence

Ideally, a plan should not have any violated constraints. However, it is naive to believe that planning problems have solutions in which all constraints are satisfied. The CAMPS architecture uses a numeric measures of that reflects the *degree of belief* in a constraint being violated.

In addition to *degree of belief*, CAMPS also associates a **consequence category** with every constraint. The consequence category provides a *qualitative* measure of the seriousness of the constraint's violation. The consequence category helps to order the constraints for evaluation and provides a metric by which problem-solving strategies can selectively check and relax constraints. Consequence categories include *feasibility, survivability, success, efficiency* and *assumption.*

4. Variables in CAMPS

CAMPS variables occur as arguments in predicate expressions. Generally speaking, a CAMPS variable is unrelated to a LISP variable. A variable typically, but not always, corresponds to a slot in some plan element. Conversely, at an implementation level, each slot can be mapped to a CAMPS variable, which in turn holds what we heretofore have informally spoken of as "the slot's value."

The **status** of a variable reflects the nature of the its value. If the value of a variable is unknown its status is `:UNATTACHED`. If it is bound to a single value its status is `:FIXED`. If there is a problem-solving strategy that claims to be able to suggest values for the variable, the variable's status is `:CANDIDATES`. The strategy in this situation is termed "a generator."

Conceptually, a **generator** is a list of candidates. They are implemented as streams that return candidates from a set of possibilities. Number generators, for instance, provide a stream of all numbers between plus and minus infinity. Such a generator could receive a restriction messages that would restrict candidates to positive integers. Another `:RESTRICT` message sent to that generator could further restrict candidates to numbers less than seven.

CAMPS also supports generators that deal with objects other than numbers. An entire class of generators deals with plan elements. `:RESTRICT` messages sent to an instance of this class of generators could be based on plan element capabilities.

The primary purpose of CAMPS variables is to facilitate unification. Unification is the process whereby two patterns consisting of constants and variables are matched in such a way as to bind variables in one pattern to variables in the other. Unifying two variables not only forces them to have the same value but ensures that anything that affects one will also identically affect the other.

5. MODES of Constraint Evaluation

5.1 Normal mode evaluation

Predicates evaluated in normal mode are treated much the same as predicates in formal logic. One important distinction is that a CAMPS predicate can indicate that insufficient information is available: CAMPS predicates can return *TRUE, *FALSE (or NIL if there is not enough information to determine the predicate's logical value). Predicates in CAMPS also return two additional values: a **degree of belief** that the predicate is true and a degree of belief that the predicate is false. This last value is also known as the **degree of disbelief**. The second and third values returned support the Dempster-Schafer model of reasoning with uncertainty.

The logical value returned by the predicate is a function of the belief-disbelief values. If belief exceeds disbelief by a certain threshold, the predicate's logical value is *TRUE. If disbelief exceeds belief by that threshold, the predicate's logical value is *FALSE. If the difference between belief and disbelief does not exceed some minimum value, the predicate returns NIL meaning that the evaluation could not determine the truth or falsehood of the proposition[1].

A constraint that enforces a relationship between two slots of a plan element is either **satisfied** or **violated** depending on the logical value returned by its predicate. Formally, the predicate being *TRUE indicates that the constraint is VIOLATED, so that a slot is "acceptable" if the logical disjunction of it's constraints is *FALSE.

5.2 Bias mode evaluation

Following a general strategy of "delayed commitment" [Sacerdoti77], CAMPS has the ability to "look ahead" before fixing the value of a slot. In bias-mode, *predicates are evaluated for their side effects* on the generators associated with the CAMPS variables serving as arguments to the predicate. These side effects usually take the form of restricting an unattached variable to a set of acceptable candidates by attaching a generator to the variable, further restricting the set of a variable's candidates by sending its attached generator an appropriate :RESTRICT message, or (when we're lucky) fixing a variable's value to a single candidate.

As planning proceeds, constraints are evaluated in bias-mode, producing sets of candidates for each variable associated with a slot of a plan element. A slot will typically have several constraints associated with it, all of which are trying to be satisfied. One constraint will restrict a variable's value to a certain set of candidates; a second constraint might further restrict that set of candidates to a subset of the first and so on until a set of acceptable values is produced and/or some constraints post violations.

This approach postpones fixing the value of a slot until as much information as possible has been considered, guiding the constraint satisfaction process towards a successful conclusion without needless search through the space of (im)possible plans.

CAMPS provides two unification contexts. In bias-true, unification "attempts" to return a *TRUE result by binding the predicate variables appropriately; in bias-false, unification attempts

---

1 Note that this scheme can distinguish between **absence of information** and **contradictory information**. A belief/disbelief pair (0.5 0.5) indicates a lot of contradictory information, while (0.0 0.0) indicates a complete absence of information. Since we will use the same scheme to describe "confidence" in a suggested corrective action, this allows us to distinguish between a suggestion with little to recommend it (0.2 0.0), and a good suggestion with a lot of risk (0.6 0.4).

to impose restrictions on the variable that produce a *FALSE result The overall effect of bias mode evaluation is to avoid search by initially trying to restrict variables to values that have a good chance of simultaneously satisfying interacting constraints.

5.3 Make- mode evaluation

A violated constraint is often serious enough that a problem-solving strategy will choose to try to fix it. The basic idea behind behind make-mode evaluation is that the predicate associated with the violated constraint has useful information about how to fix itself. When an unsatisfied constraint is evaluated in a make-mode, a list of *delta-tuples* is returned. These delta-tuples suggest ways in which the violated constraint might be satisfied. They embody the *local* knowledge about the constraint, or more specifically, its associated predicate and conditions. It is the task of some higher level problem-solving strategy to evaluate these suggestions in terms of the overall planning goal and eventually choose one or more to execute.

6. Using Make- Mode Evaluation

The three modes of constraint evaluation -- **normal**, **bias**, and **make** are closely connected. **Normal** sees if the plan is in trouble; **bias** tries to keep the plan out of trouble; while **make**-mode suggests ways to keep the plan out of trouble it is already in.

6.1 Crucial Ideas

The key elements of CAMPS's solution to a planning problem, evidenced by having one or more constraint violations, depends on answering the following questions:

- What change can be made to a plan that will fix the problem?
- If the change is made, what is the belief that the problem will be corrected?
- What is the expectation that the proposed change will trigger a *ripple* effect? That is, will fixing this problem in the prescribed manner introduce a disproportionate number of new and perhaps more difficult problems?
- What problem-solving strategy is responsible for making the change?

Both local and global reasoning is required to address these issues during plan refinement and replanning. The metaplanning component of CAMPS provides the global perspective. Make-mode evaluation of constraints and the delta-tuples returned, provides the local perspective. In particular the **belief** that the proposed change will correct the difficulty combined with the **disbelief** derived from the *ripple likelihood* provide a reasonable selection criteria upon which a more global criteria can be based. CAMPS uses both of these perspectives in a combined top-down bottom-up approach to generate acceptable plans.

6.2 Delta Tuples

Specifically, a delta-tuple describes a possible way to modify a plan element in order to satisfy a violated constraint. As such, it must embody the essential features of some planning action such as rescheduling a task, using a different number of resources or using a different resource altogether. Implementationally, these planning actions typically distill down to changing the value of some slot of some plan element instance. Thus, **delta tuples** carry the following information:

- **Plan Element**. The plan element to be modified by the delta-tuple.
- **Slots**. A list of the slots expected to change if the action suggested by this delta tuple is taken. This list gives the responsible problem-solving agent a basis for selecting among alternative delta tuples. A reasonable selection criteria might try to localize the effects of carrying out a suggested action by minimizing the number of slots changed. Another might try to avoid changing slots that the user specifically set or prefers to not change.
- **Success Predicate**. The predicate expression that will evaluate *TRUE if the suggested planning action is taken. This may or may not be the predicate of the constraint whose violation is being repaired.
- **Recipient**. In all cases, the suggested action will take the form of a message to be sent to some message-receiving object. This recipient could be the plan element itself, the generator associated with a changing variable, or a responsible problem-solving strategy.
- **Message**. The actual message sent to the recipient to effect the change.
- **Arguments**. The appropriate message arguments, typically plan elements.
- **Confidence Pair**. A Dempster-Schafer pair of confidence measures which give the *a priori* belief that the change proposed by the delta-tuple will satisfy the predicate or constraint.

6.3 Generating Delta-Tuples

The information needed to generate delta-tuples resides with the predicates. Each predicate has a set of general *operations* for changing the value of one or more of its arguments such that the predicate will be satisfied. For example, CAMPS has a predicate that enforces the relationship that `ARG1` must be greater than `ARG2`.

```
(*GREATER* ARG1 ARG2)
```

Associated with the predicate are two make-true operations, `:MAKE-GREATER` and `:MAKE-LESS`. If a violated constraint involving this predicate is evaluated in make-true mode, CAMPS will attempt to create a delta-tuple by applying the `:MAKE-GREATER` operation to `ARG1` and another delta-tuple by applying the `:MAKE-LESS` operation to `ARG2`.

In order for these operations to have any meaning in terms of the plan being generated, CAMPS must first trace the source of each variable serving as a predicate's argument. In the simplest case, the variable corresponds directly to a slot of a plan element. In that case, the delta-tuple would specify a *message* to be sent to a *plan element* to restrict the value of that slot variable's generator to the newly computed set of acceptable values.

In summary, make-mode evaluation is built on the following approach:

- Each predicate has associated with it a set of general operations that if applied to its arguments would satisfy the predicate.
- Each variable bound to a predicate argument is examined to determine if and how these operations could be properly applied to the variable.
- These operations are then applied to the variable, yielding a new suggested value or range of values for the variable that promises to satisfy the predicate. It is at this point that a delta-tuple is created.

7. Example

Make-mode constraint evaluation enables CAMPS to intelligently resolve the resource conflicts that present an especially difficult problem. Most planning systems deal with many types of resources and many tasks competing for those resources. The consequence of modifying a resource allocation to satisfy one task will likely effect the viability of some other task. Our example is from the

NASA cargo loading application. Integration of experiments into racks for one mission will typically overlap with the rack-experiment deintegration of previous missions. This situation is further exacerbated by schedule changes and mission payload reassignments that force replanning and rescheduling.

CAMPS represents sets of homogeneous resources as resource-pools. In the NASA application, each rack used for holding a Space-Lab experiment is distinguishable, and is represented as a RACK unit quantity resource-pool so that racks are tracked individually (e.g., by serial number). A separate plan element including the RESOURCE-UTILIZATION capability records a single resource requirement being met from a single pool; it includes the following slots:

:CONSUMER the task requesting the resource
:RETURNING-TASK the task returning the resource.
:SUPPLIER the resource-pool supplying the resource
:BEGIN the time that the resource is first need, directed to the :START slot of the :CONSUMER.
:END the time the resource is returned to the pool, directed to the :FINISH slot of the :RETURNING-TASK.
:EARLIEST-BEGIN is the :EARLIEST-START of the :CONSUMER.
:LATEST-END is the :LATEST-FINISH of the :RETURNING-TASK.

RACKS include the RESOURCE-POOL capability, which tracks availability of the resource. Each RESOURCE-POOL includes the following information in an internal (i.e., "non slot") form:

:ALLOCATIONS A list of resource-utilizations
:AVAILABILITY A list of sublists, each of which indicates a duration and the quantity of resources available during that duration.

A constraint attached to the RESOURCE-UTILIZATION capability requires that the :SUPPLIER have sufficient quantity of the resource between :BEGIN and :END times. As planning progresses this constraint, which uses the *RESERVE* predicate in its relationship, is checked in bias-true mode. When satisfiable, a *RESERVE* predicate evaluated in bias-true mode will have several side effects:

The new RACK-UTILIZATION is pushed on to the pool's ALLOCATION list and the pool is put into the RACK-UTILIZATION's :SUPPLIER slot. Also, availability of the pool is updated to reflect the new resource utilization.

A failed reservation constraint means that the designated RACK could not supply the quantity of resources (in this case, 1) requested by the :CONSUMER because some other task(s) reserved the resource at an overlapping time interval. For example, changing a mission's launch date will usually cause once-successful rack utilizations to suddenly have violated constraints involving *RESERVE* predicates.

The user or an automated problem-solving strategy can focus on one of these constraints, and ask CAMPS to suggest corrective actions by evaluating the *RESERVE* predicate in the **make-true** mode. The resulting delta-tuples typically include:

Reschedule the task's start or end time to the closest time when the required resource becomes available for the required duration. This would amount to sending a :RESTRICT message to the generator attached to the task's START or END variable in order to restrict its value to a new acceptable range.

Change the resource requirement if the task's requirements can be met by some other resource with similar capabilities or by some other resource-pool.

Modify conflicting tasks by finding those reservations in the resource pool whose rescheduling would enable the task-at-hand's resource requirement to be met and reschedule those reservations.

Reduce the quantity of resources requested by the task to the number of resources available at the time. For resource-pools of only one resource, (as is the case with experiment racks) this is not an option and for those cases CAMPS will not generate a delta-tuple that suggests using zero of that resource.

A simple example dealing with rack resources is shown in figure-3. The RACK-DEINTEGRATION task of mission STS-1002[2] overlaps with the RACK-INTEGRATION task of STS-1003. In this particular case, the same rack SN003-s is desired by both tasks. The *rack-resource-utilization* associated with STS-1003 posts a constraint violation that reflects its failure to reserve its desired resource. Figure-6 shows the delta-tuples returned by re-evaluating that constraint in make-mode.

The belief/disbelief pairs shown embody several general considerations. Each one of these tasks has an earliest and latest start and earliest and latest finish that represent acceptable slack in the schedule. For the delta-tuples dealing with changing start and end times, the belief decreases and the disbelief increases as the difference between the original and suggested time increases.(figure 5) For the delta-tuples that suggest using another resource, the belief is a function of the number of other resources in that pool and/or the number of pools with the same rack capability[3].

Only a glance at figure-6 shows that one delta-tuple seems the most likely to succeed. The final step in the planning process is to invoke the chosen delta-tuple by sending the message and arguments off to the specified recipient.

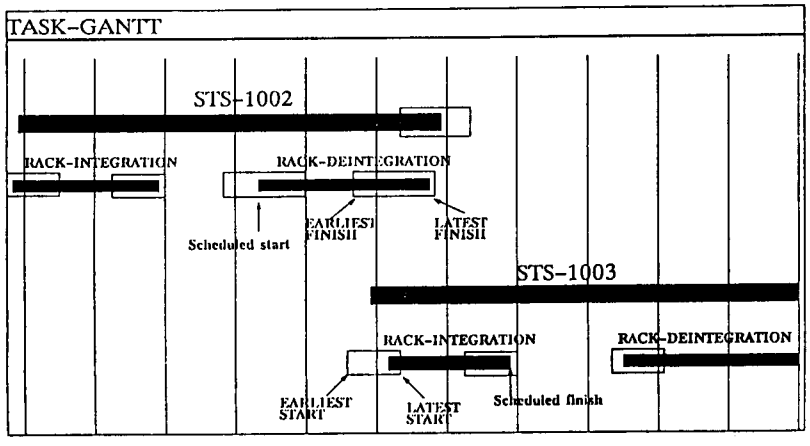
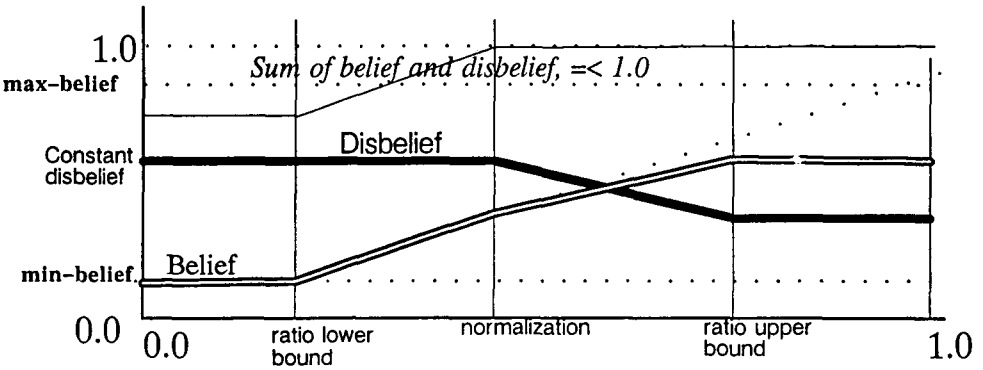


Figure 3. A Gantt chart showing a very small portion of a payload preparation for two hypothetical space lab missions. Rack integration -- putting experiments into racks and racks into the module -- is the consumer task of racks, while rack deintegration readies the rack for reuse.

Figure 4. Calculating a belief and disbelief for a suggestion. Given the number of pools believed acceptable but not yet considered (e.g., the number of remaining candidates) and the number of pools, calculate the ratio. A low ratio indicates that most potential candidates have been rejected. Disbelief is a constant, perhaps derived from the type of the resource. Normalization forces the belief and disbelief to total less than 1 by multiplying each by 1/(belief+disbelief) when their sum exceeds 1.0.



Ratio of acceptable pools unconsidered (assume: 1) *to* pools at facility (assume: number of instances)

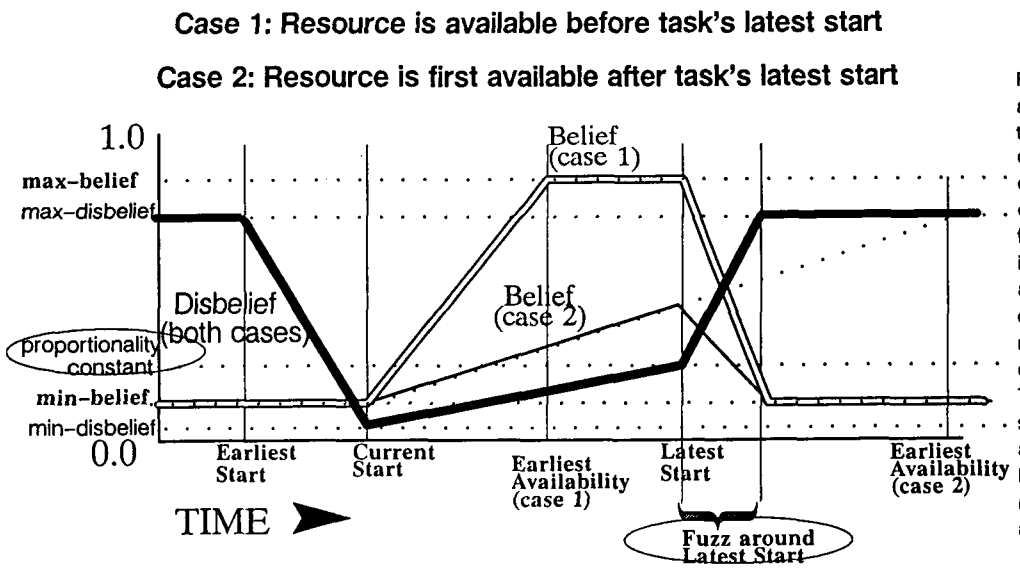


Figure 5. Unnormalized belief and disbelief as a function of the delayed start. Two arbitrary constants are used: a constant describing how much more disbelief increases as a function of time after the start is delayed beyond the currently assumed "latest start;" and a constant which, when multiplied by the task's current duration, gives a tolerance. Two cases are shown: (1) shows the resource becoming available before the task's latest start; (2) shows the resource becoming available after the task's latest start.

Plan Element	Slots	Message	Recipient	Confidence Pair
Overall-Rack-Deintgration-1002	(:END)	:DELTA-RESTRICT	Overall-Rack-Deintgration-1002	(0.65 0.35)
	Suggests to finish deintegration earlier			
Overall-Rack-Integration-1003	(:START)	:DELTA-RESTRICT	Overall-Rack-Integration-1003	(0.35 0.45)
	Suggests to start integration later			
STS-1002	(:START)	:DELTA-RESTRICT	STS-1002	(0.1 0.8)
	Suggests to switch in mission sequence			
Rack-Resource-Utilization-1002	(:SUPPLIER)	:DELTA-RESTRICT	Rack-Resource-Utilization-1002	(0.1 0.8)
	Suggests using a different rack			

Figure 6. Given the constraint violation of figure 3, EMPRESS-II returned four delta-tuples as shown by this table. While we argue that selecting the "best" delta-tuple should be mediated by higher-level problem-solving strategies, we can see that simply picking the "most believed" suggestion is a good general heuristic.

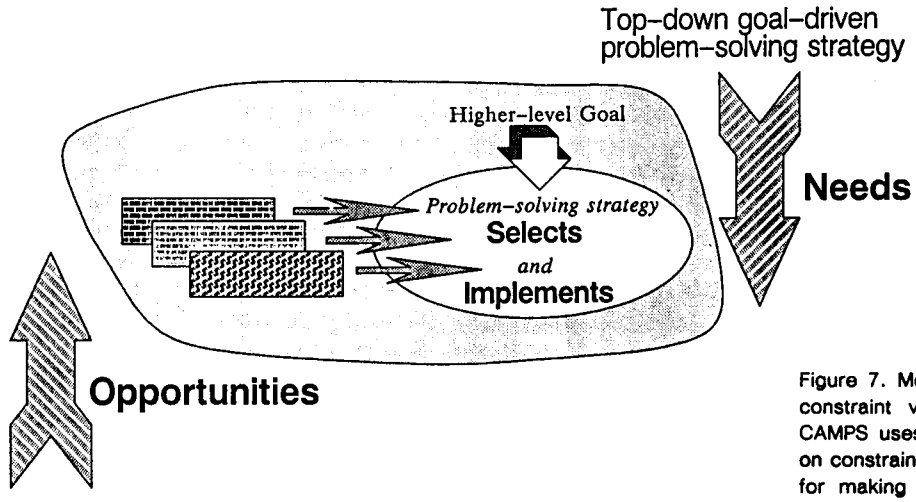


Figure 7. Most planning systems view constraint violations as "problems." CAMPS uses MAKE- mode evaluation on constraints to suggest opportunities for making a plan better. Note: the effort of determining that a constraint is violated is about 90% of the effort needed to suggest the "obvious" ways to correct the violation.

8. Conclusion

Like many recent planning systems, CAMPS builds on a hierarchically structured metaplanning component. This component represents the planning system's "self-knowledge" and is used to control the application of primitive operators (CAMPS predicates) towards plan realization. In addition, make-mode constraint evaluation imbues the CAMPS primitive operators themselves with a degree of "self knowledge." When the planning process runs into difficulties, a dialog commences between a globally-oriented problem-solving strategy and the local constraint instances. The dialog is initiated by a problem-solving strategy that examines the violated constraints and selects one or more to evaluate in make-mode. These constraint instances dutifully return a list of suggestions embodied in the delta-tuples.

The returned delta-tuples are examined by the problem solving strategy which can choose one or more to apply or seek some alternative means of plan repair altogether. The purpose of the delta-tuples is to pose those simple alternatives that would typically cause a minimum of plan disruption and reconstruction. They embody the generally accepted heuristic of any intelligent problem solver: "Consider the easy things first."

9. Bibliography

[Brown85] Brown, Richard. Agendas: A Metaplanning Mechanism. M-series M85-26, MITRE Corporation, Burlington Road, Bedford MA 01730, July 1985.

[Brown86] Brown, Richard, "A Solution to the Mission Planning Problem" in *Proceedings of the Second Aerospace Applications of Artificial Intelligence,* Dayton, Ohio, October 13-17, 1986.

[Davis80] Davis, Randall. Meta-rules: Reasoning about Control. *Artificial Intelligence*, 15, 1980.

[Fox83] Fox, Mark S. Constraint-Directed Search: A Case Study of Job Shop Scheduling. PhD thesis, Carnegie-Mellon University, 1983.

[Sacerdoti77] Sacerdoti, E. D., A Structure for Plans and Behavior, American Elesevier, New York, 1977.

[Stefik81a] Stefik Mark. Planning and Meta-Planning. (MOLGEN: Part 2), Artificial Intelligence 16(2) 1981.

[Stefik81b] Stefik Mark. Planning with Constraints. (MOLGEN: Part 1), Artificial Intelligence 16(2) 1981.

[Wilensky80] Wilensky, Robert. Meta-Planning. First NCAI 334-336, August 1980.

2 None of these mission numbers correspond to ones actually planned. They *are* derived from a NASA/KSC planning exercise that determined whether there were enough racks for missions through the year 2000.

3 These numbers are based on general considerations and cannot reflect situation specific knowledge that might clearly favor a suggested course of action in spite of its low belief. However in most cases these numbers provide a good indication of the relative merits of one suggestion over another. It is typically better to shift a subtask a little than reschedule its entire parent task; and using another resource has a better chance of succeeding if many similar resources can be found.

RANGE AND MISSION SCHEDULING AUTOMATION USING COMBINED AI AND OPERATIONS RESEARCH TECHNIQUES

Mansur Arbabi, Ph.D.
Michael Pfeifer
International Business Machines Corporation
Federal Systems Division
18100 Frederick Pike
Gaithersburg, Maryland 20879

ABSTRACT

Ground-based systems for Satellite Command, Control, and Communications (C) operations require a method for planning, scheduling and assigning the range resources such as: antenna systems scattered around the world, communications systems, and personnel. The method must accommodate user priorities, last minute changes, maintenance requirements, and exceptions from nominal requirements.

Described are computer programs which solve 24-hour scheduling problems, using heuristic algorithms and a real-time interactive scheduling process. The computer utilized is an IBM System/370, Model 3081, and an IBM 3279 color graphic display.

INTRODUCTION

Ground-based systems for Satellite Command, Control and Communications (C) operations require a method for planning, scheduling and assigning the range resources such as: antenna systems scattered around the world, communications systems, and personnel. The method must accommodate user priorities, last minute changes, maintenance requirements, and exceptions from nominal requirements.

Recognizing this need and its potential application to programs such as Data System Modernization (DSM) for the U.S. Air Force Satellite Control Network, IBM has pursued an Independent Research and Development (IRAD) effort to investigate a means of automating the scheduling of range resources for a satellite ground-based C system.

In existing systems, schedules typically are manually prepared for times in the future ranging from many months to one day, and, in some cases, near real-time changes must be accommodated. This manual scheduling is a very labor-intensive process and, at best, it offers scheduling accuracy of one minute. Over the past few years, the number and complexity of satellites have increased significantly. These increases have strained the capacity of manual scheduling, necessitating the analysis of automated scheduling techniques.

This article addresses the results of the three-year research project undertaken by IBM's Federal System Division at Gaithersburg, Maryland. Described is a computer program which solves 24-hour scheduling problems, using heuristic algorithms, in less than two minutes on an IBM System/370 Model 3081 using an APL interpreter under MVS. This program provides results in user selectable time unit granularity, and with accuracy constrained by computer precision limits.

RANGE SCHEDULING

The range scheduling problem involves allocation of range resources to satellite operations. The allocation process is done for planned activities which range from six months in the future to near real time. The problem is complicated by time constraints and last minute modifications. The range scheduling function will be subjected to increasing pressure as the number of space vehicles increases. The severity of this problem is increased by other factors such as the addition of antennas and other resources at existing sites, reduced turnaround time, and increased demand for shared resources. It is also important to take full advantage of future increases in computational capability to sustain a high level of system utilization.

There are several technical issues related to this effort. Many schedules are required to cover time periods from 24 hours to six months. The schedules satisfy different purposes and must be presented in appropriate levels of detail. User requests for services are not static, and provisions must be made for changes on short notice. Many priorities must be accommodated. Allowances for schedule modifications due to malfunctions in either the satellite or the ground support equipment must be taken into account. A method of presenting automatically developed schedules in a meaningful way is essential to the success of automated range scheduling. It is expected that the users will interact with the system to generate and modify schedules.

As a background for this task, a thorough investigation of previous work on scheduling of ground resources in support of satellite operations was made. It included both NASA and DoD scheduling efforts.

The only automated scheduling found was for small problems (less than 50 requests, two antennas with short windows). It was also found that many agencies within DoD and NASA are interested in a solution to the same problem and are investigating generalized scheduling techniques.

Objective

The objective of this research was to determine the feasibility of computer-generated range scheduling and to demonstrate such a capability.

Approach

Many of the functions performed by the schedulers can be performed readily by computers. Certain other functions require further research to bring them closer to automation. The functions of the scheduling process considered for this study are those which had not been previously automated. These are:

-request processing

-production of weekly schedules

-scheduling conflict identification and resolution

-production of daily support schedules

-real time schedule changes

An overview of the scheduling function is shown in Figure 1.

Remote telemetry, tracking and command antennas, located around the world, are used to communicate with satellites when they are within line-of-sight range. Each antenna can "contact", at most, one satellite at a time.

Before an antenna can be used for a contact, there is a certain amount of "set-up" time or "turnaround" time that is required by the ground crew to reconfigure the antenna system.

Users place demands on the system by requesting that blocks of (contiguous) antenna time (also known as contact times) be allocated to their satellites. These requests can take various forms. Often, but not always, the specific antenna and exact time for the contact are not specified. An antenna preference, a contact priority, and an earliest and latest time for the contact may, however, be given.

Users may request periodic contacts or multiple simultaneous contacts. Finally, users may request continuous contact with their satellite over long periods of time. These requests can be met by piecing together overlapping contacts from multiple ground antennas. The process is called a "hot handoff".

There are also non-satellite support requests which are confined to a single antenna for such purposes as preventive maintenance.

Since normally more ground antenna support time is requested by the various users than is available, conflicts in the user requested support must be resolved. A good scheduling algorithm can minimize these conflicts and help alleviate this situation.

When stated generally, the scheduling problem is quite difficult. Linear programming techniques have little (practical) pay-off for the scheduler. The basic problem can be reduced to a mixed integer linear program, but

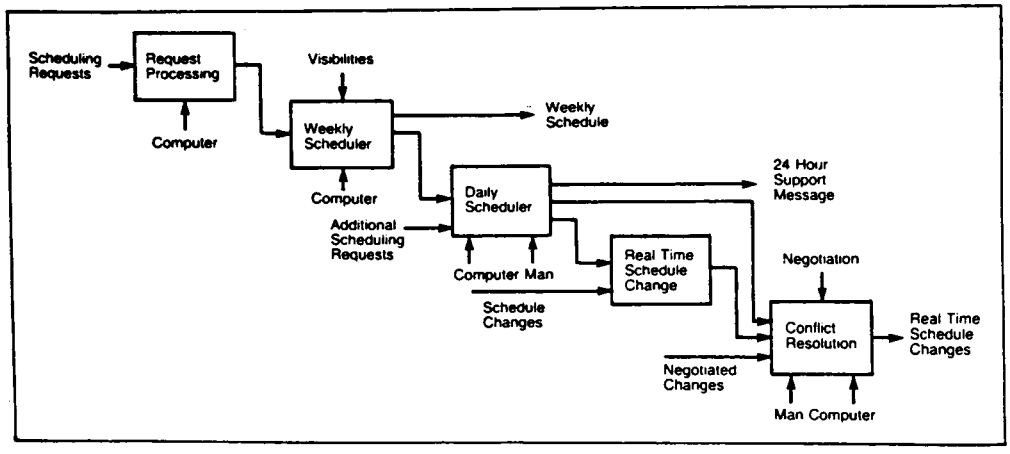


Figure 1. Overview of Scheduling Functions

this is of little (practical) consequence due to the large number of variables that are required. A heuristic technique would appear to be the only feasible approach.

Fortunately, real-world problems have additional attributes. Satellites tend to fall into one of three (almost) disjoint classes. Each class has it own special contact request pattern.

Low altitude satellites have an apogee of under 500 miles. They are visible over a ground antenna for only about ten minutes before they disappear over the horizon. Contacts, when requested, are for the entire time that the satellite is within the line of sight of an antenna. Most of the high priority requests come from this class.

Medium altitude satellites have an altitude which averages 12,000 miles and maintain line of sight with an antenna for up to 11 hours. Contacts re generally requested for 10 minutes' duration within a 45-minute window during which users may prefer a particular antenna to make the contact.

Near synchronous satellites have altitudes in the vicinity of 22,000 miles. If they are at the right position on earth, antennas can maintain line-of-sight contact for many hours (or continuous in the case of truly synchronous satellites). Users request varying length contact times and "hot handoffs" generally come from this class of satellites.

Non-satellite support requests are station-specific, but generally are fairly flexible as to when they are scheduled. The length of the support period ranges from 10 minutes to several hours.

The most significant accomplishment of this effort was the development of a new continuous time scheduling (CTS) algorithm for range scheduling. As described below, it is believed that the CTS algorithm demonstrates, for the first time, the feasibility of providing effective automation support to the complex scheduling operation.

BACKGROUND OF THE SCHEDULING PROBLEM

The CTS algorithm is the result of earlier work on scheduling that began in 1981. Initially, an in-depth review of the manual scheduling techniques was conducted. Not surprisingly, they were found to be very sophisticated. The scheduling problems encountered were very complex. Typically, they involved as many as 300 requests to be satisfied by as many as 14 antennas during a 24-hour period. The numbers are increasing year by year. Working over many years, manual range scheduling personnel have developed powerful tools for handling these requirements and have evolved a complex set of priorities, rules, and exceptions. Most of these have not been formally documented, but are learned by extensive on-the-job training.

On the average, the operationally certified range scheduling personnel each have more than ten years of experience. By observing current procedures over several weeks, including several continuous 24-hour periods, an appreciation was gained for the problem and for the sophistication and limitations of manual scheduling methods.

In parallel, an extensive survey of existing automated scheduling systems was conducted. Reviewing some commercial, DoD and NASA systems, it was found that none was suitable for the scheduling loads and complexity needed.

Accordingly, IBM's efforts were directed to develop a new approach. Initially, so-called mathematical programming models were considered that attempted to establish optimum schedules by simultaneously allocating resources to all the space vehicles. It was determined that such models were feasible for scheduling fewer than 50 requests, but that the storage requirements and run times associated with larger numbers were unacceptably large, increasing exponentially with the number of requests.

Next, several heuristic models that attempted to "duplicate" the scheduling rules used by the manual schedules were developed. After investigating these approaches, a so-called "discrete laxity algorithm" was devised. By scheduling one satellite vehicle at a time, this approach could develop reasonably good schedules with long, but marginally acceptable, levels of computer resources (for example, storage and run times).

The results were documented to allow a comparison to the present manual process. It was learned that the principal limitations of the discrete laxity approach were the five-minute time unit granularity and the inability to handle special case requests.

The CTS algorithm was developed in 1983 to remedy the discrete laxity limitations. Figure 2 shows the paths by which the several heuristic and optimization techniques were combined to arrive at the CTS solution that uses the best features of both heuristic and optimization methods. Figure 3 summarizes the mixed integer equations utilized in the schedule optimization problem.

A SIMPLE SCHEDULING PROBLEM

A simple example of the scheduling problem is presented. It consists of five requests: R1, R2, R3, R4 and R5. These requests are shown on the top diagram of Figure 4. Each request is specified by a duration, and a window having a start time and an end time. For example, request R1 has a duration of four time units and a window starting at time zero and ending at time 13. Request R5 has two separate windows. This example is a simple one since it is limited to a single antenna and to very simple request forms.

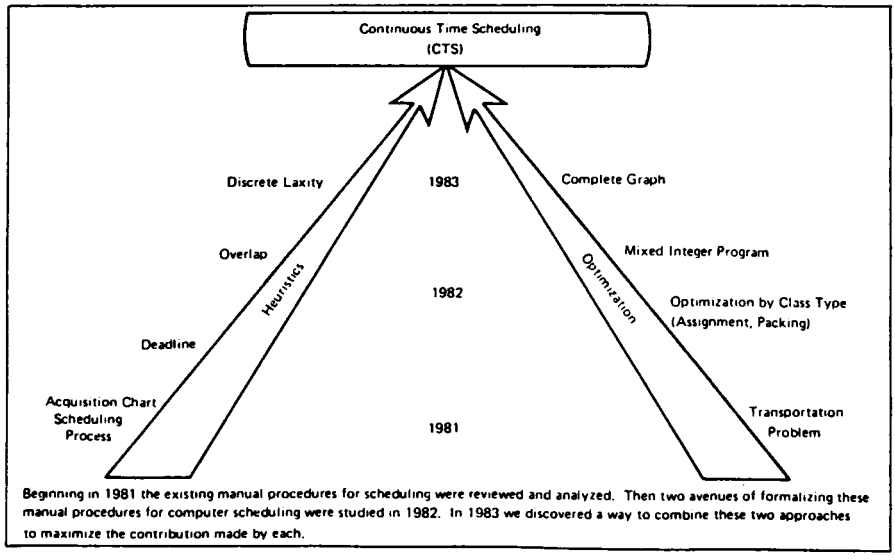


Figure 2. Progression of Scheduling Approaches

Objective Function:
Schedule as many requests as possible

$$\text{Maximize} \sum_{\forall K} V_K X_K$$

Constraints:
Schedule each request only once

$$\sum_{K \epsilon R_I} X_K \leq 1; \ \forall_I$$

Schedule each request within its window

$$A_K + S_K + C_K \leq B_K; \forall_K$$

Schedule requests to avoid concurrent resource use

$$A_J + S_J + C_J \leq A_K + S_K - T_K + M\delta_{JK} + M(2_i - X_J - X_K)$$

$$A_K + S_K + C_K \leq A_J + S_J - T_J + M(1-\delta_{JK}) + M(2 - X_J - X_K)$$

$$S_K \geq 0; \forall_K$$

$$X_K = (0, 1) \forall_K$$

$$\delta_{JK} = (0, 1) \forall (J,K) \epsilon P$$

Definitions

I	=	Request Index
J	=	Segment Index
K	=	Segment Index
A_K	=	Beginning of Segment K
B_K	=	End of Segment K
C_K	=	Length of request on Segment K
δ_{JK}	=	1 if request on K is started before request on J; 0 if otherwise
M	=	A large number (i.e., at least 3 times the scheduling period length)
P	=	The set of pairwise combinations of overlapping segment of each antenna
R_I	=	Set of segments which service request I
S_K	=	Offset between the beginning of Segment K and the beginning of its request
T_K	=	Turn around time of the antenna on Segment K
V_K	=	Preference value for scheduling a request on Segment K
X_K	=	1 if request I is scheduled on Segment K; 0 if otherwise
$\forall$	=	for all

This scheduling approach provides the best solution but unfortunately its computer running time and computer memory requirement grow exponentially with the number of input requests.

Figure 3. Mixed Integer Programming for Schedule Optimization

Before looking at the solution, it would be instructive for the reader to try various approaches such as "first come, first served" and "earliest deadline".

The steps necessary to arrive at a heuristic solution to this problem are shown in Figure 4. The top diagram is a barchart representing each request window. The problem is then represented in a graph form with all of its mathematical constraints. This graph formulation is then solved by evaluating the interaction between each request and the remainder of the requests.

The graph of the diagram of Figure 4 consists of nodes and links. The start time constraints for each request are shown by inequalities within the nodes of the graph; for example, the request R1 start time interval is:

$$0 \leq S \leq 9.$$

This means R1 could start any time between 0 and 9 and still remain in its window. When a request has more than one window segment, then for each segment there will be one time interval; for example, request R5, which has two segments, is shown with two sets of irregularities. The interaction between a set is shown by constraints on the links between the nodes of the graph, for example R1 and R3 start times S and S are free of conflict whenever:

$$S - S \leq 4$$

The last diagram in Figure 4 shows the solution tree. The request R2 is checked against R5, R1, R3 and R4 first for non-preemption and then for maximum laxity. Then the start time is selected for R2. In this manner we proceed from one request to the next until all requests are checked. The start times arrived at by this process are:

$$S = 9,\ S = 17,\ S = 13,\ S = 5,\ S = 0$$

Figure 5 shows the solution to the problem. Note that all the requests have been scheduled, and request R4 is scheduled in the middle of its window.

RESULTS

Figure 6 shows the results obtained in applying the algorithm to two representative problems. These scheduling results demonstrated that automated scheduling is indeed a viable alternative to manual scheduling. The average elapsed time required to develop a manual 24-hour schedule is 36 hours. The average number of weekly labor hours required to provide a manual 24-hour schedule is 645. The CTS algorithm scheduled greater than 98 percent of the input requests in less than three minutes of CPU run time on the IBM 3081 K32. It is expected that a 100 percent operationally optimum solution can be obtained with a small amount of manual intervention by the schedulers.

In Figure 7, computer run time is shown as a function of the number of requests schedule for a 24-hour sample problem.

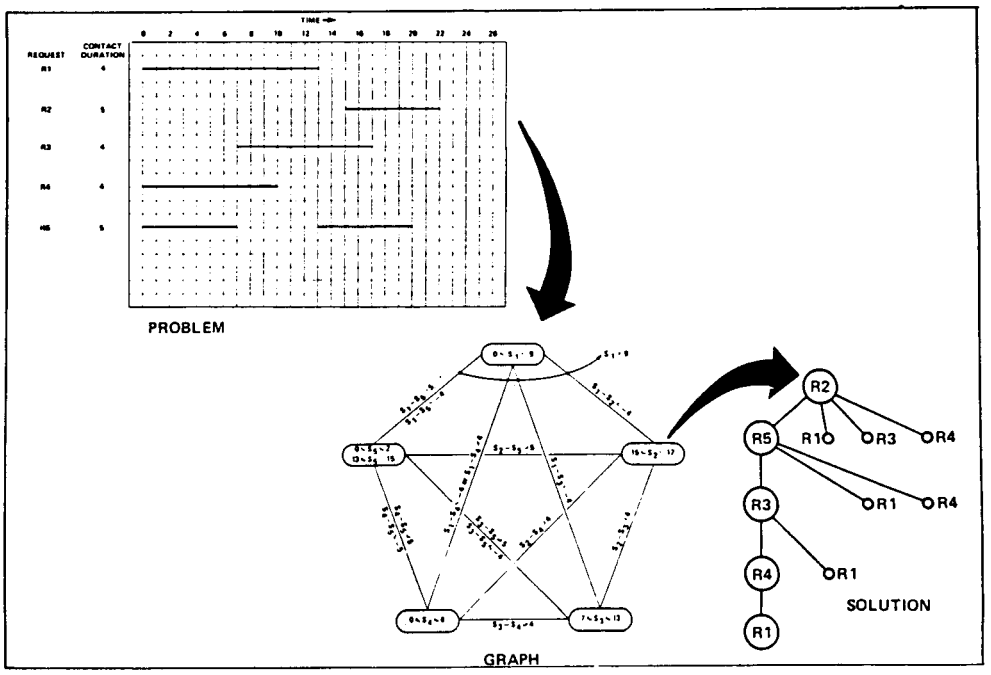


Figure 4. A Simple Scheduling Problem

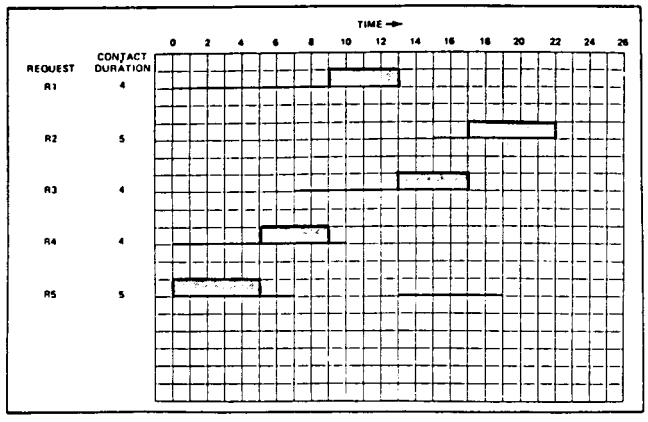
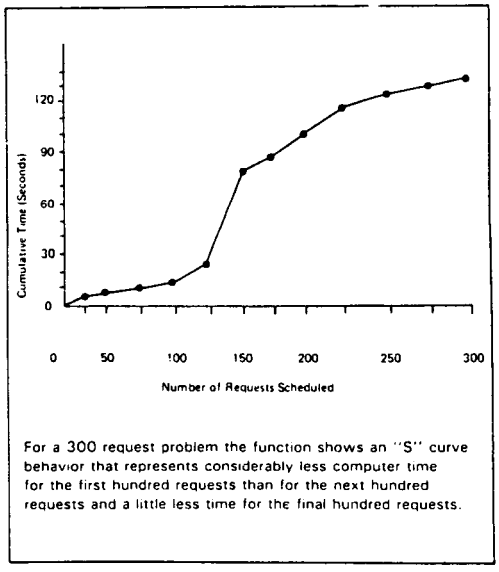


Figure 5. Solution--The Simple Scheduling Problem

SAMPLE PROBLEMS		
PROBLEM PARAMETERS (INPUT)		
	Case 1	Case 2
• No. of Antennas	12	12
• No. of Visibilities	617	933
• Request Data:		
– No. of Requests	291	292
– No. of Req. Segments	1499	933
– No. of Flight Req.	240	245
– No. of Non-Flight Req.	51	47
• Time Unit Granularity	1 Min.	1 Min.
OUTPUTS		
• Number of Requests Scheduled	286	289
• Percent of Requests Scheduled	98	99
COMPUTER RUN TIME		
• Processing Time Using MVS-APL on IBM 3081 K32	133 sec.	62 sec.
In each case a 24-hour scheduling problem was solved in less than 3 minutes.		

Figure 6. CTS Algorithm Applied to Sample Problems



For a 300 request problem the function shows an "S" curve behavior that represents considerably less computer time for the first hundred requests than for the next hundred requests and a little less time for the final hundred requests.

Figure 7. Computer Running Time in Seconds vs. Requests Schedules (MVS-APL on IBM-3081

This algorithm has not yet been implemented in the operational Air Force Satellite Control Network System. It promises to be capable of handling scheduling problems with a high degree of efficiency and flexibility.

REFERENCES

1. IBM-FSD Data System Modernization: System Scheduling (AT13), Document No. C27-020-13-01, 16 April 1980.

2. IBM-FSD Data System Modernization, Generic Range Scheduling Functional Analysis, Contract F04690-81-C-003, 10 October 1982.

3. IBM-FSD Range Scheduling Automation, 2G59 IR&D Report, 1982.

4. IBM Mathematical Programming System Extended-Mixed Integer Programming Reference Manual, SH19-1099, August 1981.

5. IBM Mathematical Programming System Extended-Control Language SH19-1147, October 1978.

ADVANCED DECISION AIDING TECHNIQUES APPLICABLE TO SPACE

Robert J. Kruchten, Major, USAF
Advanced Technologies Program Manager
Rome Air Development Center (RADC/COAD)
Griffiss Air Force Base, New York 13441-5700

1. ABSTRACT: The Command and Control Directorate of RADC has had an intensive program to show the feasibility of applying advanced technology to Air Force decision aiding situations. Some aspects of the program, such as Satellite Autonomy, are directly applicable to space systems. Other parts of the program, while not directed toward space applications, have developed techniques which could be used in space applications. For example, RADC has shown the feasibility of decision aids that combine the advantages of laser disks and computer generated graphics; decision aids that interface object-oriented programs with expert systems; decision aids that solve path optimization problems; etc. The purpose of this paper is to review some of the key techniques that could be used in space applications. It reviews current applications, their advantages and disadvantages, and gives examples of possible space applications. The emphasis is to share RADC experience in Decision Aiding techniques.

2. INTRODUCTION:
RADC has undertaken a major effort in the area of decision aid development. As part of this effort, RADC has developed a variety of decision aids for specific problems. These aids were meant to prove the feasibility of using high and low technology techniques to improve the decision process. In addition, RADC has extensively studied the problem of satellite autonomy through both in-house research and through contracted efforts. The techniques used in all these programs ranged from relatively simple operations research techniques (eg multi-attribute utility analysis, etc) to advanced artificial intelligence techniques (eg model-based reasoning, etc). Interestingly, it was often found that the inclusion of a known technique into a decision aid offered benefits far beyond the immediate problem being addressed. Unfortunately, it was sometimes also found that there were subtle disadvantages to some of these techniques. Often the benefits and disadvantages remain hidden until evaluation by the prospective user. The remainder of this paper examines some techniques that have been especially useful.

3. ADVANCED DECISION AID TECHNIQUES:

3.1. Object-Oriented Programming

3.1.1. DESCRIPTION

Object-oriented programming is a method of programming that associates code with real world objects. The individual objects are things that have behavioral characteristics. These characteristics are coded in the form of methods and procedures. The behavioral characteristics are invoked by sending messages to the objects. For example, a variety of objects may have procedures to move. A message "move" (with appropriate arguments) sent to one of the objects will automatically invoke the proper procedures to move that particular object. The sender of the message does not have to have any knowledge about how the move is accomplished. A key feature of object-oriented programming is that individual objects can inherit characteristics from other objects. Programs written in object-oriented programming are generally easy to create, extremely easy to change, and dramatically reduce memory requirements. RADC has a number of programs using object-oriented programming some of which are described below.

One RADC program uses objects for map representations. Normally a digital map data base will indicate what features are located at each location. Thus if the system is at a location that is the intersection of two roads, it will know

the roads it is on. A person looking at the map could tell basically where the roads went, what they connected with, distances, etc. Unfortunately, the computer would have to find the answer to these questions via exhaustive searches. RADC has a program that captures key map features as objects. Each object knows about itself and has procedures to determine connections with other objects and to answer questions about itself. In this system, when the computer is at a road intersection, it can send messages to each road object. It is thus possible to immediately determine which road goes to what city, the distances involved, and the route of travel.

Another RADC program uses object-oriented programming to find the best path for an aircraft through a dense series of ground to air threats. In this program, the threats are "smart" objects that know their capabilities against a variety of different aircraft and, more importantly, know the best means for an aircraft to avoid the threat. As an aircraft approaches the threat, it sends a message to the threat requesting information on paths to avoid the threat. The message contains arguments giving the aircraft type and capabilities. The threat returns alternative paths, their lethality, and their distance. The aircraft can use this information to select the best path.

Finally RADC has an object-oriented program that projects enemy activity into the future. It has various enemy units defined as objects that have behavior characteristics similar to real enemy units. These characteristics allow the units to move about within an area, given an objective. Each unit knows how to find its own path to its objective. It knows its rate of travel over various terrain, the impact of other units using the same roads, etc. All units are given objectives and then the system is queried to show unit locations over time. This system interacts with a rules-based mission planner to plan aircraft missions. It allows the planner to plan based on realistic projections of where the enemy will be.

3.1.2. Advantages

Another advantage is that an object-oriented program tends to be very compact. For example, a complex simulation using object-oriented programming could be a factor of ten or more smaller than a similar program using traditional techniques.

Another potential advantage of the object-oriented concept is that it will be possible to combine it with parallel processing techniques. To date, this has been done in only a most rudimentary fashion, but it appears to be feasible to hand off separate object processes to separate processors.

Object-oriented programming allows the developer to use more natural thought processes to visualize the problem and develop a program approach. This is possible because it releases him from many of the mundane tasks of linking together the pieces of an object.

Finally, the object-oriented system is easy to code and maintain. Changes made to an object are automatically propagated to all the appropriate instances of the object. Code is easier to write because the problem is broken down into smaller, more reasonable pieces. Thus, overall it is quicker to develop and get running than more conventional methods of programming.

3.1.3. Disadvantages

The primary disadvantages of object-oriented programming is the lack of robust language implementations to support it. The primary work is in ZETALISP FLAVORS, ZEROX LOOPS, and SMALLTALK. Although these implementations are very good, they do not fully implement the concepts. Other, more common languages offer only minimal support to this concept.

Another disadvantage is that it is difficult to fully predict the runtime performance of an object-oriented system. The problem occurs because of the very advantages of the approach. Object-oriented programming greatly simplifies the developer's tasks and allow him to actually build a program that exceeds the capabilities of his processor to execute in real time.

3.1.4. Space Applications

Object-oriented programming combining with some research in truth maintenance systems (TMS) offers hope of resolving unanticipated anomalies. The traditional approach to anomaly resolution is to define the anticipated anomalies, their indications, and the proper corrective actions. Unfortunately, most complex anomalies are unanticipated (we tend to

design out the ones we anticipate). However, using object-oriented programming to create multiple models of a system (thermal, electrical, structural, etc); we define how the system should work. When unexpected situations arise, it will possible to use TMS concepts to reason about the differences between the model predictions and the actual data thus defining the problem. Goal directed search techniques can then be used to generate a solution and the models can be used to test the solutions.

One key application of the object-oriented system is that it can be used to create a highly interactive simulation of any system. Most systems can be broken down into separate modules. These modules can, in turn, be further broken down into more detailed modules. At lower levels there is much in common between the modules. Object-oriented programming allows for generic objects (modules) to be created. Instances of these modules can then be easily made and interconnected. They can inherit characteristics as needed. The end result is a software simulation of a system. Object-oriented programming is ideal for large interactive simulations. Its modular construction allows a complex system to be rapidly built and de-bugged. In fact, it is possible to create objects that could be used in multiple space programs. A related application is to allow a satellite configuration to be designated by an interactive object-oriented model. As the configuration changes (consumable, position, failures, etc), it is easy to modify the model and thus system performance can be projected at any time.

A related application to the simulation described above is simulation used to support systems engineering design studies and tradeoffs. Complex architectures can be built using object-oriented techniques relatively easily. These can serve as system "breadboards" to check on overall system performance and design tradeoffs.The simulations themselves are not meant to be complete or to have perfect fidelity. The key is to build the simulations early in the design process in order to support the design.

Another application is in display technology. A circuit that is built and displayed as an object can be easily designated by an operator. This would give him immediate access to all relevant information on that circuit. Object-oriented representations can also be used in search routines by a computer. Thus a complex network of objects could be searched. When the system is at any particular node in the net it has the immediate and automatic capability to query that node.

3.2. Natural Language

3.2.1. Description

Traditional man-machine interface for software has been menu driven, special function keys, touch-screen, mouse/track-ball, or rigid syntax typed input. There has been much research into a more "natural" form of input. Natural Language is the machine analysis of typed sentences. That is, the transformation of a sentence into a machine usable form. RADC has developed systems that have shown the feasibility of using Natural Language as a way of communicating between the man and the machine. These systems allow the machine to carry on a dialog with a person even though the individual sentences contain ambiguity. They do this by establishing a loose context relationship between the sentences being input, previous sentences, previous machine responses, and knowledge of the domain. The net result is a relatively free flowing, conversational type dialog.

3.2.2. Advantages

Natural Language allows an operator with virtually no training to be immediately effective. Syntactic errors are virtually eliminated and no pre-defined formats are required to be memorized.

A secondary advantage of the Natural Language research is its development of solutions to problems that must be evaluated in context. Since most of our sentences and words can only be understood in context, Natural Language researchers were forced to develop techniques for establishing context relationships. Those techniques can be fruitfully used in other applications as defined below.

3.2.3. Disadvantages

Natural Language involves both input and output. As an output medium, it allows for unanticipated, machine generated statements or queries. As the primary means of machine input, Natural Language is far from ideal. Most operators are

not skilled typists and cannot input sentences quickly or accurately. In addition, if the task is repetitive, a person will be much faster and accurate with standardized input or output schemes.

3.2.4. Space Applications

One obvious application of Natural Language is for man-machine interfaces. Here it has the advantages and disadvantages described above. A not so obvious application is in data interpretation. Any one data channel or sensor output can only be interpreted in context of all other data, past history, and knowledge of the domain (eg. what happens during an eclipse, etc). Technically, data interpretation has many of the same problems as language interpretation. Natural Language techniques allow loose context relationships like those in involving telemetry to be created.

3.3. Video Disk Displays

3.3.1. Description

Most graphic displays are generated using digital data that is either vector plotted or bitblt'd to fill a screen. In the first case, graphics draw commands are used to draw an image using a digital data base that defines a series of vectors to be drawn. In the second case, the image is created from a large array representing the individual pixels to be drawn. Bitblt has the advantage of creating a display very quickly, but it requires large amounts of memory if there are many different displays to be created. Vector plotting is more common because it requires less memory, but it is usually quite slow to create an individual image. In addition, complex images created by a vector plot also involve large amounts of data. Normally resolution of the image is sacrificed to reduce the data overhead. The impact is that curved lines appear as stepped lines and detail is eliminated from the image.

RADC has recently developed several tactical planning aids that required tactical maps to be displayed with icons superimposed on the maps. The technique used was to combine a commercial laser disk image with computer generated icons. The laser disk is a standard commercial NTSC laser disk with each frame containing an image of a different map (or a different scale). Laser disks represent an extremely dense medium and contain up to 108,000 separate NTSC images on a single disk. It is possible to select any one frame in less than one second. The laser disk image is essentially anything that could be displayed by a commercial television (graphics, photo, etc). As implemented, the laser disk image is displayed on a high resolution screen with NTSC resolution. The computer generated graphics are displayed with high resolution on the same screen. Thus, it is possible to rapidly display many complex images using an inexpensive and small system (our system used an IBM AT).

3.3.2. Advantages

The primary advantage of the laser disk system is many, very high quality images are available on a relatively small, inexpensive system. Computer generated imaging is still available and simply overlays the video disk image. The commercial hardware for this system has hardware zoom and pan capabilities that operate much the same as hardware zoom and pan on bitblt or vector created images. However, the laser disk system has the additional capability of selecting another image instead of expanding or moving the existing image. The new image selected by the laser disk system would still retain a high resolution whereas zoom of a bitblt or vector created image loses resolution after a factor of 4-6 times. For map applications, this means using hardware zoom up to a factor of about 4; then selecting another map with a different scale.

Another advantage of the video disk is that the storage media is radiation hard and easily removable. When removed from the drive system, The individual disks are easy to store and relatively indestructible.

A final advantage of the video disk system is the high level of user acceptance. The systems are very user friendly and, most importantly, they provide images similar to the ones currently used by the user. For RADC's tactical programs, this means map images identical to those carried by the pilot. For other applications, it could mean schematics or illustrations found in other references.

3.3.3. Disadvantages

The primary disadvantage of the laser disk system is the laser image is

unavailable to the computer. Thus, a line on a map representing a road is unknown to the computer. This disadvantage can be somewhat alleviated by having a separate digital representation for the computer. Whereas the operator may see a nice, smooth curve for a line; the computer may use a relatively crude jagged representation of the same feature. This requires many tradeoffs to be made between operator needs/desires versus the computer needs for maximizing performance and memory requirements.

Another disadvantage with the laser disk system is the relatively high cost of first disk. The first disk for an RADC system cost approximately $50K for 34K images. Second and later disks are considerably cheaper ($600).

Finally, this system requires the laser images to be static images. The disk represents a read-only medium and cannot currently be used for applications requiring real-time changes to the images.

3.3.4. Space Applications

The laser disk system is appropriate for any application involving many unchanging images. It could easily be integrated into a diagnostic system involving logic diagrams, illustrated parts breakdowns, photographs, etc. It can also be used for high resolution background displays.

3.4. Expert (Rule-Based) Systems

3.4.1. Description

A rule-based expert system normally has a knowledge base of facts and if-then rules plus an inference engine to make inferences. A key feature of this type system is that the system developer concerns himself with capturing the knowledge rather than the details of the inference mechanism. Rule-based systems are the most common type of expert system being developed today. Most of RADC's rule-based systems either use a rule-base as part of a larger system or are primary concerned with assessing information.

3.4.2. Advantages

Rule-based expert systems are relatively easy to construct, very easy to modify/maintain and can handle incomplete, ambiguous, or conflicting data. They are unique in their ability to capture high level human thought processes. That is they easily capture rules of thumb (heuristics). More traditional systems can perform similar functions, but they require the persons thought process to be abstracted into a more acceptable form for the machine.

3.4.3. Disadvantages

Rule-based systems are not good for all applications. They often tend to be slow and require relatively large memories. Most importantly, they sometimes give wrong answers. In traditional programming, the goal is to build a system that meets some performance requirements. The expert system tries to mimic the expert. Like the expert, it makes mistakes. This is especially true near the boundary area of its knowledge. These are problems that match part of its rules, but also include facts beyond its knowledge base. In these areas, the rule-based system may give misleading and wrong answers whereas the more traditional system would crash.

Another problem with a rule-based system is in expecting them to solve problems requiring very basic knowledge. Actually they perform at their best by helping raise the performance level of a decision maker. These systems have not been able to capture basic (deep) knowledge that experts use to solve unique problems.

3.4.4. Space Applications

These kind of systems are ideal for removing the burden of controllers to check for common problems. They are also helpful for building scheduling systems. A key consideration in their use is whether the system will require frequent adjustment (change) by the operator (experts). The rule-based systems normally allow rules to be added or deleted by the user. Most of the experience at RADC has shown that they can raise the level of a decision makers to a new plain. Whereas before he was totally engrossed in mundane tasks, the rule-based system easied his workload so that he could concentrate on an assessment of the big picture.

3.5. Rapid Prototyping

3.5.1. Description

The last technique in this paper is not concerned with a software design approach. Instead, rapid prototyping refers to a development technique. Currently, DOD software programs are

developed using DOD STD 2167. This standard involves an orderly systems engineering design approach where the design evolves form high level system requirements to low level detailed item requirements. Along the way The requirements are documented in the various levels of specifications (A, B, C); reviewed in formal design reviews (SDR, PDR, CDR); and finally, audited (PCA FCA). Throughout this process configuration control procedures track the system and generally the requirements are very well defined before coding commences.

Knowledge based systems cannot generally be effectively designed using the above approach. The difficulty lies in capturing human, expert knowledge (including heuristics) via the specification process. A more effective way is to build a prototype of the system and iterate its design through sessions with the experts. RADC has a number of knowledge based programs that have used a rapid prototyping development approach. The approach is made possible by many advances in the software development environment that make it easy to create data structures, stubbed interfaces, etc. It is also easy to make changes, incrementally to the system under development.

3.5.2. Advantages

The primary advantage of this approach is the ready acceptance of the user. This process has the user intimately involved in the total development cycle. His critiques of the system result in immediate feedback on the design. Another iteration of the prototype provides feedback to the user. This interaction makes the user a true part of the development process and allows him to see actual performance tradeoffs as they occur.

3.5.3. Disadvantages

A key disadvantage with this approach is the absence of formal procedures. Since the approach is contrary to the accepted development techniques, there are no standards for documentation, design reviews, or testing. This makes it extremely difficult to create contract packages with measurable end products. Disagreements between user experts create difficulties in defining the prototypes. Similarly, availability of experts becomes a critical path to the overall development. If the experts cannot devote sufficient time or the experts are constantly being replaced, the system development rambles.

Tests also tend to be more ad-hoc because there are few specific requirements (no specs) to measure against. Thus, it is difficult to quantitatively measure the software developed. The weakness in the tests of knowledge based systems is probably a primary reason for the heavy emphasis on systems that can explain their results.

3.5.4. Space Applications

It would seem that some form of rapid prototyping is necessary for successful knowledge acquisition if one is building a knowledge based system. Thus if the primary consideration of the system design is capturing a human reasoning process, then rapid prototyping should be used. For these situations, the systems engineering approach tends to be incomplete. Rapid prototyping can also be used for other applications, but it should be carefully weighed against the more traditional approaches. It is a relatively new technique and has not been evaluated against all types of development projects. Long term, some combination of a systems engineering approach, for technical requirements identification, and rapid prototyping, for knowledge acquisition and feasibility proof, will be probably be best.

4. Recommendations/Conclusions:

The purpose of this paper was not to reveal some radically new and unique approaches to space related problems. Rather, it is meant to use solutions and ideas to other problems and show how they may help solve space related problems. Although research continues, most of these techniques have been proven to be very effective in other applications. Most of these solutions and ideas have been presented in the form of programming techniques. All of the software programming concepts described here require relatively large processors (in terms of memory size) compared to existing space qualified hardware. However we anticipate future on-board hardware will be capable of using this techniques. This is feasible because each of the above techniques show how new methods of programming can simplify the problems and reduce the machine requirements compared to more traditional progamming approaches. Thus, they open the way for new software applications. More importantly, they offer hope for improved system performance and reduced support costs.

AUTOMATIC ROUTING MODULE

Janice A. Malin
Systems Control Technology, Inc.
1801 Page Mill Road
Palo Alto, California 94303

ABSTRACT

Automatic Routing Module (ARM), a tool to partially automate Air Launched Cruise Missile (ALCM) routing, is installed at HQ-SAC and is used operationally by JSTPS mission planners. For any accessible launch point/target pair, ARM creates flyable routes that, within the fidelity of the models, are optimal in terms of threat avoidance, clobber avoidance, and adherence to vehicle and planning constraints.

Although highly algorithmic, ARM is an expert system in the sense that it: (1) rapidly creates plans based on heuristics, or rules, supplied by planning experts; (2) relieves planners of much of the tedious and time-consuming portions of the route planning process; (3) supports both expert and non-expert planners in route creation; (4) allows the planner to change the rule base; (5) recommends a course of action and provides the planner with a means to modify that recommendation; (6) provides a menu-driven, user friendly interface plus interactive graphics; and (7) relies on a statespace, paths, and decision tree that must be searched, complete with cost function, to arrive at an optimal route.

Because of the heuristics applied, ARM-generated routes closely resemble manually-generated routes in routine cases. In more complex cases, ARM's ability to accumulate and assess threat danger in three-dimensions and trade that danger off with the probability of ground clobber results in the safest path around or through difficult areas. The tools available prior to ARM did not provide the planner with enough information or present it in such a way that ensured he would select this safest path.

1. INTRODUCTION

Systems Control Technology (SCT) began working on basic research in optimization applied to automated planning systems in 1978 under DARPA sponsorship. The motivation for this work was to:

1. reduce planning time and manpower requirements,
2. improve planning effectiveness,
3. produce timely responses to scenario changes, and
4. use existing technologies to solve the problem.

These efforts resulted in a basic approach to planning and a demonstration model referred to as AUTOPATH. AUTOPATH, as outlined in Figure 1, has broad applicability to a variety of mission planning problems. It has already been successfully used in cruise missile routing, force level planning, unit level planning, asset allocation, and an on-board processing experiment.

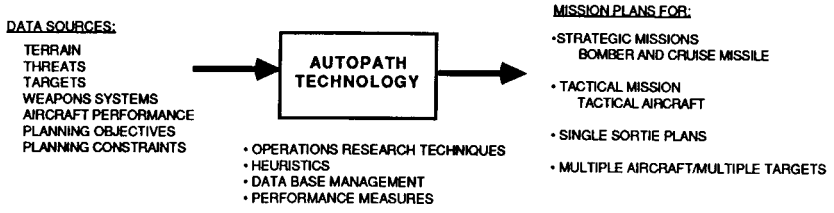


Figure 1 The AUTOPATH Technology Operates on Multiple Data Sources with Sophisticated Mathematical Algorithms and Optimization Techniques to Generate a Variety of High Level Mission Plans

This paper focuses on SCT's experience in using this basic technology to develop an automated Air Launched Cruise Missile (ALCM) routing tool for the Joint Strategic Target Planning Staff (JSTPS) at Offutt AFB. The JSTPS is colocated with Headquarters, Strategic Air Command (HQ SAC). The AUTOPATH research and development efforts laid the groundwork for a prototype system that was installed at HQ SAC in 1984. The Automatic Routing Module (ARM) that grew out of the prototype was installed in 1985. The transformation from research and development to an operational system was a complete success; ARM is used operationally at HQ-SAC to aid in the ALCM planning portion of the SIOP. ARM is a major software module. This paper is intended to briefly highlight certain aspects of the ARM software and development experience rather than to cover any aspect of ARM in detail. In that light, the topics covered are an overview of the project and its goals; the basic technology used to solve the problem; the implementation of the solution; and lessons learned from the project.

2. ARM OVERVIEW

The ALCM planning problem has many complex parts, all of which are labor and/or computer intensive processes. Computer processing plays a major role in the planning, primarily in creating and managing data bases and in analyzing and evaluating missions that were generated by expert human planners. SCT set out to automate the mission generation aspect of planning beginning in 1983.

2.1 The Planning Problem

The mission generation planning problem is to create a flyable route between a launch point and a target. This problem is easily stated but not easily solved. The ALCM is a long-range, terrain-following missile. It must follow a programmed path that guides it from launch to the target while ensuring that it does not run into any high ground (i.e., clobber). Using the terrain following capability, the missile's path must also avoid flying into enemy defenses or risk being destroyed before reaching the target. Avoiding clobber and enemy defenses along a single flight path is a difficult task that is compounded by a growing navigation error as the missile progresses to the target. To keep this error under control, the Inertial Navigation System requires periodic updates from a finite number of highly surveyed areas of terrain known as TERCOMs. Therefore, the final route must take advantage of the available TERCOMs and still avoid ground clobber and enemy defenses.

A human planner uses detailed maps which include the defenses and TERCOMs to perform this routing task. The planner has to determine the best path through the defenses and the best altitude for each leg of the flight. This is a gross oversimplication of the task because of many other considerations and constraints, some of which will be discussed later. The planner's task becomes very difficult in heavily defended areas and is further complicated by the number of missions to be planned.

Once satisfied with a mission, the planner provides the mission's flight plan to very detailed evaluation software which uses accurate vehicle performance models, local terrain, and detailed threat models to determine how good the path really is. The software can make some modifications to the path, but generally flags errors for the planner to rethink, fix, and resubmit the route for analysis. Because of the fidelity of the models, this software has been slow, making good candidate routes on the first pass highly desirable.

The objective of the ARM program is to create good quality candidate routes and to do so in a timely manner, i.e., less than ten seconds each. By meeting this objective, ARM relieved the mission planners of much of the tedious and time-consuming portions of the route planning process. It freed them to spend more time on difficult routes that do not follow all the rules given to the automated system, to fine tune routes, or to work other problems.

ARM can be given one or many missions, described by launch point/target pairs, to plan. For each mission, ARM creates a flyable route that is optimal in terms of threat avoidance, clobber avoidance, and adherence to routing constraints. Routing constraints include vehicle performance characteristics and heuristics, or rules of thumb, that a mission planner would apply if the route were to be created manually. Some of these rules are given in Table I.

Because of the heuristics applied, ARM-generated routes closely resemble manually-generated routes in routine cases. Manually created routes and ARM generated routes may diverge in more complex cases. ARM's use of three-dimensional threat models allows the system to rapidly determine the safest path around or through areas of high danger. This ability to use a deterministic approach rather than to rely on the human eye and brain to sort through the myriad of possibilties results in a route of equal or better quality than a manually generated route. Obviously, these complex cases are the ones for which ARM saves the most human planner time.

Table I ARM Rule Base Examples

VEHICLE CONSTRAINTS	PLANNING RULES	PLANNING HEURISTICS
• climb/dive rates	• terrain avoidance	• minimum fuel reserve
• turn radius	• threat avoidance	• minimum/maximum number of TERCOMS per route
• speed	• avoidance areas	
• maximum fuel load		• minimum/maximum distance between route segmentation points
• Warhead Arming Maneuver		
• Inertial Navigation System		• minimum acceptable target damage
• update frequency		• route dispersion
		• target avoidance
		• maximum distance between TERCOMs
		• maximum accumulated danger

2.2 Man-in-the-Loop

While ARM is a success, automation in mission planning has not reached the stage where the man-in-the-loop is not required. As one would expect, ARM requires a considerable amount of external data. Target, launch point, TERCOM, and defense data are just some of the data that must be provided and verified. Nominally this is the job of one person, the Mission Controller/Data Base Manager. In addition, all of the rules must be provided. These are saved from session to session, but can be changed as desired.

ARM provides the capability to review the routes and all intermediate information. The planner module is completely interactive and supports full, high-resolution, color graphics displays of the scenario, including latitude/longitude grid, launch points, targets, TERCOMs, and defenses. One very useful display feature that is provided and that is missing from planning maps is altitude-dependent danger contours for clobber danger, enemy defenses, or combined danger plus total danger contours at the optimal altitude for each statespace cell. With these displays, the planner has a much better understanding of why ARM chose a certain threat penetration, for example.

The graphics module also allows the planner to rapidly change the recommended route and reevaluate it. The planner has full control over changes to TERCOM selection, navigation point placement, leg-by-leg altitude and turn radius selection, as well as launch point and target specification.

2.3 ARM As An Expert System

ARM is not a true expert system as that term has come to be accepted today. ARM is an expert system in the sense that it:

1. creates plans based on rules supplied by planning experts,
2. supports the planner in his task,
3. allows the planner to change its rules,
4. recommends a course of action and provides the planner with a means to modify that recommendation,
5. provides a user-friendly, menu-driven interface plus a graphics display to support the planning process and understanding of the problem and its solution, and
6. relies on a statespace, paths, and decision tree that must be searched, complete with cost function, to arrive at an optimal route.

ARM is not a standard expert system in the sense that it is written in FORTRAN 77 rather than in a symbolic language, is highly algorithmic, and is data and I/O intensive.

2.4 Summary

The ARM system is largely a parameter or data driven system that provides mission planners with an effective tool for generating good quality candidate routes to be input to the detailed evaluation

programs. ARM further provides the capability to support real-time, rapid-strike route generation as well as studies of proposed vehicle modifications, routing logic alternatives, threat analyses, modeling alternatives, and various "what-if" hypotheses.

3. ARM TECHNOLOGY

The SCT AUTOPATH approach is based on decomposing the overall problem into several smaller, manageable pieces. The intent is to define functionally self-contained modules that are computationally practical.

Figure 2 provides a first level decomposition of the SCT AUTOPATH mission planning approach. The five main steps pictured are described in more detail below, as they relate to the cruise missile case.

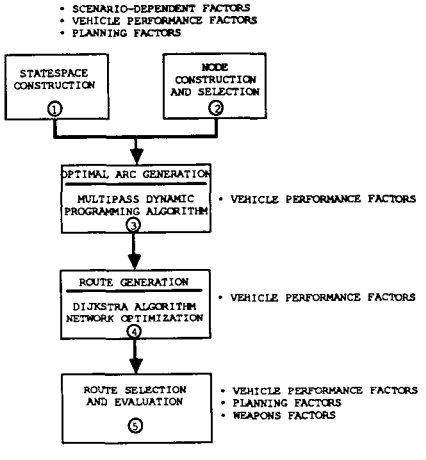


Figure 2 ARM Incorporates Many Military Planning Factors

Step 1: Statespace Construction: The statespace reflects a three-dimensional quantization of the geographic area of interest. The geographic area is divided into latitude-longitude cells with discrete altitude levels. Associated with each cell, and with each of the eight basic directions of traveling through the cell, is the 'cost' of traversing the cell. This cost is based primarily on the danger from enemy defenses; however, any number of other cost factors can be considered. For the cruise missile, it is important to consider the probability of the missile running into the ground. Thus, ARM also uses models that predict 'ground clobber' probability as a function of the vehicle's altitude and the terrain roughness. The terrain roughness is extracted from Digital Terrain Elevation Data (DTED). Danger from the enemy defenses are incorporated into the statespace through the cross-range and down-range threat models (or threat 'templates'). These templates are used to estimate the danger to the vehicle as a function of the distance and orientation from a particular threat type. An example of such a template is given in Figure 3.

Step 2: 'Node' Construction: In most mission planning applications, the vehicle has a number of routing constraints that have to be met. Most of these can be modeled by prescribing various geographic points, or nodes, through which (or through some of which) the vehicle must travel. In ARM, the nodes are launch points, TERCOMs, and targets. A complete mission can be represented by a sequence of nodes. The objective of the node construction step is to define a network that describes all possible pairs of nodes between which an ALCM could potentially travel in the given scenario. It is important to minimize the size of the this network. Thus, as many constraints as possible are considered. For example, if navigation updates are required every 'x' miles, the connections longer than this are not included in the network. Quotas of accessible nodes are also effective. These constraints are either hard constraints dictated by the vehicle or logical heuristics that are key to planning for a specific system. These constraints are defined during the 'knowledge acquisition' or requirements analysis phase of the effort.

Step 3: Route Segment Generation: The purpose of this step is to compute optimum route segments, or arcs, between each of the node pairs in the node network. For each node pair, one node is the origin node and the other is the destination node. For each destination node, a multipass dynamic programming algorithm (MDPA), is executed on a subset of the statespace that contains the node pair. The MDPA computes an optimum set of controls (directions of travel) for each cell in the statespace to the

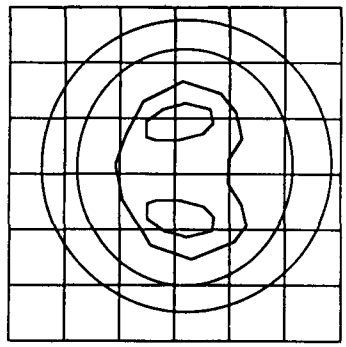


Figure 3 Sample Threat Template for a Left-to-Right Heading

destination node. Following the MDPA, the route retrieval algorithm is used to extract the optimum route segment that connects the origin node to the destination node. The route retrieval algorithms also perform whatever route smoothing is required. Once the route segment has been extracted, the total transit cost between the nodes is known and is stored for further processing.

Step 4: Route Generation: The next step in the process is to link these route segments into nearly complete routes, or paths. To preserve flexibility and to reduce the number of paths, the launch points and targets are not linked to the routes during this step. The basic algorithm used is the Dijkstra shortest path algorithm which provides a very efficient way to extract the set of route segments that produces a minimum cost route between an initial and terminal TERCOM node pair. Since each initial TERCOM can ultimately reach many other TERCOMs, the Dijkstra algorithm actually results in a tree for each initial TERCOM containing the best paths from the TERCOM to every other TERCOM it can possibly reach. The transit cost for each path of each tree is known and preserved.

Step 5: Route Selection and Evaluation: In ARM, the starting nodes in the Dijkstra search set are TERCOMs accessible to launch points and the terminal nodes are TERCOMs accessible to targets. The number of paths that can connect a launch point/target pair is now a manageable number with the transit cost of each readily available. Working backward from the target, it is a simple matter to select the tree and path that, together with the cost of traveling from the launch point to the tree and from the tree to the target, optimizes either the probability of arrival (Pa) or the probability of Damage (Pd) for the launch point/target pair. In doing so, ARM evaluates all potential routes it investigates against the planning criteria, but saves only the route description and evaluaton for the route it selects. Figure 4 shows a very simplistic example of one tree and its accessible launch points and targets.

This metnod of attaching the launch points and targets as the last step provides the capability to quickly link different launch points or targets to a validated path.

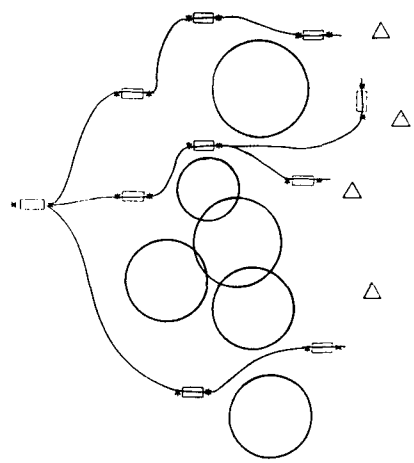


Figure 4 Sample Route Selection Options

4. IMPLEMENTATION CONSIDERATIONS

Because of the rule base and decision logic used in ARM, one may expect that it is written in a symbolic language. It is not. ARM is implemented in FORTRAN 77 and installed on an IBM 3090-200 and VAX 11/780. The IBM is the primary host with the VAX used for demonstration and off-site development and maintenance. FORTRAN was chosen for a variety of reasons and, in retrospect, appears to have been a reasonable choice. The driving factors were:

- highly algorithmic problem solution,
- manipulation of up to two gigabytes of data,
- required computational and I/O speed,
- limited virtual memory on the target IBM host computer necessitating overlays or task swapping,
- lower development cost using the FORTRAN prototype as a baseline, and
- the available compilers on the host computer at the time.

All of these factors precluded selecting a symbolic language. This list of constraints is still valid four years after the initial choice was made. Certainly FORTRAN is not the only answer for the near future, and technology is rapidly advancing; however, a caution must be raised here. While demonstrations written in symbolic languages may appear to solve the problem at hand, constraints such as those listed above should be evaluated before launching into the development of an operational automated system written in a symbolic language.

Once the language was chosen, core space, allocation of disk space, computational speed, and I/O speed became the critical design factors. As configured, the host computer provides only about seven megabytes of virtual memory. This is an absolute physical constraint while all the others were practical requirements. The virtual memory constraint was satisfied by using code overlays, increased disk I/O and, therefore, execution time, and a small loss of modeling fidelity, e.g., larger statespace quantization.

Obviously, the solutions to the core space problem compounded the other design problems. Execution time for arc and tree generation became an issue because of the increased I/O and the sheer volume of data to be processed. There were a limited number of disk packs available, and we desired not to have a single file span two disk packs. If run unrestricted, ARM would generate arcs between each pair of nodes and create a tree for every TERCOM. This could result in hundreds of thousands of arcs and many more trees than necessary. These problems were mostly solved by the heuristics. For instance, minimum/maximum TERCOM separation is considered plus a quota system for determining the number of arcs that will be generated for a TERCOM. Trees need only be built for those TERCOMs accessible to a launch point. Therefore, algorithms used in the prototype had to be redesigned to be more time efficient.

A commercial data base management system might have made the design easier, but at least at that time, would not have provided the speed needed. The data base management system is written entirely in FORTRAN and is adapted to ARM's needs. Many linked lists are used rather than wasting disk space for fixed format files. With the larger, faster disks available now, that same decision might not be made today.

Portability was not a design issue, but several standards invoked on the project, mostly to promote maintainability, actually provided a reasonable basis for a portable system:

- Very few extensions of FORTRAN 77 were used;
- The only calls to IBM-specific functions are to open files and to the system clock;
- No machine specific data base management system;
- A core standard graphics package is used;
- The overlays are not embedded in the code.

Today, the software can be run under IBM MVS, VM/CMS, and MicroVAX on upward under VMS.

5. LESSONS LEARNED

The ARM project started in the Summer of 1983 resulting in a Phase 0 prototype installed in the Summer of 1984; a Phase I prototype in the Fall of 1984; and an operational system in the Fall of 1985. It is currently in a maintenance and modification mode. During these phases, there have been several hard-learned lessons that point out that no shortcuts can be taken in the development of an operational system. The main lessons were:

- Flexibility breeds user confusion and introduces a source of errors;
- Automation has an impact on operational procedures;
- Demonstration models only solve one aspect of the design problem;

- End user involvement is critical from the beginning of the project to foster total acceptance of and efficient transition to the use of the end product;
- Frequent user training is required in light of frequent user (military) turnover and program modifications;
- Acceptance testing should be conducted within the total operational environment--not as a stand-alone program;
- Redundancy in documentation is a configuration management nightmare;
- Documentation standards do not guarantee a document that leads the user through the system step-by-step.

The last five of the above problems are common to any system and need no further explanation. The first three have major implications for large automation efforts like ARM and deserve more elaboration. The need for flexibility in a large automated system cannot be denied. Flexibility in ARM is provided through parameterization. There are several hundred tuning parameters in the system that are used to ensure route quality. Some of them describe the planning heuristics, some the planning constraints, some the vehicle performance capabilities, and still others refer to the options such as terrain masking. This places a heavy burden on accurate documentation plus an overwhelming task on the selected users to review the documentation and the chosen data settings to ensure sound values. There is the added problem of determining which levels of users should have permission to change which variables.

The following steps have been implemented in ARM to help solve these problems:

- optional range of value check on each numeric variable, e.g., $0 < Ps < 1.0$,
- value check on monotonic sequences,
- value checks on some interrelated variables,
- user access levels for each variable,
- user write-access levels for each file, and
- common on-screen or off-line data base update capability.

Yet, none of these tools, nor extensions of them, relieve the primary data base manager from having to review the data base for consistency. What is needed for ARM and systems of its kind are front end systems that will review the data base and explain to the user the consequences of his composite data base. Such a system should be directed at individual modules and at the system as a whole.

Operational procedures are invariably altered by the introduction of a new tool that automates a portion of the procedures. The tool automates a process, but it in turn requires support (e.g., data base preparation, program monitoring, computer graphics replacing paper maps and drawing, additional emphasis on other processes). Hopefully, this does not take more time than the original process that has been automated! This change in procedures needs to be well thought out and new roles assigned to the staff who will use the tool well in advance of its introduction.

The final points are directed at demonstration models. A demonstration model may show a solution to be feasible but in fact the solution may not be practical when all the real world constraints are applied. In ARM's case, the enormous data bases and core space limitations were unknown when the model was built and were not considered in the model. Many of the algorithms used had to be modified to support these constraints. Thus, it is not necessarily true that the algorithms used for a prototype can or should be applied to a system requiring an operational data base many orders of magnitude larger than the test case. For other prototypes, the lesson is that these problems should be determined before the hardware is selected. In the ARM case, that was not an option. Fortunately, the ARM software was ultimately ported to a much more powerful IBM computer than the original target machine. This allowed the intermediate data base to be generated in about one day. This equates to a saving of months of human effort for each planning cycle.

A further problem with demonstration models for brand-new systems is that they may only solve one part of the problem. Figure 5 shows the components that contributed to the operational version of ARM. Only the portions of the problem indicated in gray were addressed in the demonstration model. This is typical of software resulting from research and development efforts. The prevailing thought is that the operational system should not cost very much since the problem has already been solved. The fact is that these other areas are equally important and become a driving cost factor.

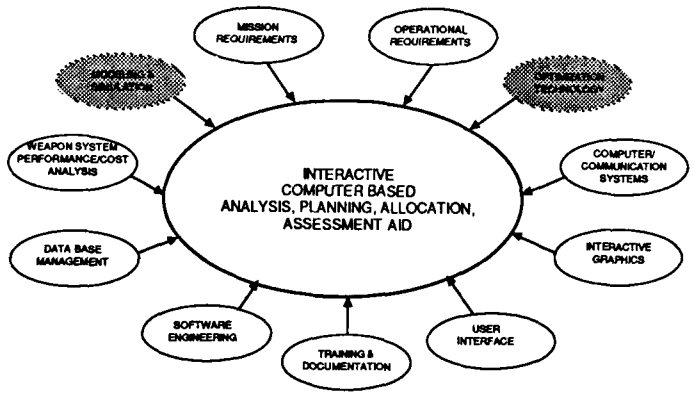


Figure 5 Successful Application of Optimization Requires More Than an Understanding of the Algorithms

6. CONCLUSION

ARM is a powerful and extremely flexible tool to support ALCM planning. The ARM project has shown that, with ingenuity, it is possible to automate a large problem. ARM has effectively automated the generation of candidate ALCM missions freeing the mission planners from tedious and time-consuming problems to plan more efficiently and concentrate on more difficult problems.

ACKNOWLEDGEMENTS

I would like to thank everyone on the ARM team from SCT, JSTPS, SAC, and DARPA who believed in the project and worked so hard to make ARM the success that it is today. Steve Rainbolt, Milt Grossberg, Jack Murphy, Kemp Deicke, John Russell, Sandy Henninger, Carol Tycko, John Mordeson, Tim Croll, Rich Vargus, Ellen Rubin, and Tina Youngs from SCT and Maj. Dale Richter, Maj. Clif Banner, Lt.Col. Dave Enos, and Maj. Fred Guice from SAC-JSTPS deserve special recognition for their roles in developing, guiding, and maintaining the ARM system.

REFERENCES

1. "User's Manual, 1986 Automatic Routing Module (ARM) Maintenance and Enhancement Software," B008, Systems Control Technology Inc., Palo Alto, California, September, 1986.
2. "Data Base Specification, 1986 Automatic Routing Module (ARM) Maintenance and Enhancement Software," B021, Systems Control Technology Inc., Palo Alto, California, September, 1986.
3. "System/Subsystem Specification, Program Specification, and Maintenance Manual, 1986 Automatic Routing Module (ARM) Maintenance and Enhancement Software," B016, Systems Control Technology Inc., Palo Alto, California, September, 1986.
4. "Software Interface Control Document, Volume I, 1986 Automatic Routing Module (ARM) Maintenance and Enhancement," B012, Systems Control Technology Inc., Palo Alto, California, September, 1986.

ROBOTIC AIR VEHICLE

Blending Artificial Intelligence with Conventional Software

Christa McNulty Joyce Graham Paul Roewer

Texas Instruments Incorporated
P.O. Box 660246, MS 238
Dallas, Texas 75266

1 ABSTRACT

This paper describes the Robotic Air Vehicle system sponsored by both the Defense Advanced Research Projects Agency (DARPA) and Air Force Wright Aeronautical Laboratories (AFWAL). The program's objective is to design, implement, and demonstrate cooperating expert systems for piloting robotic air vehicles. The development of this system merges conventional programming used in passive navigation with Artificial Intelligence techniques such as voice recognition, spatial reasoning, and expert systems. The individual components of the RAV system are discussed as well as their interactions with each other and how they operate as a system.

2 INTRODUCTION

Challenging modern air defenses poses significant dangers such as loss of crew and aircraft. Intelligent unmanned flight systems can provide a viable solution to eliminate the loss of high-value aircraft and complement our manned force. The technology to allow the intelligence and adaptability of a pilot to be added to an unmanned flight system is being developed on the Robotic Air Vehicle (RAV) program. The RAV contract was awarded in September of 1985 by the Defense Advanced Research Projects Agency (DARPA) and Air Force Wright Aeronautical Laboratories (AFWAL). The program's goals are to design, implement, and demonstrate cooperating expert systems for piloting robotic air vehicles. The approach being used by Texas Instruments is to combine conventional programming with Artificial Intelligence (AI) techniques. This approach leverages established technologies, such as control theory and navigational terrain algorithms, with more recent techniques such as expert systems that give the RAV system the ability to plan, execute, and alter its mission. The mission scenario addressed in this paper is the reconnaissance of heavily defended areas.

The AI techniques applied in the RAV system are natural language understanding, voice recognition, expert systems, and spatial databases. The conventional software systems in the RAV system are the aircraft simulation, passive navigation, and the terrain following/ terrain avoidance route planner.

The following section, Robotic Air Vehicle System Software Architecture, explains the individual systems, both AI and conventional, that compose the RAV system. The RAV Hardware section describes the types of computer systems and their configuration required to run the entire RAV system. The Reconnaissance Mission Example section illustrates the interaction of various RAV system components executing during a reconnaissance mission.

3 ROBOTIC AIR VEHICLE SYSTEM SOFTWARE ARCHITECTURE

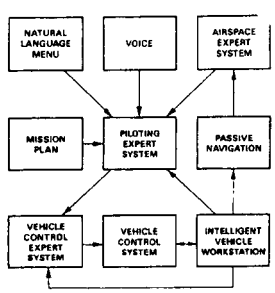


Figure 1. RAV SYSTEM SOFTWARE ARCHITECTURE

3.1 Command Inputs

The Robotic Air Vehicle system has two methods for command input: a natural language interface and a voice recognition system. During system execution, either input method can be used provided the user completes a command with one method before changing to the other method to enter the next command.

A natural language interface is provided to the expert systems through Texas Instruments NLMenu (TM). NLMenu is a menu-based approach to natural language understanding. The user is presented with a series of menus from which valid commands can be built. The menus are semantically constrained by a defined grammar so that only valid phrases within a context can be generated. The TI-DARPA fighter pilot grammar used in this interface is a set of in-flight commands used by pilots in general that has been tailored to the characteristics of high performance aircraft. This interface establishes an easy means to view the valid commands, to enter only complete commands, and to quickly enter commands for running the RAV system during knowledge engineering sessions with piloting experts.

Voice recognition technology is applied to the RAV system to enable communication with air traffic control or other manned aircraft and to receive intel and weather updates. The same TI-DARPA fighter pilot grammar used for the NLMenu system is also used in the grammar for the Voice system. The voice recognition in the RAV system uses the connected word recognition technology developed under the DARPA Robust Speech contract. This technology is currently implemented on the Texas Instruments Odyssey signal processing board.

3.2 Mission Plan

A route planner generates the mission plan using a conventional algorithmic approach. Given both a starting and ending point, the route planner finds an optimal path between these points. The mission plan consists of a series of intermediate waypoints, headings to those waypoints, and various altitudes to maneuver along the terrain following, flying close to the ground and terrain avoidance, flying around mountains instead of over them, path. This plan is used as input to the Piloting Expert System for navigation during the ingress, at-target, and egress phases of a mission.

3.3 Passive Navigation

For covert missions, the RAV system uses passive ranging of distant ground objects to navigate. Passive ranging relies only on passive sensors such as video and Foward Looking Infra-Red radar (FLIR) for its inputs. The Passive Navigation System uses these inputs in conjunction with data from a digital map to calculate algorithmically the current aircraft position with respect to the terrain. This position is sent to the Airspace Expert System where it is then distributed to the other RAV systems as needed.

3.4 Airspace Expert System

The Airspace Expert System (AES) provides situational awareness to the Piloting Expert System about any airspace object whose sphere of influence encompasses the aircraft. In a three-dimensional airspace, these objects can include airports, tactical aids to navigation, jet routes, circumference of missile sites, and other aircraft.

The AES is a multi-layered system with increasing layers of functionality. At the core is a spatial database built using the TI Relational Table Management System. The spatial database contains information essential for navigation such as aeronautical charts, instrument approach plates, and airport directory information. Residing on top of the spatial database layer is an intelligent query translator which provides access to information from different database sources without the user having to know the database structure. A computational layer is used to perform basic navigational computations such as bearing, range, or time to a specified point. The highest layer contains expert reasoning capabilities used to give early warning notifications of airspace boundaries or threats and to respond to such queries as locate a suitable divert base.

3.5 Piloting Expert System

The Piloting Expert System (PES) has the capability to perform both piloting and navigational tasks. The PES contains the knowledge to execute mission plans which include takeoff, standard instrument departure, navigation to an IP and target, egress, final approach and landing. The PES is initially activated by receiving

commands from NLMenu or Voice. The PES accomplishes a command or series of commands by executing the appropriate plans in the knowledge base and doing any combinations of the following: executing the plan from the route planner, retrieving spatial knowledge from the Airspace Expert System, or issuing commands to the Vehicle Control Expert System or Vehicle Control System.

The PES contains multiple knowledge bases such as takeoffs, departures, navigation, holding patterns, approaches, and landings. These knowledge bases are written in the TI Dallas Inference Engine and Inference Corporation's Automated Reasoning Tool (TIDIE/ART) knowledge representation. Briefly, the TIDIE/ART representation consists of three main components, OBJECTS which represent aircraft state variables (e.g. airspeed, altitude, etc.); NEEDS which designate what task is needed (e.g. departure-climb); and PLANS which are how that need or designated task is to be performed (e.g. intercept-inbound-plan). Within the plans are steps which can be either event or time driven conditions to be met before continuing to the next step or actions in the form of directives to the Vehicle Control Expert System or the Vehicle Control System.

3.6 Vehicle Control Expert System

The Vehicle Control Expert System (VCES) has the capability to perform basic aircraft and aerobatic maneuvers for fighter aircraft. The VCES contains the knowledge to perform expert autopilot maneuvers such as turns at varying bank angles, loops, aileron rolls, Immelmanns, and others. Commands can be received from NLMenu and Voice to execute a maneuver or from PES to execute a sequence of maneuvers.

Also written in the TIDIE/ART knowledge representation, the VCES has the same structures for decision making as the PES but the knowledge is at the task level for specific maneuvers as opposed to the mission level knowledge in the PES. Within a given plan are both waits on specified conditions and settings of objects such as bank to a designated or "target" value. Through the setting of various combinations of target variables of over 20 different objects, the VCES is capable of performing any aerobatic maneuver.

3.7 Vehicle Control System

The Vehicle Control System (VCS) is designed to provide vehicle control of the aircraft using the same abstraction barrier as a pilot, that of the basic aircraft inputs of rudder, stick, and throttle. The VCS is responsible for achieving and maintaining the target values of aircraft state variables set by the expert systems. The VCS makes changes to the appropriate physical control mechanism and monitors the progress of the current value toward the target value. It continues to make corrections as needed until the current value equals the target value, or is within some tolerance of the target value. Then the VCS makes corrections that are required to maintain this target value.

The VCS is written in Lisp and runs in conjunction with the aircraft simulation. The aircraft state variables are classified into two groups: those whose value is the current value set by the simulation equations of motion and those whose value is the target value set by either the VCES or the PES. The variables are named by both a particular area such as airspeed, altitude, heading, etc. and by the physical control mechanism to be used such as stick, throttle, speedbrake, etc.

3.8 Intelligent Vehicle Workstation

The Robotic Air Vehicle program is one of the first users of the TI Intelligent Vehicle Workstation (IVW), a tool for the development of expert systems in the area of intelligent vehicles. The IVW allows the user to 1) define a vehicle platform with equations of motion and the environment in which the vehicle will operate, 2) display a variety of viewpoints (e.g. cockpit panel, out-of-the-canopy, contour map), and 3) observe the vehicle's behavior under the control of the expert system. For the RAV system, the IVW houses the F-16 equations of motion, displays the aircraft dials and gauges, and simulates the navigation and communication world.

3.9 Inter-System Communication

To communicate between the individual systems within the RAV

system, both a message scheme and command interpreter are used. The command interpreter translates the commands from either NLMenu or Voice into the appropriate need for the expert systems or into the state variable setting for the simulation. The message scheme, known as the Postoffice, routes commands and other information to the proper RAV system. It performs the system configuration by installing the individual systems on particular machines. Thus the Postoffice can easily determine the location of the system receiving the message and can quickly route the message to the correct address.

4 ROBOTIC AIR VEHICLE SYSTEM HARDWARE ARCHITECTURE

The Robotic Air Vehicle system software currently resides in a hardware configuration of five TI Explorer Lisp Machines and one Digital Equipment Corporation MicroVax II. This configuration is used during system development and test when it is advantageous to have the individual system's displays visible, however, other configurations with fewer Explorers can be used. All the machines are linked together via Local Area Network (LAN). The Explorers, symbolic processors, house all the AI software and the vehicle control and simulation software. The MicroVax II, a floating point numeric processor, performs the mathematical computations required for the conventional software systems.

4.1 Explorer (TM) Lisp Machine

The Explorer is a symbolic computer, developed by Texas Instruments as a tool for developing Artificial Intelligence software systems. It is a seven slot enclosure, single user workstation designed to process high level, stack oriented, LISP like languages. The Explorer utilizes the NuBus (TM) architecture that handles 32 bit addressing and data. The basic Explorer system consists of: a black and white monitor with a mouse and keyboard, a system chassis, and mass storage units. This basic system can easily be expanded to include a microphone or headset utilized by the Voice system, via the connections in the monitor. The system chassis contains one slot for the addition of optional hardware as well as one for additional memory, providing up to a total of 24 megabytes of Random Access Memory (RAM). The top surface of the system chassis allows the stacking of up to four mass storage units. The system hardware has built-in self tests and is user maintainable, meaning that the user has the ability to alter both the system hardware and software configurations.

4.2 Odyssey Processor

The Odyssey board can be installed in an Explorer chassis and connected to the NuBus. The processor can communicate directly with the RAV software environment via its Lisp-callable device-service routines. Although the Odyssey board is used exclusively in the RAV system for voice recognition, it also can perform signal and image processing for other applications.

4.3 MicroVax II (TM)

The MicroVax II parallels the computing power of the larger Digital Equipment Corporation's Vax systems such as the 11/780, but is smaller in physical capacity. The system has 5 megabytes of RAM and 70 megabytes of disk storage. Since floating point arithmetic is best executed on the MicroVax, the mathematical computations of the RAV route planner and passive navigation are performed there and the results transmitted across the network to the Explorer systems.

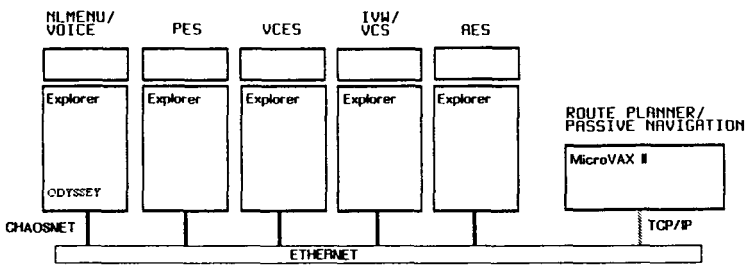


Figure 2. RAV SYSTEM HARDWARE ARCHITECTURE

4.4 Ethernet (TM) And Software Protocols

When more than one Explorer is required for software development, a means of communication is provided to permit asynchronous data transmissions. The Explorer is equipped with Ethernet hardware and controlling software to permit integration into any LAN. The Explorers and the MicroVax II used for the RAV system are physically connected via an Ethernet cable. Ethernet is an industry standard for computer communications and easily supports a multi-system development environment as in the RAV system. The network communications between the Explorer and the MicroVax II is through the Department of Defense's protocol software, TCP/IP. The Explorer to Explorer communications utilize the selected standard protocol for the Explorer, M.I.T.'s Chaosnet.

5 RECONNAISSANCE MISSION EXAMPLE

This section is provided to illustrate how the RAV system components interact during a typical reconnaissance mission. A reconnaissance mission consists of the following mission phases: takeoff from home base, fly a standard instrument departure to an initial waypoint, navigate the various ingress waypoints to the IP, at the target take the recon photos, navigate the various egress waypoints, and finally perform an approach and landing to the home base.

To start the RAV system, the beginning, IP, target, and ending waypoints are specified to the route planner that produces a terrain following/terrain avoidance route. A command is issued from either NLMenu or Voice to execute the mission from a specified airport. The PES processes the command performing a takeoff and standard instrument departure from the specified airport. Once airborne, the PES is continually receiving its current position from the passive navigation system via the AES. Next, the plan from the route planner is processed by the PES for use in navigating the aircraft to the initial waypoint and to each successive waypoint. Navigation to these waypoints includes the PES' issuing of airspeed, altitude, heading, radial, and other targets to the VCS. To monitor the progress of the RAV along the misson, the aircraft's current position is dynamically updated on an aeronautical section map displayed by the AES. A display of the pertainent cockpit dials and guages by the IVW system continually updates the current status of the aircraft. As the RAV nears its home base, the PES issues a query to the AES to determine the bearing and range to the approach fix, the point at which final approach begins. The AES also updates its display to show the approach plates and runway diagram of the airport. Using the airports Instrument Landing System, the PES executes plans for final approach and landing.

6 SUMMARY

This paper discribed the Robotic Air Vehicle system currently under development at Texas Instruments for the DARPA/AFWAL contract F33615-82-C-1841. The RAV system has combined multiple forms of both numeric and symbolic processing to achieve the objectives of piloting a robotic air vehicle and navigating with passive sensors.

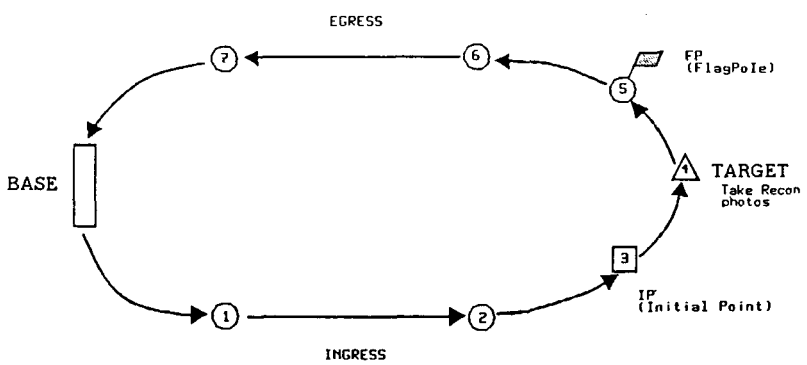


Figure 3. RECONNAISSANCE MISSION EXAMPLE

7 ACKNOWLEDGEMENTS

Explorer, NuBus, and NLMenu are trademarks of Texas Instruments Incorporated. Ethernet is a trademark of Xerox Corporation. MicroVax II is a trademark of Digital Equipment Corporation.

8 REFERENCES

ART Reference Manual, Los Angeles CA: Inference Corporation (1986).

Blair J., Schricker K. "Robotic Air Vehicle: A Pilot's Perspective" NAECON '86 Proceedings, Dayton OH (1986).

Lystad G. "The TI Dallas Inference Engine (TIDIE) Knowledge Representation System" NAECON '87 Proceedings, Dayton OH (1987).

McNulty C. "Knowledge Engineering for a Piloting Expert System" NAECON '87 Proceedings, Dayton OH (1987).

Scott S. "A Generalized Airspace Expert System" NAECON '87 Proceedings, Dayton OH (1987).

Texas Instruments Explorer Technical Summary, Austin TX: Texas Instruments Incorporated (1984).

U. S. Air Force, Tactical Air Command Regulations 60-2, Vol III (8 July 1985).

U. S. Air Force, Flight Manual USAF/EPAF Series Aircraft F-16A/B, USAF, Change 1 (16 April 1982).

SWAN: AN EXPERT SYSTEM WITH NATURAL LANGUAGE INTERFACE FOR TACTICAL AIR CAPABILITY ASSESSMENT

Robert M. Simmons
Systems Research and Applications Corporation
2000 North 15th Street
Arlington, Virginia 22203

ABSTRACT

SWAN is an expert system and natural language interface for assessing the war-fighting capability of Air Force units in central Europe. The expert system is an object-oriented knowledge-based simulation with an alternate worlds facility for performing "what if" excursions. Responses from the system take the form of generated text, tables, or graphs. The natural language interface is an expert system in its own right, with a knowledge base and rules which understand how to access external databases, models, or expert systems. The distinguishing feature of the Air Force expert system is its use of meta-knowledge to generate explanations in the frame- and procedure-based environment.

INTRODUCTION

The goal of the SWAN project is to demonstrate the feasibility of artificial intelligence technology to assess tactical air capability for the Air Force. SWAN deals with four airbases in central Europe, the aircraft and squadrons assigned to those airbases, munitions and other resources required for sortie generation, and the missions flown by the aircraft. Notable factors outside the scope of SWAN include weather, targets, and enemy capability. Air Force units are tasked to fly missions via the Air Tasking Order (ATO). A unit's ability to execute the ATO depends on several factors: weather, resource limitations, aircraft availability, and others. Based on its knowledge about aircraft, munitions, and missions, SWAN determines the ability of a unit to execute its tasking. SWAN identifies any factors which limit a unit's capability and provides facilities for relaxing constraints to improve projected capability, and for tightening constraints to perform sensitivity analysis.

Figure 1 shows the high-level SWAN modules. The SWAN front-end accepts English questions covering a wide range of queries: general domain knowledge, tasking, capability, limiting factors and "what if" excursions. After understanding a question, the SWAN natural language interface sends a semantic representation of the question to the expert system for processing. When the expert system returns an answer, SWAN constructs a semantic representation of the answer for subsequent generation.

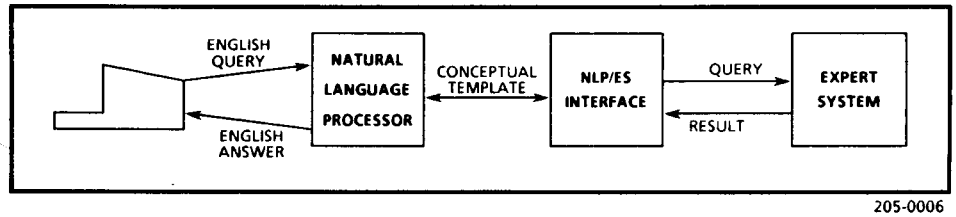


Figure 1. SWAN modules.

This paper describes our approach to two important aspects of SWAN: explanation and natural language query of the expert system. There is a wide spectrum of approaches to explanation. Simple rule-based systems explain by tracing the rules that led to a conclusion. There is little control over

the level of detail in the explanation; some rules may be at a high conceptual level while others perform low-level calculations. At the other end of the spectrum are explanations from human experts. They are concise, to the point, and tailored to the listener's expertise. SWAN uses meta-knowledge to produce explanations derived from the expert's view of the domain, not from a trace of the expert system's calculations. Our approach produces explanations based on general knowledge of the domain and specific data from the user's question and the expert system. The other focus of this paper is natural language query of an expert system. Such query is difficult because of a recurring disconnect between the way a user phrases a question and the relatively rigid way the knowledge is stored. We describe a knowledge-based approach that gives SWAN the flexibility it needs to understand indirect queries. A typical question in our domain is "What mission does Hahn airbase fly?", difficult in light of the fact that airbases do not fly missions; it is the aircraft at those bases that fly. Our approach frees the user from having to know how (even where) the knowledge is stored.

The next two sections of the paper present general descriptions of the expert system and natural language processor. The remainder of the paper describes in detail the explanation facility of the expert system and the natural language interface strategy for handling indirect queries.

The SWAN Expert System

SWAN's expert system includes an extensive domain knowledge base and LISP procedures which model the sortie generation process. The knowledge base is frame-based, developed in KEE. The sortie generation model runs in LISP on top of KEE. Various procedures calculate the capability to execute tasking, determine limiting factors, and modify assumptions or constraints to permit recalculation. Modification of assumptions or constraints results in the generation of an "alternate world" and recalculation within that world. The use of alternate worlds gives the user a powerful capability to investigate possible remedies to factors which limit capability. SWAN has the ability to move between worlds by referring to the assumptions that make the world unique. The user can compare results across worlds. SWAN can generate explanations about specific results and its domain in general; this is significant considering the absence of classical expert system rules. This capability is discussed in detail in the section "EXPLANATION IN A NON-RULE-BASED ENVIRONMENT."

The Natural Language Interface

SWAN's natural language interface combines syntactic parsing with semantic analysis to produce a "deep structure" semantic representation of a question. This deep structure representation is used to communicate with the expert system. The natural language interface has five processing phases: (1) preprocessing; (2) parsing; (3) semantic analysis; (4) interface processing; and (5) answer generation. Preprocessing includes a spelling checker and morphological analyzer. SWAN's parser is an adaptation of the DIAMOND parser and DIAGRAM grammar from SRI's TEAM system [1, 6].

DIAMOND often generates multiple parses for a question, due to the lack of semantics which specify how sentence constituents should be syntactically attached. SWAN must determine the deep structure of the question from among those several parses. SWAN's parser accepts sentence fragments such as noun phrases, prepositional phrases, and verb phrases. SWAN recognizes such elliptical questions and processes them during the semantic understanding phase. The parser uses a lexicon of approximately 5,000 words. The lexicon contains syntactic information for the parser as well as semantic pointers into the natural language interface's knowledge base. We emphasized Air Force jargon to help SWAN understand the everyday language of users in this domain.

EXPLANATION IN A NON-RULE-BASED ENVIRONMENT

Explanations from an expert system are necessary to instill confidence and to facilitate testing of the system. The level of detail is important -- explanations must be complete enough to account for all

system behavior, general enough to be meaningful to the user, and specific enough to clearly relate to the question at hand [2]. There are several types of explanation that SWAN must handle:

1. Justification for basic facts in the knowledge base;
2. Explanation of results; and
3. Explanation of processes.

Explanations of the first type must refer to the original knowledge source (expert or document) or to supporting logic. The second type of explanation comes from questions such as "How did you get that answer?" The third type comes from questions such as "How do you determine the best munition to use?" or "Why do you need that information?"

In a non-rule-based environment, explanations are difficult to generate. A typical SWAN computation might pass control from a LISP function to a knowledge base query. Access to a frame in the knowledge base may trigger a daemon which transfers control to another LISP function which performs additional knowledge base access, and so on. There is no homogeneous thread of logic as in traditional rule-based systems such as MYCIN [3].

The approach to explanation in SWAN is based on meta-knowledge. This work is similar in spirit to the XPLAIN system [4] and the EES project [5], except that SWAN does not automate the development of an expert system with a program writer as in EES. SWAN maintains a meta-knowledge network of "common sense" domain knowledge about the sortie generation domain and the expert system itself. "Common sense" in this context is relative to the domain; the knowledge consists of general information about objects in the domain and their relationships. There are three types of network entities: objects, events, and scripts. Events are relationships between two objects. SWAN employs a minimal set of relationship (or link) types for defining events. The primitive relationships in the network and their associated semantics are as follows:

1. PRODUCE. X causes an increase in the quantity of Y.
2. CONSUME. X causes a decrease in the quantity of Y.
3. ALTER. X alters the state of Y. The semantics of this relationship are further refined by specifying the degree to which X alters Y and the manner in which X alters Y.
4. DETERMINE. X is necessary and sufficient for Y.
5. REQUIRE. X requires Y in order to be a causal agent.
6. USE. X uses Y as an instrument.

We have been able to encode the bulk of the general domain knowledge using PRODUCE, CONSUME, and ALTER; the network is referred to as the PCA network. Figure 2 shows a representative segment of the network. Our use of a minimal set of links permits the construction of general rules regarding cause/effect relationships between events. For example, SWAN's PCA network defines the following events:

E1. AIRCRAFT -- produce--> SORTIES
E2. AIRCRAFT -- consume--> FUEL

We have a general rule capable of determining that event E1 may REQUIRE event E2. The rule basically states that producers must consume in order to produce.

Besides having rules which can infer relationships between events, SWAN permits the explicit linking of events using a number of higher-level cause and effect relationships:

PREVENT/CAUSE/PERMIT
CANCEL/CURE/NEGATE
COMPETE/SUBSTITUTE

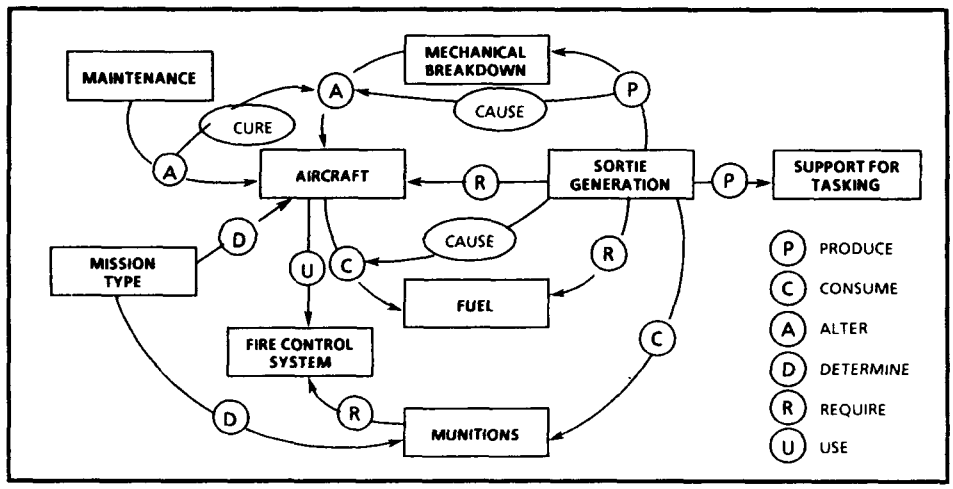


Figure 2. Part of the SWAN PCA network.

PRECEED/FOLLOW/OCCUR-WITH

For example, events E1 and E2, above, might be connected explicitly with a PERMIT relationship:

E3. E2 --permits--> E1

Explicit representation of inter-event relationships is more efficient, but not necessary. In fact, the PCA network is capable of recording new links after inferring relationships between events. Events E1 through E3 and the PCA rules allow SWAN to generate answers to questions such as "What happens if I run out of fuel?" and "How can I get more sorties?" The third type of PCA entity is the script. Scripts organize events and provide an abstraction mechanism for dealing with complex activities at varying levels of detail. Scripts and events can be named, effectively making them objects able to participate in events. SWAN has scripts for refueling, for sortie generation, for maintenance, and others.

Using the PCA network, SWAN generates explanations by relying on knowledge of general processes and relationships. SWAN explains its results by knowing how a process WOULD generate the results. Human experts often explain their results by reworking a problem, going into detail when necessary, to trace the development of the result. This means that "How would you...?" and "How did you...?" questions can be answered identically except for the level of specificity in the answer. SWAN answers the first type of question by traversing the PCA network, looking for appropriate scripts and factors which influence events in the scripts. To explain results from a computation, SWAN uses the same process but refers to information from the question and information from the knowledge base to generate specific answers. For example, the PCA network has the following events regarding aircraft and munitions:

E4. AIRCRAFT --use--> FIRE-CONTROL-SYSTEM
E5. MUNITIONS --require--> FIRE-CONTROL-SYSTEM

Consider the question "Why can't F-16s carry Sparrow missiles?" From its taxonomy, SWAN knows that the F-16 is an AIRCRAFT and the Sparrow missile is a MUNITION. SWAN searches the PCA network for a path between AIRCRAFT and MUNITION, retrieving the information in E4 and E5. This is sufficient to generate a general explanation:

AIRCRAFT USE FIRE CONTROL SYSTEMS REQUIRED BY MUNITIONS

To produce the specific explanation, SWAN substitutes information from the question into the general explanation:

F-16S DO NOT USE THE FIRE CONTROL SYSTEMS REQUIRED BY SPARROWS

Notice that the negation assumption in the original question leads SWAN to generate a negative answer.

The PCA network also supports general "What if...?" and "How can...?" questions through a network traversal strategy that resembles forward and backward rule chaining. In the forward direction, SWAN examines the links from an object to determine the effects of an increase or decrease in that object. For example, for the question "What happens if we fly more sorties?", SWAN would examine links involving the PCA node SORTIES and produce the following information:

INCREASING SORTIES WOULD PRODUCE MORE TASKING SUPPORT
INCREASING SORTIES WOULD CONSUME MORE FUEL
INCREASING SORTIES WOULD CONSUME MORE MUNITIONS
INCREASING SORTIES WOULD PRODUCE MORE MECHANICAL BREAKDOWN

The above explanation points out that flying more sorties is good, but it has a cost associated with it. To answer goal-directed questions such as "How can I get more aircraft?", SWAN looks for events leading into the PCA node AIRCRAFT that produce the specified effect (increase). SWAN's answer to the question would be:

THERE ARE 3 POSSIBLE WAYS TO INCREASE AIRCRAFT:
YOU CAN INCREASE WARGOER RISK OR INCREASE MC RATE TO INCREASE AIRCRAFT STATUS WHICH AUGMENTS AIRCRAFT INVENTORY.
YOU CAN CHANGE 4102 PLAN TO INCREASE AUGMENTATION WHICH INCREASES AIRCRAFT INVENTORY.
YOU CAN DECREASE ENEMY AIR DEFENSE TO DECREASE HOSTILE ATTRITION WHICH DESTROYS AIRCRAFT.

The first recommendation says that you should either fly with aircraft that are broken (accepting risk) or improve the maintenance rate (MC is Mission Capable) for the aircraft. The second states that you can try to alter the resupply plan (4102) that provides for additional aircraft from the United States in wartime. The third recommendation says that you should try to offset the effects of attrition, in which the enemy shoots down your aircraft. Notice that the last recommendation is wrong; we have improved SWAN's knowledge so that it understands that stopping the consumption of X is not the same as producing X.

NATURAL LANGUAGE QUERY OF THE EXPERT SYSTEM

There are two basic problems which SWAN's natural language interface solves. First, SWAN must determine the semantics of a question. Semantically-equivalent questions can be phrased any number of ways. Even for a single question, however, the presence of multiple parses aggravates the problem of semantic interpretation. Second, SWAN must get the expert system to answer the question. A user's perspective may differ considerably from that of the expert system; SWAN has the flexibility to translate between one perspective and the other.

To determine the semantics of a question from the parse tree, SWAN relies on a bottom-up approach that is largely immune to confusion from multiple parses. Semantic information associated with the leaves of the parse tree identify the basic meaning of individual words. To understand the meaning of constituent phrases, SWAN manages a process of semantic combination and attachment at each node in the parse tree. Verb nodes activate "conceptual templates" that represent domain activities or relationships. These templates form the framework for a question's deep structure. Most other leaf categories (noun, adjective, adverb, determiner) become "semantic objects" that represent domain objects or attributes. Some semantic objects can combine with each other to produce different objects. In other cases, semantic objects attach to other semantic objects (e.g.articles/adjectives to nouns) or to conceptual templates.

SWAN interprets the combination of semantic objects as a query of the expert system's knowledge base. For example, consider the phrases:

aircraft at Hahn	(prepositional modifier)
Hahn aircraft	(noun-noun modifier)
Hahn's aircraft	(possessive)

Each phrase refers to the "F-16" aircraft, the type of aircraft based at Hahn. SWAN semantically collapses these phrases by combining the semantic objects [AIRBASE HAHN] and [VEHICLE AIRCRAFT] to produce [AIRCRAFT F-16]. This results from the generation of an expert system query to determine the AIRCRAFT of HAHN. The order of combination is not important because SWAN determines that AIRCRAFT is more general than HAHN (general class versus proper name). Therefore, SWAN acts as if AIRCRAFT is an attribute of the domain object HAHN.

The natural language interface uses a knowledge base to control semantic combination and attachment. To avoid confusion between the knowledge base of the natural language interface and that of the expert system, further references to a knowledge base will be NLI-KB or ES-KB, respectively. NLI-KB covers a broader domain than ES-KB, though at less depth. Information in NLI-KB includes a CLASS/SUBCLASS taxonomy, attachment restrictions, and expert system query information. Both NLI-KB and ES-KB are frame-based, containing a network of objects and attributes. NLI-KB contains knowledge about the information in ES-KB. For each pair of domain object classes that are meaningful to query, the NLI-KB contains a list of slot names that define a path through ES-KB. For example, in NLI-KB, the AIRCRAFT frame has an AIRBASE attribute. We can determine the AIRCRAFT of a specific AIRBASE by generating a query from information in the AIRBASE attribute of the NLI-KB frame below:

```
(AIRCRAFT
   ...
   (AIRBASE (INVENTORY) (FIGHTER-SQUADRON))
   ...)
```

The information above can be viewed functionally as

(INVENTORY (FIGHTER-SQUADRON AIRBASE)) = = > AIRCRAFT

The AIRCRAFT of an AIRBASE is not found in a slot of an AIRBASE frame in ES-KB; we find it by looking in the INVENTORY of the FIGHTER-SQUADRON of the AIRBASE. This representation provides a powerful notation for handling indirect expert system queries. With the addition of special "slots" in NLI-KB we can also handle queries that might seem natural to the user but foreign to the expert system. As an example, consider the question "What missions does Hahn airbase fly?" Taken literally, the answer might be "Hahn does not fly missions. Aircraft fly missions." One of SWAN's

design goals, however, is to answer questions the way the user thinks of them. To handle this indirect query, the NLI-KB information

```
(MISSION
   ...
   (AIRBASE (MISSION) (DETOUR AIRCRAFT))
   ...)
```

specifies the special slot (DETOUR AIRCRAFT). The MISSION of an AIRBASE can be found by "detouring" to the AIRCRAFT of the AIRBASE, then querying the MISSION of those AIRCRAFT. Thus SWAN understands the question as if the user had asked "What missions do the aircraft at Hahn fly?" Another special slot (SELECT ...) permits filtering of attribute values according to a user-specified function. We use this to deal with questions requiring the first value, the average value, the maximum value, the "best" value, and so on.

Verbs in a question activate SWAN's conceptual templates. A small number of semantic "predicates" represent all verbs in the lexicon. Each predicate has a number of conceptual templates defined in NLI-KB; each template represents a different sense or use of the predicate. For example, the predicate USE has several templates, including:

```
(USE (AIRCRAFT MUNITION MISSION))
(USE (AIRCRAFT FUEL))
(USE (AIRBASE RESOURCE TIME-UNIT))
```

The top template corresponds to any question about an aircraft using munitions for a mission. The bottom template deals with questions about consumption of resources at an airbase over a period of time. Semantic objects from the parse tree attach to the slots in each template. When processing of the parse tree is complete, SWAN selects the template that most completely matches and sends it to the expert system. Many of these templates require knowledge base queries while others require calculations by the expert system. The interface module shown in figure 1 accepts the conceptual template and formulates queries or calculations for the expert system. After receiving the answer from the expert system, the interface modifies the question template to become an answer template ready for generation.

CONCLUSION

A natural language interface is a useful and feasible means of communication with an expert system. SWAN's natural language interface uses a knowledge-based process of parse tree interpretation to understand a user's questions and interface with the expert system. Our approach allows SWAN to handle queries the way the user asks them. The expert system generates explanations using meta-knowledge that understands general processes and relationships in the domain. This approach is motivated by the absence of rules in the expert system which permit explanations based on traces of forward or backward chaining rules. SWAN's meta-knowledge network defines objects, events, and scripts that capture general expertise about the sortie generation domain. General rules about production and consumption provide the capability to reason about the meta-knowledge in the PCA network and derive new relationships between objects or events.

ACKNOWLEDGEMENTS

The SWAN project team is Sherman Greenstein, Dr. Mary Dee Harris, Dr. Hatte Blejer, David Reel, William Brooks, and Robert Simmons. I would like to thank Hatte Blejer, Bill Brooks, Sherman Greenstein, and Helen Simmons for their comments on this paper.

SWAN is being developed as part of the AI Tactical C2 Demonstration Project, Department of the Air Force contract number 85X-54277C.

REFERENCES

1. Winograd, T., LANGUAGE AS A COGNITIVE PROCESS (SYNTAX), Addison-Wesley, 1983, pp. 381-382.
2. Davis, R., and Buchanan, B. "Meta-level knowledge: Overview and Applications," IJCAI 5, pp. 920-928.
3. Shoftliffe, E. H., COMPUTER-BASED MEDICAL CONSULTANTS: MYCIN, American Elsevier, New York, 1976.
4. Swartout, W., "XPLAIN: A system for creating and explaining expert consulting systems," ARTIFICIAL INTELLIGENCE, 21, 3 September, 1983, pp. 285-325.
5. Neches, R., Swartout, W., Moore, J., "Explainable (and maintainable) expert systems," IJCAI 9, pp. 382-389.
6. Robinson, J., "Extending Grammars to new Domains," RR-83-123, USC/ISI, Marina del Rey, CA, January, 1984.

EXPERT SYSTEM APPLICATIONS IN SPACECRAFT SUBSYSTEM CONTROLLERS

Paul F. Marshall
NASA, Lyndon B. Johnson Space Center
Mail Code: EC2
Houston, Tx 77058

ABSTRACT

As the NASA progresses into the development phase of the Space Station, it recognizes the importance and potential payback of highly autonomous spacecraft subsystems. This paper presents priorities for embedded expert system enhancements to the automatic control systems of the Space Station thermal, EVA, and life support systems. The primary emphasis is on top-level application areas and development concerns for expert systems.

INTRODUCTION

The primary intent of this paper is to identify potential application areas and development concepts for expert systems within the operational control scheme of some Space Station subsystems.*** The application areas considered are those which are felt to have the greatest payback potential to the Space Station in terms of system safety, autonomy, and operational costs. Although the capabilities mentioned here will be useful for any subsystem, not all subsystems have been considered nor are being represented in this discussion, nor is this paper intended to be a thorough treatment of all subsystem controller requirements.

The discipline area on which this paper focuses is Space Station life support. Specific subsystems which have been considered include the Thermal Control System (TCS), the Extravehicular Mobility Unit (EMU), the Manned Maneuvering Unit (MMU), and their associated checkout and servicing systems, and some aspects of the Environmental Control and Life Support System (ECLSS). All of these subsystems involve complex continuous processes which must be controlled. Of these processes, some are mechanical (circulating fluids, heat transfer, etc.) and some involve chemical reactions, both open-loop and closed-loop. Each of these systems is a complex integration of components and subassemblies (many of which need to be actively controlled) with a high degree of redundancy and, usually, multiple levels of control. Each incorporates a wide variety of system instrumentation and depend upon numerous interfaces with other vehicle subsystems (power, data, communication, etc.) and Space Station elements (modules, airlocks, truss, experiment bays, etc.).

The controllers for these subsystems support a vehicle-level control system currently being developed known as the Operations Management System (OMS). The OMS has an on-board component and a ground based component. The overall OMS structure is depicted in Figure 1. The on-board portion of the OMS would be integrated into the Data Management System (DMS) for access and communication with the other vehicle subsystems. The portion of the figure labeled "distributed systems" refers to the individual vehicle subsystem controllers which include those mentioned above.

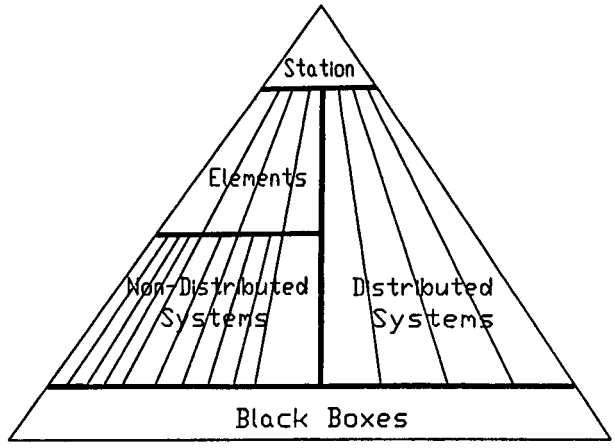


Figure 1 - Operations Management Systems Concept

***Many other functions have been identified as promising application areas for expert system approaches which also reside at the subsystem levels but may not relate to the operation and control of the subsystem itself. These higher level functions will not be discussed in-depth but some are mentioned below for completeness: resource management, maintenance assistance, logistics/inventory management, human interfaces, interactive training, and simulation support.

SUBSYSTEM CONTROL APPROACH

A very simplistic view of conventional process control, but one which is useful for the present discussion, is one which breaks a system down into a relationship of mathematical transfer functions, signal flows, and response characteristics tailored not only to the system being controlled but also to particular solution techniques. This approach has historically proven successful in the solution of control problems. Although the models used in the conventional approach have associated inaccuracies, when the technique fails it is usually due to influences which lie outside of the models and control laws. Specifically, the approach usually fails due to 1) influences of environment or other boundary condition changes which are difficult or impossible to incorporate into the control models (such as vehicle attitude perturbations, proximity to other equipment, etc.), 2) failure of system components, 3) changing goals or design assumptions, 4) improperly executed procedures, 5)incomplete or unreliable system data, and the like. If these influences could be properly identified and controlled, then the reliability and autonomy of the control law could greatly be enhanced.

It is best to view a more robust control approach to include the conventional capabilities enhanced with resources to accomplish the other aspects of the control problem. The overall control scenario can then be conceptually described as a process management system consisting of the following five functions (ref. 1):

-Data preprocessing
-System state/situation assessment
-Control action determination
-Command execution and verification
-Process management controls

The relationships of these functions are depicted in Figure 2. In the sections which follow, each

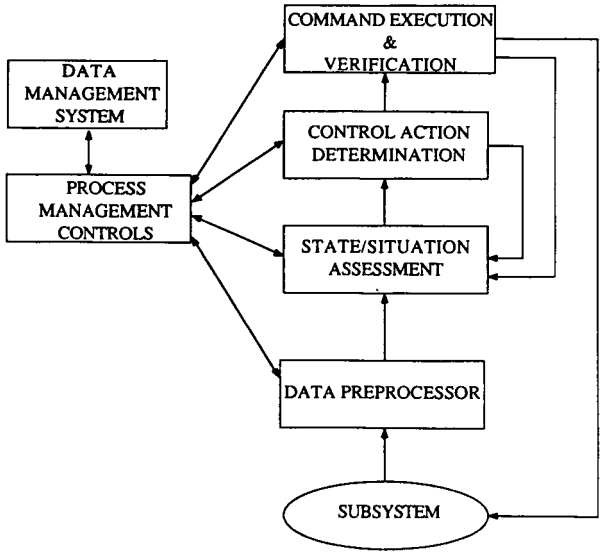


Figure 2 - Subsystem Controller Functions

of these functions are described and the areas where expert systems may be applied to accomplish the broader goals which have been established are identified.

DATA PREPROCESSING can be thought of as the set of tasks required to acquire raw data from the operating subsystem, and put them into a form which can be directly useful to the other functional areas. The preprocessor may also 'glean information' from the data by performing simple calculations using individual parameters, or recognizing or developing much more complex relationships between parameters which can generate qualitative descriptions of the system to be used in other functional areas.

While most of this task involves conventional data acquisition and numerical processing (such as engineering unit conversion, limit checking, etc.), there are some capabilities which an expert system may provide to take advantage of this functional area. These may include:

- A flexible facility for developing complex pattern matching relationships using the signatures of large amounts of data to generate qualitative information about the system, and to identify subsystem events. This function would be sensitive to subtle trends and patterns in not only individual parameters but groups of parameters as well. Statistics on the certainty of the quantitative or qualitative values returned would be maintained and passed on with the values. This facility could be the foundation of a structured and controlled data driven learning capability.
- A situation or experience driven capability to judiciously acquire data from the subsystem in order to satisfy the goals of the current operation without overloading the control system with data. This function would include a data sampling scheduler which adjusts sampling rates, or otherwise controls the supply of data according to the current and anticipated needs of the control system.

SYSTEM STATE/SITUATION ASSESSMENT tasks are associated with describing the current state of the system and identifying the state(s) to which the system may be headed at any time. Some of the activities included in this function would be configuration management, failure detection, subsystem performance status, subsystem response prediction, etc. This function would also establish specific control goals to optimize the current state and would accept higher level goals/objectives from the process management function with which to modify the specific control goals.

Expert system applications in this area may include:

- Goal management. In the context of OMS, vehicle operations imperatives and resource management objectives will be submitted to all subsystem controllers routinely. Each subsystem must, therefore, have facilities to process these objectives into subsystem control goals and to further modify these goals as operating environments or the objectives themselves change. Objectives will change, for example, with Crew Activity Plan modifications within the OMS which subsystems will need to respond to and plan for prior to implementation. The process controllers must be able to assign priorities and temporal order to the incorporation of these goals into the control strategy.
- Failure detection. Most critical failures which occur within systems of this nature can reliably be detected using conventional numerical techniques (i.e. redline limit violations, etc.) There may, however, be situations where qualitative reasoning can effectively enhance the purely numerical approach. Cascading failure detection, multiple independent failures, and failures involving incomplete or erroneous instrumentation readings are examples of these situations.
- Configuration verification. In this task, the subsystem configuration (e.g. valve positions, active components, operating mode, operational margins, etc.) is checked and verified prior to initiating any command plans. If the parametric information which the system normally uses to determine configuration becomes disabled or is otherwise incomplete, the conventional approaches will not work, and the control system could be in danger of issuing improper or incorrect commands. A knowledge based approach becomes necessary to indirectly ascertain the configuration by referring to other available instrumentation and inferring the result either through application of rules or by reference to directed (narrow scope) math models to establish plausible solutions.
- Trend analysis. The class of subsystems described here is naturally dynamic during normal operations. A qualitative analysis of the parametric trends (either individually or as groups of parameters) is often necessary to explicitly describe the state of the subsystem and to predict what states, or situations may arise with the current control approach. Trend analysis is sometimes required as well to help identify failures within the system.
- Model-based determinations will at times be required to establish details of the goal state for the system and to enable limited prediction capability for evaluating effects of proposed changes to the system as suggested by other controller functions. These process models would necessarily be of fairly narrow scope to allow efficient application of the models in real time and to assure reliable and consistent results which are easy to interpret. The models could be empirical, analytical, or hybrid representations of the process.

CONTROL ACTION DETERMINATION tasks combine to produce a control strategy and a plan to accomplish the established goals for the subsystem. The plan would consist of direct commands to be sent to the subsystem (with expected results of each), diagnostic strategies, or conditional strategies as required, in order to realize the goals. This function must also provide an effective replanning function whereby current plans are gracefully terminated and new plans are initiated as a response to changes in goals or to critical events within the system. This function would maintain flight rule standards and would verify the integrity of the system configuration and redundancy state as major factors of the provided plan.

An application area of expert systems within this planning function with great payback potential is adaptive control. Adaptive control in this context can be viewed as the techniques by which the implementation plans of the controller establishes and tailors the control strategies for the entire subsystem, and/or the response characteristics of the individual control loops (by modifying control loop gains, lag times, etc.), in order to realize the current and anticipated goal states for the process. Implementation of this function should be viewed as the principle area of enhancement to the conventional process control theory as applied to the subsystem. This activity necessitates the granting of control authority to an 'autonomous supervisor' function in order to change strategies and responses within a well developed performance envelope. It is an responsibility of this expert function to recognize its limits of control authority and never operate outside this envelope without the approval of human operators. Conceptually, the performance boundaries of the adaptive control planning function should be changeable, but only by appropriate human operators, not the controller.

Another potential expert system application area for this function is failure recovery. The proper response to failures, once detected, is not always clearly procedural. Responses can vary depending on the current state of the system, the desired goal state(s), the severity of the failure (if discernible), crew safety, changing flight rules, the criticality of the failure, available redundancy, other existing failures, and the

management of other perceived contingencies, to name a few. Expert systems present resources to efficiently manipulate both qualitative and quantitative information to resolve the many conflicting inputs, or to quickly ascertain where conflicts cannot immediately be resolved.

COMMAND EXECUTION AND VERIFICATION tasks implement the plan of action provided, verify whether each command is executed in the proper order and discern whether the anticipated system effects were realized. This function also provides a check to determine whether assumptions made during the scheduling of a command plan (such as system configuration, or boundary conditions) are continuously valid. As such, this is a task highly oriented towards computer communications and data management and major expert system applications are not obvious here (other than tasks which favorably trade off against conventional techniques in terms of performance, reliability, etc.).

PROCESS MANAGEMENT CONTROLS tasks allocate resources to each of the above functions, assign top-level task priorities, schedule activities, oversee interrupts of tasks, and monitor the overall status of the subsystem controller. This function would also communicate with the DMS, receiving vehicle operations objectives, requests for information, routing these inputs to the proper controller functions in the proper form, and providing responses to the DMS.

The primary expert system task to be performed within this function is the maintenance of subsystem knowledge as it applies to the rest of the vehicle and to the other controller functions. Specifically, this involves the activity of translating vehicle commands, objectives, or resource guidelines into clear goals or instructions applicable to the controller functions without passing unnecessary information. Once subsystem goals are established, the process manager must pass the information to the affected controller functions, establish task interrupt priorities, reallocate processor resources, etc. as required in order to oversee transition to the new objectives. The process manager must process status requests from the vehicle and must determine when subsystem events should be communicated to the vehicle due to anticipated global effects which could impact overall operational capabilities (such as reduced resource capacity).

EXPERT SYSTEM APPLICATION ISSUES

Having a suggested functional framework for what the subsystem controller must do and major areas where expert system techniques may provide the most benefit, a few major development issues which relate to both warrant some attention.

REAL TIME PERFORMANCE is a primary concern for any control system. If the control system cannot 'stay ahead' of the process being controlled, the process cannot be reliably controlled. The performance requirements for the control systems vary depending on the responsiveness of the processes. For the subsystem applications considered here, response times can vary from on the order of hours with some thermal processes to fractional seconds for the navigational requirements of an enhanced MMU. But viewed as tasks contained within controller functions, not all expert system functions will need to be equally responsive. Some tasks will always be performed on a lower priority basis and should never be highly time critical tasks, such as model-based reasoning: The selection of any programming technique, tool, or mix of technology for application to any control system task needs to be made with consideration of its demonstrated performance as well as its ability to arrive at solutions, difficulty and cost of development, maintainability, etc.

THE BASING OF SUBSYSTEM CONTROLLER FUNCTIONS on-orbit or on the ground is a decision which requires careful consideration by subsystem and OMS designers alike. Given communications resources such as two TDRSS satellites and the Space Station Information System (SSIS), it is not obvious that any of the noncritical subsystem control functions should be placed onboard. Onboard computing services will always be more scarce and expensive than similar services on the ground, particularly if the computing requirements involve multiple processor types. While specifying the criticality of functions within the control scheme is premature at this point for Space Station, this must be a strong criterion in arriving at a consistent common approach for subsystem control design.

CONCLUSIONS

Expert system applications within subsystem controllers provide the system engineer with tools to solve problems which cannot reliably be solved with conventional techniques. From this perspective, expert systems can enhance the conventional control approach and should be designed to easily interact with numerical processes. Operating within a real time system, expert system tasks should be well constrained within the envelope of known solutions, and should provide new methods for moving around the envelope with a high degree of autonomy from onboard or ground based operators. Expert system tasks should respond to human requests and commands at a high level of priority and should never be allowed to address problems which lie outside the known and demonstrated set of solutions.

While this discussion provides a structure to identify potential expert system application areas within the overall controller context, it is not a comprehensive approach. Each subsystem discipline should be made part of a process to develop subsystem controller functional requirements from the 'bottom-up' in the context of the 'top-down' development of the vehicle OMS. This should allow both hardware and software technologists to focus on what will be required for the Space Station subsystem controllers. It should also provide

program managers clear programmatic requirements to make it possible to attain them. An effort like this will also highlight to line organizations the skills needed to support development of controller software, and should reduce subsequent software conversion and interface problems later in the program.

ACKNOWLEDGMENTS

The author wishes to acknowledge the following individuals who provided inputs, insight, and perspective to most of the ideas presented: Mr. Mike Lounge, Mr. Joe Chambliss, Mr. Jack Aldridge, Mr. Neal Hammond, and Mr. Bob Savely.

REFERENCES

1. D'Ambrosio, Bruce, et al, "Real-Time Process Management for Materials Composition in Chemical Manufacturing," IEEE EXPERT MAGAZINE, Summer 1987, pp. 80-93.

2. Yakowitz, Sidney J., "Adaptive Control Processes," MATHEMATICS OF ADAPTIVE CONTROL PROCESSES, American Elsevier Publishing Co. Inc., New York, New York, 1969, pp. 26-48.

3. Whitsett, C. E., "Role of the Manned Maneuvering Unit for the Space Station," Paper #861012, 16th Intersociety Conference of Environmental Systems, San Diego, California, July 14-16, 1986.

4. Rankin, J. Gary, "Space Station Thermal Management System Development Status and Plans," Paper #851350, 15th Intersociety Conference on Environmental Systems, San Francisco, California, July 15-17, 1985.

THE KATE SHELL- AN IMPLEMENTATION OF MODEL-BASED CONTROL, MONITOR AND DIAGNOSIS

July 1987

Matthew Cornell

Artificial Intelligence Office (mail stop: DL-DSD-22)
NASA/Kennedy Space Center, Florida 32899 (305) 867-3224

ABSTRACT

The conventional control and monitor software currently used by the Space Center for Space Shuttle processing has many limitations such as high maintenance costs, limited diagnostic capabilities and simulation support. These limitations have driven us to develop a knowledge-based (or model-based) shell to generically control and monitor electro-mechanical systems. The knowledge base describes the system's structure and function and is used by a software shell to do real-time constraints checking, low-level control of components, diagnosis of detected faults, sensor validation, automatic generation of schematic diagrams and automatic recovery from failures. This approach is more versatile and more powerful than the conventional "hard coded" approach and offers many advantages over it, although, for systems which require high speed reaction times or aren't well understood, knowledge-based control and monitor systems may not yet be appropriate.

INTRODUCTION

Background of Problem

The current Launch Processing System (LPS) at the Kennedy Space Center (KSC) has the duty of checking-out and launching Space Shuttle vehicles. The responsibility begins with testing the Orbiter's many systems in the Orbiter Processing Facility, continues through the assembly and check-out of the integrated vehicle (External Tank, Solid Rocket Boosters and Orbiter) and ends with final countdown and launch. The computing system which supports these tasks is a decade-old network of 15 consoles (1 console = 1 computer & 3 monitor/keyboard units), a huge data-communications system (Front End Processors and Common Data Buffers), and large Mainframes for data storage and retrieval, documentation and configuration control. The console computers themselves have a 16-bit address bus, 64K of RAM and a single 80Mb Winchester disk. To provide a feel for the capability of this system, the total number of "Function Designators" (FD's) in the KSC database, which describe all commands and measurements, is approximately 95,000 including Cargo, Facility, Ground and Flight FD's. The actual number used for each launch is in the neighborhood of 40,000.

LPS (Conventional) Software Approach

The software applications running on the console computers used to check-out the Space Shuttle vehicle are written in a NASA-designed procedural language called Ground Operations Aerospace Language or GOAL. These GOAL procedures are conventional control and monitor programs specific to the system the writer/engineer is responsible for and vary widely from system to system and even within a single system (due to the fact that many people participate in writing system test programs). The typical program is "hard-coded" to the components and FD's of the system it controls and monitors, specifying exactly what steps to take in order to command any component to any desired state.

LIMITATIONS OF LPS/CONVENTIONAL APPROACH

Limited Diagnostic Capability

The program's only ability to perform diagnostics are of the failure tree sort which maps a restricted number of possible failure conditions (pattern of measurements) to a probable cause, determined by the engineer and then coded in GOAL. Naturally, a system of reasonable size makes the ability to do a complete tree search impossible. The number of

possibilities grows exponentially with each new component added.

Limited Display Capability

The user interface consists of character-based "display skeletons" with cursor keys for pointing and issuing commands. Systems are displayed as well as possible but the displays can not be dynamically reconfigured and are limited in number for each console. As in many applications today, much of the GOAL code is written to handle the user interface.

Limited Data Retrieval Capability

Data Retrieval is through a command-line interface with unforgiving syntax. Graphs of stored data are available but necessarily restricted due to the character-based display and screen size.

Low-to-Medium Fidelity Simulation

For application program testing and training, simulation is done off-line by software engineers on a system called Shuttle Ground Operations Simulator (SGOS) in a custom simulator language. One problem is that the software engineers are often not familiar with the true operation of the hardware and it takes many iterations to get even the more basic functions working correctly.

PROBLEMS RESULTING FROM CONVENTIONAL APPROACH

Expensive to Maintain

Because each application program is specific to its system, any modifications or corrections in the equipment require the system engineer to individually change all applications and SGOS programs and retest and reverify them (usually a long process) before they are operating correctly.

No Sensor Validation

One of the biggest and most-encountered problems seen during check-out of many systems is that of sensor validation. Many launches and tests have been delayed or aborted due to a sensor indicating a faulty component when, in actuality, the component was fine and the sensor or measurement channel itself was at fault. Some diagnostics can be written to help with this problem but, as stated earlier, it is impossible to handle all cases. Delays of this sort in the Space Program have equated to millions of dollars.

Additional LPS/Space Center-Related Problems

Two additional problems, experienced system engineers reaching retirement age and accelerated launch schedules, are leaving relatively inexperienced personnel to monitor and run the check-outs. These, combined with the problems above, result in both delays and errors in the day-to-day processing.

Obviously, out-dated equipment plays a role in some of the problems listed. But replacing the hardware with no change in software except for the ability to have larger programs will not solve these problems. What is required is a much more intelligent control and monitor approach. We believe the knowledge-based or model-based techniques described here and exhibited in the Kennedy Space Center's Knowledge-based Autonomous Test Engineer project (KATE) are a viable solution.

The remainder of this paper will cover the following topics:

- The Model-based Approach- A discussion of our understanding of model-based control, monitor and diagnosis and how we've applied these techniques to some of the problems at KSC.
- The KATE Implementation- A short status of the KATE project describing its capabilities.
- Limitations- A description of the kinds of systems and situations the KATE shell may not be applicable to.
- Future- A summary of some projects related to the KATE technology and how we hope KATE will affect these other programs.

THE MODEL-BASED APPROACH

Since our understanding of this approach is from an engineering perspective, not a fully comprehensive, academic one, this discussion will necessarily be limited to the work done in developing the KATE shell. Davis [1] and de Kleer [2] provide more complete information on model-based systems. The model-based approach is founded on the idea of describing a system of components in terms of its structure and function. Structure is how the components of the system are interconnected, for example "The output of transistor one is connected to the inputs of transistors two and three". Function, on the other hand, tells us how the component operates in terms of relating inputs to outputs, i.e. a transfer function, and tells us how the component can be expected to perform. For example "The output of transistor one is 2.5V when the input is above or equal to 1.0V and the output is 0.0V when the input is below 1.0V". This

explicit description allows us to effectively separate the system-specific knowledge from the procedural "smarts" which reason about the system's operation from the knowledge base. This dichotomy means physical system changes, such as replacing existing components or adding new components (both of which might change the behavior of the system) require a very easy knowledge base change without impact to the procedural portion (the shell) of the system. This assumes the shell is robust enough to handle the new component.

This approach is markedly different from rule-based control and monitor systems which are very specific to a particular system. The underlying approach to control, monitor and diagnosis employed by these systems is really no different than the conventional approach described above. For diagnosis, rule-based systems use rules about the components to map a particular pattern of measurements and states into a probable cause. They don't really reason about why the answer is probable; such knowledge is simply written down by the engineer (where the real understanding is) and put into rules. Control is done in a similar manner. The options for getting a component to a desired state are decided by the engineer (using his internal model of the system and reasoning skills) and then committed to a rule-base. The major advantages offered by modern rule-based systems are their highly-developed user interfaces (they usually run on Lisp Machines) and development and graphics tools.

Having the system described in a KATE-style knowledge-based format allows our shell to do the same kinds of things engineers do with the information.

Monitoring

Since the shell knows the values of all commands which control the system and how the components of the system work, it can derive the expected values of all objects (including measurements) by "propagating" the effects from the commands down through the knowledge base to the measurements. If these expected measurement values don't match the measured values coming in from the system there is either a malfunctioning component or an inaccurate knowledge base. Hopefully, the system's knowledge base has been carefully developed and matches the real hardware.

Diagnosing

One of the most desirable operations that can be performed on a system is that of diagnosis, the process of determining which components are causing the system to behave in an incorrect manner. The way our shell's diagnoser does this is by gathering a list of all objects which could affect the discrepant measurement (see "Monitoring" above) and eliminating those whose failure can't account for the anomaly. (We liken this process to that of a "mind game"- "If object A failed then it would cause B, C and D to read b, c and d. Since they're not, A must be OK"). If, after processing all controlling components only one is left, then it must be the culprit. Otherwise you're left with a list of suspects, any one of which could be failing. (Our diagnoser only checks for single points of failure, not simultaneous, unrelated multiple failures. The claim can be made that, at least with most systems at KSC, single point failures are the norm and simultaneous, unrelated multiple failures rare [3]). For more information on diagnosing see Scarl [4].

Simulating and Training

Because the knowledge base describes the system's components and operation as accurately as possible, our system is by definition a high-fidelity model. Our shell uses this model inherently in monitoring and diagnosing but has another copy of the knowledge base which runs in parallel to the first and acts as the real-world system. Objects (components) can be failed and the simulator process will propagate the failure through its knowledge base, sending the affected measurements to the constraints (monitoring) system just like the real system would. This allows you to develop the knowledge base (and currently for us the shell) without being physically connected to the system. Also, operators can be trained on this system which (if modelled correctly) will perform just as if connected to the hardware.

Controlling

In traditional control and monitor software the methods of controlling each component of interest are directly and explicitly coded. With the approach used in KATE, a more declarative style of control is available. Instead of telling the program exactly how to accomplish the control goal you can simply request any object (command, measurement, component or pseudo-object such as an internal pressure) to be in any state. The control portion of the shell searches backwards in the knowledge base from the component to the controlling commands, building a context-sensitive (uses current component states) math expression along the way. This math model is used to construct any

and all possible combinations of commands that would give you the desired state. Any combinations which violate previous requirements (see "Maintaining" below) are not allowed. Currently, the KATE shell automatically uses the option which requires the fewest number of commands*. What this means is the low-level (component-level) control is automatically done for you. Upon this low-level base will be built a very high-level control language whose statements would read just like a procedure describing the desired operation of the system. For example: "Maintain AFT-DUCT-FLOW at 25 LBS/MIN for 15 MINUTES".

Maintaining

If it is desired to maintain a particular object in a particular state (like a purge-pressure at some value) you can get it to that state and keep the fact that it's to be maintained on a list. In the future, any diagnosis that indicates the failed component(s) effect that maintained object, the control code can be automatically run again to reinstate that now-violated request. (Recall the control portion is context-sensitive which includes not using failed components).

Drawing

The structure information in the knowledge base gives you much of what you need to draw engineering-style schematic diagrams. Once drawn, the actual values for each measured component (as well as inferred values for all others) can be displayed dynamically, creating a "live" drawing. If system-grouping information is supplied or can be inferred, the displays can be on a functional level with the operator able to request the "level" of viewing. For example, a valve with all its supporting circuitry, commands and measurements can be displayed by itself on the screen, or a number of valves with just their significant measurements (perhaps position) shown. The drawing also allows the user to do control by pointing and clicking the computer's mouse on any component and inputting a desired value.

Single Point Failure Analyzing

Since the control portion of the KATE shell accumulates all possible combinations of commands to achieve a particular goal, its routines can be run on each component to produce a list of components which are single points of failure, i.e. they operate without redundancy. This single point failure analysis program could be run dynamically after each diagnosed failure to give a running account of system vulnerability.

Knowledge Base Generation from CAD Files

Because this entire approach is based on the knowledge base, it is vital that it is as correct and as complete as possible. To aid the system developer in this, much of the knowledge base creation task can be accomplished automatically by a program which uses CAD-generated drawing information.

Automatic Documentation/Retrieval

Since any information can be stored with each object in the knowledge base, it provides a very convenient way to have component documentation (specification sheets, failure histories, vendor/supplier data) and digitized video very easily accessible on-line.

THE KATE IMPLEMENTATION

Many of the features listed above have been embodied in the KATE project at the Kennedy Space Center. KATE is an expansion of an earlier joint MITRE/KSC project called LES, or LOX (Liquid Oxygen) Expert System. LES is a real-time monitor and diagnosis expert system which was used to successfully monitor six Space Shuttle LOX loadings at the Space Center. Scarl [4] and Scarl [5] describe LES's capabilities and methods in detail.

The KATE shell is implemented using a modified version of frames and the Frame Representation Language [6] to represent and access the knowledge base. It runs on both AT-class microcomputers (for demonstrations) and Lisp machines, the latter being used for further development. The code is written entirely in Common Lisp, with extensions used for the user interface and multi-tasking. As of this writing, the KATE shell exhibits monitoring, goal-seeking control/maintenance and diagnosis of failures for objects without state or feedback. In the control and diagnostic portion of the shell, work is being done to handle components which utilize feedback and internal state in their operation and control loops. Work has just begun to update KATE's drawing system to do completely automatic schematic diagram generation and display.

*This is not the final way KATE will chose which option to use; there are usually natural priorities and requirements that would constrain the selection and would be integrated into the control code.

A plotting system is functioning which is smart about how to plot all measurements related to a particular measurement or component. Work is also progressing on a program to do single point failure analysis and generation of KATE-compatible knowledge bases from standard CAD drawing files.

Current development of the KATE shell is being done in order to provide an operational control, monitor and diagnostic system for the Environmental Control System (ECS) in the Orbiter Maintenance and Refurbishment Facility (OMRF) which is currently under construction. This building will be used for storing Space Shuttle Orbiters and doing maintenance work on them and the ECS system will provide a conditioned flow of air to four areas of the orbiter for the purposes of ventilation, cooling, and controlling static electrical discharge. Because the OMRF ECS system consists of many kinds of components not currently handled by KATE, the development of its knowledge base is driving expansion of the shell's capabilities. This will be the first operational real-time control and monitor expert system at KSC and it has the potential of being included in other projects at the Space Center.

LIMITATIONS

Although the systems the KATE shell has been applied to have been mostly electromechanical in nature (pumps, valves, relays, discrete and analog commands and measurements) it can be applied to many others. However, there are some cases and conditions where this approach might not be appropriate. (Those related to speed or efficiency might be addressed by faster computers or more efficient code):

Poorly Understood Systems- Since the KATE shell relies entirely on a very accurate description of the system, the systems which it can handle must be well understood in terms of identifying each component, how the components work and exactly how they are connected, for example an electrical circuit. A biological system (like the human body), since its functioning is not well understood, would not be a good candidate for this type of representation and modeling. However, you can model abstract concepts and you can model on different levels of abstraction with KATE, assuming you understand how those concepts or "boxes" interact and function on the level being modelled.

High Speed Systems- The systems we have applied KATE to have had significant measurement changes occurring about once per second. The reaction times displayed have been on the order of a maximum one second delay from the time when a measurement changes to the time when the shell can react with a command issuance. Diagnosis times are around four seconds for our most complex example* and we expect the maximum to drop to around one second.

"Broad" Systems- Since KATE's searching is depth-sensitive, it is most efficient for systems which have a small number of components between commands and measurements. Systems such as LOX fit this description with mostly four or five components between commands and measurements (one branch) but many copies of those branches repeated throughout the system for redundancy.

Poorly Instrumented Systems- Because KATE, like an engineer, uses supporting measurements for detecting faults and eliminating faulty components during a diagnosis, it needs good system visibility. The more measurements available, the better the shell (or an engineer) can detect the presence of a misbehaving component and determine exactly which component it is.

FUTURE

KATE could be applied to, and could have a positive impact on, many projects at the Space Center as well as in other control and monitor applications. What follows is a description of some possible application areas as well as a few future KSC projects and how they could benefit from model-based technology:

Ground Data Management System (GDMS)- GDMS is a major project being developed at KSC to support interface verification, integration and test activities for the various elements of the Space Station, from prelaunch on, at the Space Center. It will process and test Space Station "work package" items, interfaces broken for shipping, interfaces between different launch package items, and interfaces between the launch package and the orbiter. GDMS consists of a large multi-channel network system, each channel capable of handling about ten times as many commands and measurements as the current LPS. Each channel can support up to 256 mini/Mainframe computers acting as application processors and up to 256 engineering workstations for graphics and display processing. Each processor will also have access to large archival and retrieval subsystems, database systems,

*The LOX replenish valve is a very redundant subsystem which has around 20 components and 12 commands and measurements associated with it.

documentation processing, and other networks via gateways and bridges. The KATE shell could be the application doing the test and check-out job on the application processors. Since the elements will probably be very different from one another and will have to be tested before integration, trying to do so with conventional software technology would require a tremendous effort. Even if this is possible, the effort would not be applicable to other elements. With the knowledge-based approach, because only the knowledge base would have to be developed by the experimenter or vendor (probably from a CAD system), application code would never have to be written.

Heavy Lift Launch Vehicle- A check-out and control system will need to be created for the future heavy lift launch vehicle(s) being planned by NASA and the Air Force. The types of test and check-out tasks that need to be performed for this will be very similar to those done by LPS (and GDMS) and therefore could benefit greatly by employing some of the ideas in the KATE shell.

Electric Power and Research Institute (EPRI)- Currently, a NASA technology transfer program is underway with EPRI to apply the KATE shell to problems in the power industry and nuclear power plants in particular. Early investigation and interviews with plant managers have identified many areas where KATE could offer substantial savings and safety features.

Other Applications- Other areas which could benefit from the approach taken in KATE include on-board aircraft, ship and spacecraft systems, the Space Station, satellites and the National Aerospace Plane.

Conclusion

We feel that a knowledge-based control and monitor approach, such as exemplified by KATE, may be the only solution to problems where the systems under test are well understood, can be described in a structure-function format and it is desirable to do automated control, monitor, diagnosis, drawing, sensor validation and redundancy management on them. For large systems requiring diagnostics, traditional tree-structured methods are impractical, for identifying all possible failure states can easily become an impossible task. Having the low-level control performed automatically by the shell provides the basis for a very high-level procedural language which would eliminate or greatly simplify application program writing as we know it today, saving development time and money and providing features impossible to achieve with current methods.

REFERENCES

1. Davis, Randall, "Diagnostic Reasoning Based on Structure and Behavior", Qualitative Reasoning about Physical Systems, First Ed., MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1986, pp. 347-410.
2. de Kleer, Johan, "How Circuits Work", Qualitative Reasoning about Physical Systems, First Ed., MIT Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1986, pp. 205-280.
3. Jamieson, John, "Single Point Failures at KSC", unpublished, DL-DSD-22, Kennedy Space Center, Florida, 32899.
4. Scarl, Ethan, Jamieson, John, Delaune, Carl, "Sensor-based Diagnosis using Knowledge of Structure and Function", IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-16 No. 6, November, 1986.
5. Scarl, Ethan, Jamieson, John, Delaune, Carl, "A Fault Detection and Isolation Method Applied to Liquid Oxygen Loading for the Space Shuttle", Proceedings 9th International Joint Conference Artificial Intelligence (IJCAI-85), pp. 414-416.
6. Roberts, R.B., Goldstein, I.P., "The FRL Manual", Massachusetts Institute of Technology AI Laboratory, Memo 409, September, 1977.

ACRONYMS

ECS	Environmental Control System
EPRI	Electric Power and Research Institute
FD	Function Designator
GDMS	Ground Data Management System
GOAL	Ground Operations Aerospace Language
KATE	Knowledge-based Autonomous Test Engineer
KSC	Kennedy Space Center
LES	LOX Expert System
LOX	Liquid Oxygen
LPS	Launch Processing System
OMRF	Orbiter Maintenance and Refurbishment Facility
SGOS	Shuttle Ground Operations Simulator

An Expert System for Design of Digital Autopilots

(Paper not provided by publication date)

BEYOND RULES
THE NEXT GENERATION OF EXPERT SYSTEMS

Jay C. Ferguson/Robert E. Wagner
Ford Aerospace & Communications Corporation
1260 Crossman Avenue
Sunnyvale, California 94089-1198

ABSTRACT

This paper introduces the PARAGON Representation, Management and Manipulation system. The concepts of knowledge representation, knowledge management, and knowledge manipulation are combined in a comprehensive system for solving real-world problems requiring high levels of expertise in a real-time environment. In most applications the complexity of the problem and the representation used to describe the domain knowledge tend to obscure the information from which solutions are derived. This inhibits the acquisition of domain knowledge, the capability to perform knowledge verification/validation, places severe constraints on the ability to extend and maintain a knowledge base while making generic problem solving strategies difficult to develop. Ford Aerospace has pioneered a unique hybrid system to overcome these traditional limitations.

INTRODUCTION

This paper introduces the PARAGON Representation, Management and Manipulation system. The concepts of knowledge representation, knowledge management, and knowledge manipulation are combined in a comprehensive system for solving real-world problems requiring high levels of expertise in a real-time environment. In most applications the complexity of the problem and the representation used to describe the domain knowledge tend to obscure the information from which solutions are derived. This inhibits the acquisition of domain knowledge, the capability to perform knowledge verification/validation, places severe constraints on the ability to extend and maintain a knowledge base while making generic problem solving strategies difficult to develop. Ford Aerospace has pioneered a unique hybrid system to overcome these traditional limitations.

To address these problems, Ford Aerospace has developed a model-based paradigm which is realized in a system called PARAGON. PARAGON consists of three major areas: Knowledge Representation, Knowledge Management, and Knowledge Manipulation. Knowledge Representation is the foundation of PARAGON. The knowledge representation determines how you manage and manipulate the knowledge of the domain. A hybrid representation scheme was chosen that integrates frames [Minsky], semantic networks [Quillian], classification hierarchies [Quillian], blackboards [Hayes-Roth], demons [Waterman], transition networks [Petri, Woods], and rules [Shortliffe]. Knowledge Management encompasses the acquisition of the domain models the translation of the models into the knowledge representation, and the verification and validation of the domain models. This is accomplished through graphic interfaces that provide a framework to develop, maintain and access the knowledge base. Knowledge Manipulation provides a modular set of generic problem solving modules [Clancey) that can be combined to solve different types of problems within a domain.

KNOWLEDGE REPRESENTATION

Ford Aerospace has designed a formalism in which to express knowledge about the world by modeling the structure and function of the concepts that are part of the problem we are trying to solve. This formalism evolves the classification/rule based representations to a more explicit, consistent and robust structured conceptual network representation [Sowa, Mylopoulos]. Rule based systems lack the explicit structure for expression of descriptive hierarchical and temporal knowledge. Classification systems attempt to rectify this situation by providing a hierarchical structuring mechanism to express the knowledge. These systems have had limited success because they lack a methodology to direct and dictate the structure of the

knowledge. This has resulted in a mixing of completely different types of knowledge within the same structure. Both types of systems, rule and classification, have been combined in new representation systems [Fikes, Stefik]. These hybrid systems solved some problems, created new problems and left many problems unaddressed.

Although current tools incorporate frame-based classification techniques with rule-based knowledge representation mechanisms, they fail to provide generic problem solving techniques. These mechanisms allow more generic algorithms but lack the structure and information required to solve problems generically. Although consistent, these representations lack an underlying principle from which a description of the domain can be derived. In a rule-based system there is no structure to dictate what may be used in the context (antecedent) or the action (consequent) of the rules. This flexibility allows a completely ad hoc development to occur.

The same situation exists for frame-based classification systems. There are no guiding principles or constraints on what may be an object, what it may contain, the structure of the classification hierarchy, or how objects communicate with each other. Conceptually this level of freedom is appealing; however, without the required structure and constraints large systems quickly become unextendable, impossible to validate and difficult to maintain [McDermott]. Because these areas are handled in an ad hoc manner there is not enough consistency to apply generic problem solving techniques to these representational systems. This inconsistency also causes problems with the ability to acquire knowledge through generic tools.

PARAGON is a modeling paradigm used as the conceptual basis for describing the domain. This paradigm is realized in a hierarchical knowledge representation that allows concepts to be defined, their inter-relationships specified and their behavior described within a dynamic conceptual network. The network consists of two types of entities, conceptual entities (the nodes) and relational entities (the links). Both entities are defined in a classification hierarchy. Each class of entities is characterized by a specific definition that describes the behavior (semantics) of the entity.

This approach allows the structure in the domain to be directly represented within the computer in a generic and consistent manner that corresponds to the way people perceive the domain. The domain is structured along five dimensions (figure 1); definition, composition, functional relationships, structural relationships, and sequential behavior. Each relationship has a well-defined behavior that describes how information is propagated between associated concepts. This allows knowledge to be described in a modular and well defined manner at varying levels of granularity.

Figure 2 is a slice of the representation that displays the major concept and relationship types within the system. This structure makes the efficient management of large amounts of complex knowledge possible through a set of knowledge acquisition interface tools. The concepts of generalization, abstraction, and cause/effect are combined in a uniform and consistent manner. Many existing representation techniques are integrated to provide consistency for both the representation and manipulation of the domain information.

Concepts are the building block of the system. Each concept can have a set of attributes that describe it as an individual entity. Primitive concepts organize the local attributes, inter-relationships and behavior of instances [Quillian]. Abstract concepts are defined through the aggregation of other concepts [Sowa]. Abstract concepts hide the internal details of their components while maintaining the external relationships to other concepts. Definitional concepts are a covering set for the specializations that belong to the concept [Brachmann]. Definition concepts are used to store generic information and to create new specializations based on the definition of their current specializations. State concepts specify the context in which particular facts are currently true. Event concepts specify how to determine the attribute values in a state and the actions that may occur. Concepts are inter-related with other concepts through functional, structural and temporal relationships.

In order to describe behavior we have developed a representational methodology to model temporal knowledge within a declarative framework. Sequential descriptions are used to represent processes and to model the dynamic behavior of concepts within a domain. When this information is represented in code it is not accessible for explanation or reasoning. Similar to the process of moving situation/action knowledge from code into rules, this methodology is opening the black-box of code and making declarative knowledge available to solve the problems within a domain. By formalizing the representation of procedural knowledge, generic algorithms can be developed to manipulate,

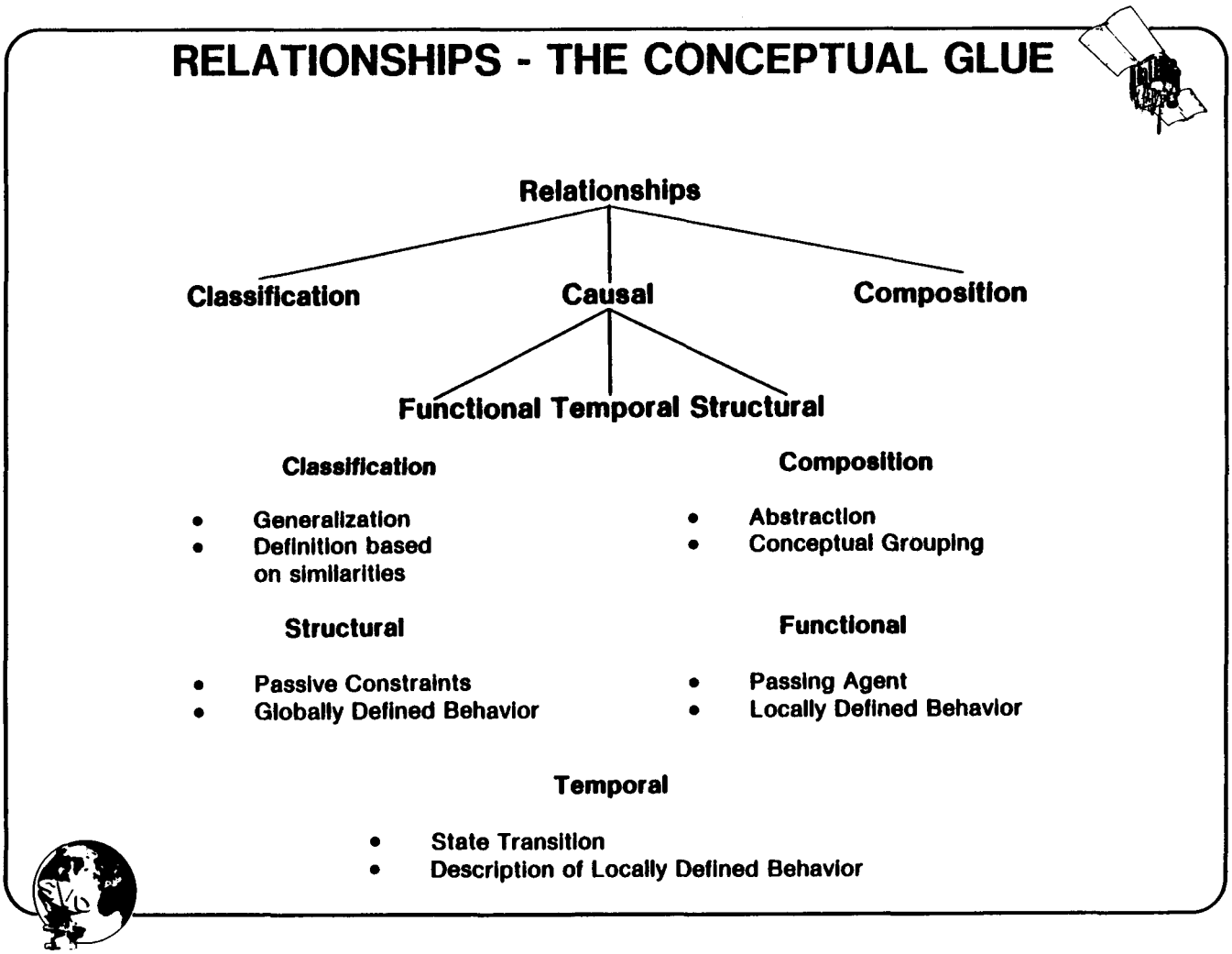


Figure 1. Relationships in PARAGON

reason about, and explain temporal events.

A process is described by a set of states, the conditions in which a state transition may occur and tne events that take place in each state. An event in a process is used to compute the value of attributes local to the concept that tne process describes. PARAGON uses a theory of LOCALITY that defines the inter-communication of independent parallel processes and constrains access to information by requiring explicit causal relationships to be defined between the communicating processes [Hoare]. Events can only affect local attributes. Events can access external information in their computations through the explicit causal relationships associated with the concept. The conditions in which a state transition can occur are constrained by the same mechanism. Processes maintain temporal histories based on the

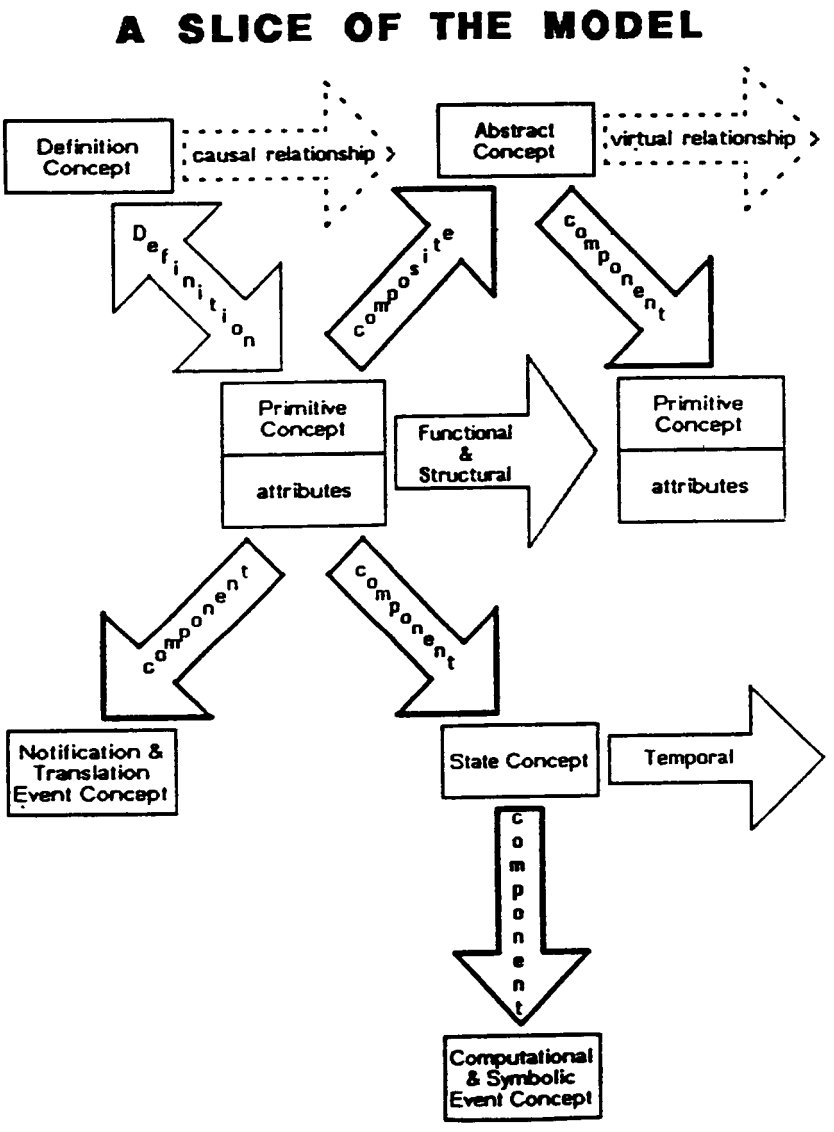


Figure 2. PARAGON Representation

hierarchical structure of the process [Kahn]. This data base can be used for various forms of temporal reasoning. An outcome of the theory of LOCALITY is that each process is a self-contained description that has no side-effects. Information is passed between concepts on an access only basis. These properties lead to a highly programmable parallel processing system.

This methodology combines both discrete and continuous processes to create a hybrid model-based representation system. Techniques have been developed to simplify complex processes and events through equation segmentation and linear approximation. This amounts to reverse engineering knowledge from mathematical equations. Both code-based procedure descriptions and equation-based computations are explicitly represented in a consistent descriptive representation.

KNOWLEDGE MANAGEMENT

We have also developed a set of graphical interfaces that are used to map the domain concepts, structure, inter-concept and intra-concept relationships, and temporal behavior into the PARAGON knowledge representation system. The knowledge acquisition tools are a reflection of the underlying model-based knowledge representation. This allows direct translation of the acquired data into the knowledge base through generic algorithms while maintaining a high degree of correspondence between the data and the world view of the knowledge engineer (cognitive resonance).

The purpose of these tools is to allow experts with minimal training to enter knowledge into the system while capturing enough information to solve problems that occur within the domain. Each tool is a graphical interface that provides the expert with a framework in which he can enter the various types of knowledge. This framework is obtained through generalization, abstraction and the inter-relationship of concepts. Users are allowed to define specializations of relationship classes to specify specific semantic and communication information in a particular domain. Various methods have been developed to efficiently organize, index and access the large amounts of information contained in the knowledge base.

Knowledge management consists of the acquisition, validation and maintenance of the information that is used to solve the problems in the domain. In most expert systems there is no way to acquire the information through a generic mechanism. By using a modeling approach to expert systems we are able to acquire the information through graphic interfaces from knowledge engineers untrained in artificial intelligence or programming. This knowledge can then be validated based on the consistency of the representation and tested through simulations using the model description. The knowledge engineer has the ability to define complex concepts by grouping primitives through composition.

Input through the graphic interface provides a consistent and well organized knowledge base that can be extended and maintained. This is possible because PARAGON is representing information in a well defined and structured methodology that emphasizes the semantic instead of syntactic level. This is sometimes referred to as deep vs. shallow models [Genesereth, De Kleer]. Most expert systems use information in the form of rules and frames. The frames are used to store the facts and operations that can be performed on the facts (LISP programs). The rules are used to represent contextually dependent information, control sequences and contextually dependent behavior. All of this information is stored at a syntactic level. There is no well-defined methodology used to process the data. Programs are used to give the words in the rules and frames some meaning. The meaning is based on syntactic patterns of characters that match other patterns of characters. Most of the time the meaning in rule based expert systems is derived by the person looking at the rules and not the rules themselves.

Knowledge acquisition is the process of entering knowledge about a domain into the computer and translating the knowledge into an internal representation [Musen, Tsuji]. This area has traditionally been a bottleneck for building expert systems. The problem has been that most application systems are not understood or understandable at the level of heuristic rules. Rules do not offer any framework in which a knowledge engineer can work. The knowledge engineer must build the framework himself by trial and error. Because rules offer no structuring mechanism this task becomes increasingly difficult as the system grows.

By approaching the task from a modeling perspective the knowledge engineering becomes feasible with generic tools. PARAGON allows the knowledge engineer to define the concepts in the domain based on actual examples. Because the knowledge engineer has samples of the concepts in the domain available, this method provides a framework to develop and extend the definition

of the model. The examples are then generalized through a process of shared attributes to create definitions for the engineer to reuse in defining other similar concepts in the system. This significantly reduces the knowledge engineering task of entering redundant information, and provides a framework that the knowledge engineer can use to more easily define concepts. This methodology of definition by example also provides a generic knowledge base that can be used for other systems.

KNOWLEDGE MANIPULATION

The ultimate goal of modeling a domain is to allow questions to be asked and problems to be solved within the domain. Application domains frequently require a wide variety of problem solving categories [Clancey]. In Diagnostics and Repair domains [Ferguson, Siemens] the problem solutions require determining the current state of the system (classification, situation assessment and interpretation), determining what caused the system to be in the current state (diagnostics) determining what state(s) the system should/could be in for it to function correctly (goal determination), how to get more data about the system (testing) how to actually fix the system (planning & command), and what the future state of the system might be (prediction). Each of these areas is extremely complex by itself. To be able to deal with all of these areas within the same consistent representation is beyond any of the individual representation techniques or expert system shells available today.

PARAGON uses the dynamic memory model [Schank] applied to the domain model for event recognition (expectation failure) and causal diagnosis (explanation through accountability). Spreading activation [Quillian] on the domain model is used for interpretation. The behavioral specification and causal relationships are used for goal determination, prediction, planning and command. By having an internal deep model of the structure, behavior and causal inter-relationships we have found that we can solve many of the difficult problems within the modeling framework.

PARAGON has the advantage of access to a model of the system that it is reasoning about. This allows the knowledge manipulation algorithms to access the causal pathways and the behavioral aspects of the system. Because of the consistent and formalized nature of our representation, we can apply generic problem solving algorithms to the model independently of the domain. Another major advantage is the reduction of search through indexing techniques. Because the model is used to focus on small, localized sets of components, the search eliminates most of the model immediately. This allows more time to be spent dealing within a restricted environment to solve the problem. This modeling paradigm can be used as the basis for complex, real-time expert systems. Because of the capabilities of the knowledge representation language to model the different dimension of a domain, many types of reasoning can be performed using generic algorithms.

CONCLUSION

We have described a new paradigm to solve problems using a domain model and generic acquisition, representation and manipulation methods. The domain independent capabilities of PARAGON are derived from the use of robust and consistently defined behavioral and structural descriptions to model knowledge at the semantic level. Defining how information is propagated through various relationships and constraining how concepts can be related achieves a uniform and consistent set of semantics. PARAGON also has the unique ability to represent temporal and behavioral knowledge which can be used for all classes of problem solving. By making this information available in a declarative formalism, the level of reasoning able to be performed by the system moves from a shallow level to a deep level. At the same time the models that are constructed are natural to the domain engineer. This makes the task of definition, validation, and maintenance of large & complex domain knowledge base feasible. The most important feature of PARAGON however, is the ability to develop a specifiable methodology for the construction and testing of expert systems. Without such a methodology, construction of expert systems would only be applicable to small problems by a limited supply of highly trained knowledge engineers. All these features of PARAGON have combined to make building large expert systems a reasonable task.

ACKNOWLEDGEMENTS

This work is the result of a combined effort of the Artificial Intelligence Group at the Sunnyvale Division of the Ford Aerospace & Communications Corporation (Marilyn Golden, Alain Rostain, Ron Siemens, Dennis Heher, Tihmer Toth-fejel, Vince Contreras, Paul Pownall) and the progressive management that allowed it to happen.

REFERENCES

1. Brachmann, R.J., On the Epistemological Status of Semantic Networks, in Associative Networks: Representation and Use of Knowledge by Computers, pp. 3-50, edited by N.V. Findler, Academic Press, 1979.

2. Clancey, W.J., Heuristic Classification, AI Journal, Vol. k27 Number 3, Dec., 1985, pp. 289-350, North-Holland, Amsterdam.

3. Contreras, V., Ferguson, J., Knowledge Management: An Introduction to the Concept of Knowledge Management as a System for Controlling the AI Development Environment for an Expert System which Performs Satellite Anomaly Resolution, Mil Com 86, 1986

4. De Kleer, J., Brown, J.S., A Qualitative Physics Based on Confluences. Qualitative Reasoning About Physical Systems. MIT Press. 1985, pp. 7-83.

5. Ferguson, J.C., Siemens, R.W., Wagner, R.E., STAR-PLAN: A Satellite Anomaly Resolution and Planning System, Proceedings of AAAI Workshop on Coupling Symbolic and Numerical Computing in Expert Systems, August, 1985.

6. Fikes, R., Kehler, T., The Role of Frame-Based Representation in Reasoning, Communications of the ACM, Vol. 28 Number 29, Sept. 1986.

7. Genesereth, M.R., The Use of Design Descriptions in Automaed Diagnosis, Qualitative Reasoning about Physical Systems, D.G. Bobrow, pp. 411-436, MIT Press, Cambridge, Mass.

8. Golden, M., Siemens, R., Ferguson, J., Whats Wrong With Rules?, Westex 86.

9. Hayes-Roth, B., The Blackboard ARchitecture: A General Framework for Problem Solving?, Stanford University HPP Report HPP-83-30, 1983.

10. Hoare, C.A.R., - Communicating Sequential Processes. 1985, Prentice-Hall.

11. Kahy, M.G., Ferguson, J.C., Shortliffe, E.H., & Fagan, L.M., An Approach for Structuring Temporal Information in the ONCOCIN System, Proceedings of the Symposium on Computer Applications in Medical Care, 1985.

12. Minsky, M., Matter, Mind & Models in Semantic Information Proceedings, Edited by Marvin Minsky, MIT Press, Cambridge, Mass., 1968.

13. Musen, M.A., Fagan, L.M., Combs, D.M,., Shortliff;e, E.H., Facilitating Knowledge Entry for an Oncology Advisor Using a Model of the Application Area, Stanford University Technical Memo KSL-86-1.

14. Mylopoulos, J., Brodie, M., Schmidt, J., On Conceptual Modeling. Springer-Verlag, 1984.

15. Petri, C.A., Fundamentals of ;a Theory of Asynchronous Information Flow. Information Proceedings 1962, Proceedings of the IFIP Congress 62, Munich. North Holland Publishing Company, Amsterdam, pp. 386-390.

16. Quillian, M.R., Semantic Memory, edited by M. Minsky, Semantic Information Processing, Cambridge, Mass., MIT Press, pp. 216-270.

17. Schank, R.C., Dynamic Memory: A Theory of Reminding and Learning in Computers and People, Cambridge University Press, 1982.

18. Shortliffe, E.J., Computer-based Medical Consultation: MYCIN. American Elsevier, New York, 1976.

19. Siemens, R., Golden, M., Ferguson, J., StarPlan II: Evolution of an Expert System, The National Conference on Artificial Intelligence, 1986.

20. Sowa, J.F., Conceptual Structures. Addison-Wesley, 1984.

21. Tsuji, S., Shortliffe, E.H., Graphics for Knowledge Engineers: A Window on Knowledge Base Management. Stanford University Memo KSL-85-11.

22. Waterman, D.A., Hayes-Roth, F., Pattern-Directed Inference Systems, Academic Press, 1978.

23. Woods, W., Whats in a Link?, Foundations for Semantic Networks. In Representations and Understanding. Academic Press, pp. 35-82.

A CONCEPTUAL FRAMEWORK FOR INTELLIGENT REAL TIME INFORMATION PROCESSING

Robert Schudy

BBN Laboratories Incorporated
10 Moulton Street
Cambridge, Massachusetts 02238

ABSTRACT

By combining artificial intelligent concepts with the human information processing model of Rasmussen, we have developed a conceptual framework for real-time AI systems which provides a foundation for system organization, control and validation. The approach is based on the description of system processing in terms of an abstraction hierarchy of states of knowledge and processing functions which connect those states of knowledge. The states of knowledge are organized along one dimension which corresponds to the extent to which the concepts are expressed in terms of the system inouts or in terms of the system response. Thus organized, the useful states form a generally triangular shape with the sensors and effectors forming the lower two vertices and the full evaluated set of courses of action the apex. If the representations and processing steps in the slopes of this pyramid are correct and complete, them the processing sequence from inputs to outputs following the slopes of this pyramid results in correct behavior. Unfortunately, this path is generally too computationally expensive to be performed in real time, either by natural or artificially intelligent systems. Within the boundaries of the triangle are numerous processing paths which shortcut the detailed processing, by connecting incomplete levels of analysis to partially defined responses. Shortcuts at different levels of abstraction include reflexes, sensory-motor control, rule-based behavior, and satisficing. The correctness of shortcuts depends on whether the response inferred on the processing shortcut is consistent with the responses which would have been inferred by the computations which are being shortcut. By clarifying assumptions, relationships, and the knowledge and processing which is being approximated, this approach provides a foundation for knowledge acquisition, system design, and system validation. We have used this approach in the design of a real-time tactical decision aiding system, in defining the requirements for an intelligent aiding system for transport pilots, and in the design of an autonomous system.

Figure 1 schematically describes the overall processing steps for a real-time intelligent agent such as a pilot, an autonomous system, or a real-time aiding system. The vertical, abstraction, axis in the figure corresponds to the extent to which the processing is removed from the concrete sensors and effectors. Movement along the horizontal axis represents progression of the processing from the sensors to the effectors. The ovals in the figure represent states of knowledge, and the labelled arcs between those states represent processing which implements the indicated transitions between the states of knowledge. Different knowledge representation requirements are associated with the different states of knowledge, and different computational requirements are associated with the different classes of transitions. The particular states of knowledge and processing steps should not be interpreted as required functional partitions for particular systems. The actual flow of processing in nontrivial systems involves many more stages than are shown, and results will generally be used by several processing levels to the right of where they are produced. Many more pathways may exist between the indicated states of knowledge, and other states of knowledge may be important for some applications; these other pathways are summarized by the three classes of shortcuts listed within the three large arrows.

1. PROCESSING WITHOUT SHORTCUTS

The discussion in the next 21 paragraphs follows the information flow in Figure 1 from sensors to effectors, taking no shortcuts, and alternating between states of knowledge and the processing steps which accomplish the transformations between those states.

Sensors. The sensor state of knowledge is just the sensor data and other inputs to the intelligent agent. This input data includes "self" information from sensors such as kinesthetic sensors (e.g. actuator positions, velocities, stresses, and strains), system status data (e.g. self-test and built-in-test results), and internal environmental sensor data (e.g. temperatures, pressures, accelerations, flow rates, fuel quantity, vibration, and power supply voltages). This input data also includes environmental information communicated from other agents.

Perception. Perception is the mapping of the sensor data into symbolic descriptions of the entities and states of relevance to the intelligent agent. This signal-to-symbol processing includes of all vision processing up to and including the level of scene description, and all diagnostic processing up to the level of system status.

Entity and State Descriptions. The output of perceptual processing is a symbolic description of the state of the agent and its environment. These descriptions could be represented in terms of instances of archetypical entities and states, with the attributes refined from the sensor data. The state descriptions include relationships between entities, when those relationships can be derived from the input data.

Assessment. Assessment includes the processing stages in which the significance of the current situation is derived, in terms of the impact of the situation on the agent, its goals, plans, and actions. Assessment processing proceeds

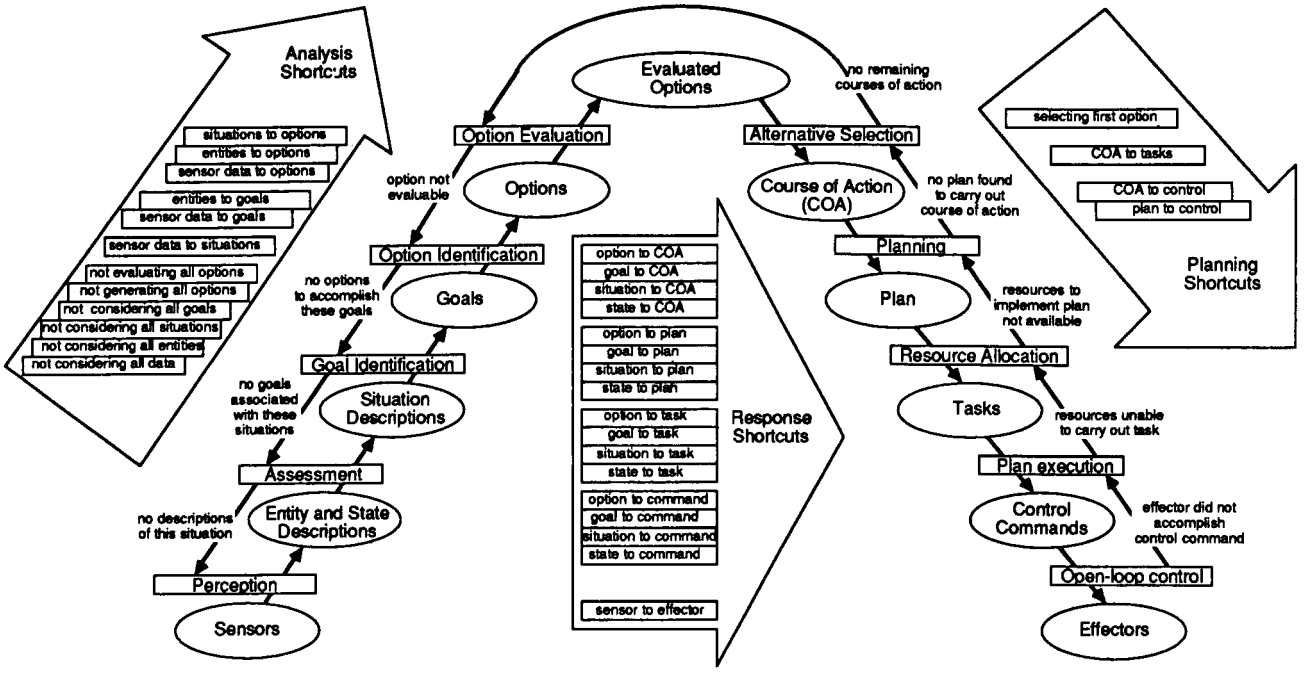


Figure 1. General Information Processing Model with Shortcuts

through the generation of successively more abstract descriptions of the situation attributes and of the situation as a whole.

Situation Descriptions. Situation descriptions are a proper superset of entity and state descriptions. Because of the scope of situation descriptions, it may no longer be feasible to describe situations in terms of single situation types. Thus situation descriptions may take the form of interrelated instances of different situation types, each of which describes some aspect of the total situation.

Goal Identification. Goals Direct and focus behavior. Thus while goals may enter at any level before this stage, they must be defined by the end of this stage, so that the goals can be used to drive the option identification step. While very general static goals such as survival may be useful in strategic planning, more specific and more near-term situation-dependent goals are more effective in driving behavior. Goal identification proceeds from assessments of the situation and from more global goals contained in the plan structure. Since different aspects of the situation may dictate different goals, resolution of goal conflicts may be required.

Goals. Goals combine descriptions of future states and situations and statements of the desirability of those states and situations. Thus the goal representation language is a proper superset of the situation representation language. Since goals involve not only states, but also situations, actions, and the temporal and other relationships between states and situations, the representation for goals shares many of the characteristics of plans.

Option Identification. With descriptions of the state, situation, and goals, feasible behavioral options can now be generated. For the overall processing to be optimal, in the sense that the overall behavior is the best for attaining the goals, this processing step must identify all options which may be optional; however not all options identified need be optimal.

Options. Options are descriptions of potential courses of action for the intelligent agent. Option descriptions may include state and entity descriptions, situation descriptions, and goals. Thus the option representation language is a proper superset of the goal representation language.

Option Evaluation. The option evaluation step is the detailed application of the goal criteria against the identified options. If the options are parametric, then evaluation can involve generating the members of the sets represented by the parametric options. This can in principle produce any number of options to be evaluated. Unless the optimality criteria impose a total ordering on the courses of action, this set may not be finite. The technologies of heuristic search differential game theory, and operations research can each address option evaluation problems in different domains, but no efficient general methods for option evaluation are currently known.

Evaluated Options. Unless powerful abstractions, heuristics, and other means are available for representing and pruning the space of options, usually only a small set of the whole family of options can be evaluated. The evaluated option set may include all of the representations in the option representation, plus information, such as total or partial orderings, and evaluative criteria, resulting from the evaluation of those options. In order that the behavior be optimal, all attributes of evaluative interest must be captured in the representation of the options. Thus subsequent representations may be restrictions of this representation of options, and can only enlarge on the evaluated options representation by expansion of abstractions contained in the evaluated options, and only then when the evaluation of those options does not depend on the expanded detail.

Alternative Selection. The alternative selection process may be trivial, if the goals impose a total ordering on the options, or it may involve the invocation of additional criteria to select amongst an equivalence class of best options.

Course of Action. The course of action is a subset of the evaluated options. If the agent is operating in an uncertain

environment involving unpredictable agents, or other sources of uncertainty, then the course of action may include branches to accommodate the generically different responses during plan execution. Thus the course of action, like the options, evaluated options, plans, and tasks, may be represented as a semilattice, with a root representing the current state, and nodes representing branches and joins of tasks and situations during possible plan executions. The course of action may still involve free parameters, or set-valued parameters, if the values of those parameters was not required to evaluate the option.

Planning. The planning process is the expansion of the course of action to the level of the operations to be performed. Note that many of the computational steps which would be considered as steps in planning, such as option identification and evaluation, may have already been performed to support option evaluation and the selection of the course of action.

Plan. The plan representation may include entity and state representations, situation representations, decision criteria, and alternative options. From a representational standpoint it is thus equivalent to the evaluated options representation. The plan differs from that representation in two ways. First, it is uniformly expanded to the level necessary to support resource allocation processing; it thus must include all parallel operations, and detailed models of timing. Secondly, the plan includes only those options which passed the alternative selection process. The plan is thus more detailed than the course of action, but includes fewer options than the evaluated options.

Resource Allocation. Resource allocation binds specific resources to specific actions in the plan. Some resource conflicts may have been resolved earlier in the processing to evaluate the course of action and to develop the plan; the resource allocation process completes those allocations. Resource allocation processing completes the development of task descriptions to support plan execution.

Tasks. The task representation is the plan representation expanded to the level of specific tasks for specific resources. Task descriptions include any parameters, other than inputs, required by the execution procedures. These task descriptions include sensing tasks, and decision tasks, information processing tasks. It may also include enabling or initiating processing shortcuts such as servo control processes.

Task Execution. Task execution follows the procedure for tasks of that type. This involves task initiation, performance monitoring, and parameter adjustment. Task execution can be influenced by the results from decision tasks, and by feedback from other running tasks.

Control Commands. The output of the task execution processing are control commands to the effectors and other resources. These control commands include all information other than input data necessary to determine the operation of those resources. The specific data depends on the resource controlled.

Control. Control processing may involve servo-loops, or can operate with limited or no feedback.

Effectors. Effectors include all resources under control of the agent, including information processing, actuators, and sensors.

If all of the processes and representations described above are complete and correct, and they are to run completion, then no feedback is required for the system to produce correct behavior. However, efficiency can be improved by implementing feedback from later processing stages to control processing at earlier stages. This cascading can take the form of the integration of later functions into earlier functions, where it can help narrow the number of options considered, or it can take the form of specific information fed back from later stages when those stages have examined partial results those earlier stages. An example of the processing of the first sort is the integration of resource allocation constraints into the option generation and evaluation stages; an example of the second sort is feedback from the alternative selection function about partially evaluated options.

2. SHORTCUT PROCESSING

Figure 1 shows three broad classes of shortcuts - those of analysis (intent formation), those of intent execution, and a class of "response" shortcuts which move at various levels of abstraction from left to right, bridging from partial analyses to skeletal responses. We currently believe that response shortcuts are a predominant pathway in human information processing. The response shortcuts range in abstraction from simple servo-control laws linking sensors and effectors, up through satisficing [Simon, 1969], which shortcuts exploring all of the courses of action. Shortcuts at an intermediate level of abstraction are particularly important in real-time decision making. Situation-response shortcuts, which move rightward from the vicinity of the *situation* description level of analysis to the vicinity of the *tasks* level of execution, are described in the companion paper *A Situation-Response Model for Intelligent Pilot Aiding.*

Situation-response processing has a role in the less abstract skill-based behavior. The pathways in Figure 1 for skill-based shortcuts proceed nearly horizontally from the vicinity of sensing and perception to the vicinity of the effectors. The establishment of such skill-based shortcuts reduces workload and reduces processing delay by uncoupling the situation assessment process from the process of adapting to changing situational parameters. In our model, the activation and management of skill-based behavior is one of the normal functions of rule-based behavior. The situation assessment function then assumes the role of enabling the execution of the skill-based behavior.

Both the risks and speed of such shortcut processing are well known. Interviews with airline pilots revealed the following example: During aircraft takeoff roll, the pilot-flying noticed a fluctuation in right engine readings. Just before takeoff velocity the pilot-flying heard a loud boom and felt the plane vibrate. Reflexively he reached to shut down the right engine (having mentally established its potential for failure). The pilot-not-flying stopped this move because he had determined that it was the left engine that had failed. Thus, the processing shortcut (enabled by the assessment the situation as a possible-right-engine-problem) resulted in a rapid, but inappropriate response.

Situation-response processing also has an important role in the more abstract knowledge-based behavior. The normal pattern is that when the limits of situation-response processing is reached, knowledge-based reasoning is initiated. Then either the situation changes while the knowledge-based processing takes place, or the knowledge-based processing produces a useful result. In either case the processing reverts to the situation-response model. Note that in this model situation-response processing serves as an input filter for knowledge-based processing, guaranteeing that the scarce knowledge-based processing resources are only invested in unusual and hence presumably fruitful problems.

3. CONCLUSIONS

Real-time performance in natural or artificially intelligent systems depends on using the lowest feasible levels of abstraction. Processes operating at different levels of abstraction can be organized so that correct behavior can be implemented using information processing shortcuts. The shortcut and other levels of processing can be organized so that less abstract processes develop abstractions for more abstract processing, and more abstract processing supervises the less abstract. The development of shortcuts in the information processing abstraction hierarchy is a basic mechanism for learning to perform more quickly.

4. References

Simon, H.S., The Sciences of the Artificial, M.I.T. Press, 1969.

Rasmussen, J., Skills, Rules, Knowledge; Signals, Signs and Symbols; and other Distinctions in Human Performance Models. IEEE Transactions of Systems, Man, and Cybernetics SMC-13 (3), 1983.

Rasmussen, J., Strategies for State Identification and Diagnosis in Supervisory Control Tasks, and Design for Computer-based Support Systems. In (Rouse) Advances in Man-Machine Systems Research, Vol. 1 pp. 139-193, 1984.

TDAS
THE THERMAL EXPERT SYSTEM (TEXSYS) DATA ACQUISITION SYSTEM

Edmund C. Hack
Lockheed-EMSCO
2400 NASA Road 1
P. O. Box 58561
Houston, Texas 77258

Kathleen J. Healey
Chief, Intelligent Systems Branch
NASA
Johnson Space Center
Houston, Texas 77258

Abstract

As part of the NASA Systems Autonomy Demonstration Project, a thermal expert system (TEXSYS) is being developed by the Ames Research Center, the Johnson Space Center (JSC) and their contractors. TEXSYS combines a "fast" real time control system, a sophisticated human interface for the user and several distinct artificial intelligence techniques in one system. TEXSYS is to provide real time control, operations advice and fault detection, isolation and recovery capabilities for the Space Station Thermal Test Bed (TTB) at JSC. TEXSYS will be integrated with the TTB and act as an intelligent assistant to thermal engineers conducting TTB tests and experiments.

This paper will present the results of our work on connecting the real time controller (running on a conventional computer) to the knowledge based system (running on a symbolic computer) creating an integrated system. Special attention will be paid to the problem of filtering and interpreting the raw, real time data and placing the important values into the knowledge base of the expert system.

Introduction

The increasing complexity of space missions to be conducted in the next twenty-five years by the United States requires that the control and monitoring systems used to support these missions take advantage of the latest in automation technologies to reduce costs and more importantly, increase reliability and safety. The Space Station with its longer than 20 year operation lifetime and the proposed Mars sample return mission will both need to exploit the recent advances in artificial intelligence to perform their mission, most notably expert systems. This need was recognized by the National Commission on Space (aka the Paine Commission) whose final report suggests that:

> ...NASA explore the limits of expert systems, and tele-presence or tele-science for remote operations, including ties to spacecraft and ground laboratories.

Recognizing these needs, the National Aeronautics and Space Administration's Office of Aeronautics and Space Technology has initiated the Systems Autonomy Technology Program. The program consists of two programs, with the Core Research and Technology Program funding basic and

initial applied research in AI and automation. In the second program, the Systems Autonomy Demonstration Project (SADP), a series of four demonstrations will be conducted to show the capability of increasingly complex expert systems applied to Space Station subsystem needs.

The Thermal Expert System

The first of these demonstrations, the Thermal Expert System (TEXSYS) is being developed to show the use of artificial intelligence technology in operation and management of the prototypes for the Space Station Thermal Control System (TCS). This demonstration is a joint project of the Ames Research Center (ARC) and Johnson Space Center (JSC). TEXSYS will be used to monitor, control and diagnose faults in the Thermal Test Bed (TTB) at JSC.

The TTB program is designed to test ground-based engineering models of the Space Station active thermal management control system, to verify the readiness of two-phase thermal technology and to provide system level evaluation of advanced thermal control technology for Space Station use.

TEXSYS will incorporate the following features:

- Monitor and control of a single thermal subsystem
- Goal and causal explanation displays
- Qualitative and quantitative simulations
- Fault detection, diagnosis and limited reconfiguration
- Reasoning using standard procedures

While the features to be included in TEXSYS are well established, the combination of these AI techniques in a single system to be used in a real time environment will provide NASA with a benchmark system to test new ideas and further refine them for mission applications.

Hardware

TEXSYS will be demonstrated using a prototype of the Space Station central thermal bus. A simplified diagram of the central thermal bus is shown in figure 1. This central thermal bus has a conventional computer system for data collection and control. The control system is a commercial software package, FLEXCON, which runs on MicroVAX II computers. TEXSYS will use a Symbolics 3650 computer for the expert system and a unix-based graphics workstation to provide a sophisticated and flexible human interface to the operator.

The Conventional Control System

FLEXCON is a schedule driven system, with a master queue of processes to be used to initiate sensor readings from the Analogic Remote Terminal Units, to convert the sensor readings into engineering units, to schedule data display updates and to start control processes. FLEXCON is capable of simple limit checking to generate alarms to the users and can run simple control loops (such as open loop and closed loop PID controllers). FLEXCON is currently used for factory automation and process control in the food industry and for monitoring and controlling a series of dams in the southern U.S..

The TTB Data Acquisition and Control System (DACS) is a hierarchical system built

using FLEXCON, with a master system controller and a controller for each of the thermal subsystems. In this project only the central thermal bus will be used, but the DACS software is capable of controlling and monitoring up to five connected bus and radiator systems in later stages of the TTB project.

TEXSYS is to use the DACS system to collect the sensor readings and dispatch control commands to the thermal bus. For example, it is necessary, when using FLEXCON alone, for the operator to control some of the solenoid values that gegulate the fluid and vapor flow in the bus. These will be under automatic control of TEXSYS. The types of data to be collected and their update rates are shown in figure 2.

Real Time Data Acquisition Software

In order to minimize the load on the symbolic processor containing the expert system, a software system has been designed that will provide TEXSYS with data inputs in a flexible and timely manner. This software, the TEXSYS Data Acquisition System (TDAS) will run on the MicroVAX which hosts the master DACS software and will be written in FORTRAN 77. FORTRAN is being used for this software for compatibility with the FLEXCON system.

The TDAS software will be written to make use of a feature of FLEXCON, called the user process facility. In this facility, a process or group of processes can be created by the user and will be scheduled by FLEXCON to run in cooperaton with the normal FLEXCON processes.

There are two goals for this software. The first is to reduce the amount of data sent from FLEXCON to TEXSYS by examining the sensor readings as they are rceived and only forwarding to TEXSYS those values that meet the current criteria of significance. The second goal is to perform as much of the numeric processing of data points in TDAS as possible, using a faster numeric processor (the MicroVAX). The DACS and TEXSYS are to be connected over a 10 mbps Ethernet connection using the DECNet protocols. (See figure 3.) This connection is to be established and managed by TDAS.

FLEXCON maintains an in-memory database of sensor locations, update rates, high and low limits, data validity limits, and current sensor readings. Also stored in this database are data items calculated from sensor inputs and the parameters associated with the control loops (setpoints, coefficients, etc.). After each set of updates is made by FLEXCON, the TDAS software will be signaled and it will scan the database looking for significant data. If nothing of significance is found, the expert system will be informed. This will also serve to verify the status of the Ethernet communications link.

The initial significance tests and calculations shall be fairly simple. As the input from some of the sensors is noisy, the data will first be low pass filtered to eliminate the high frequency noise. This process is one that the current operators perform mentally, as they watch the trend of the data points and the average value rather than the instantaneous value of a sensor reading. The data shall then be checked against previous values and the slope of the data sequence will be calculated. After these calculations are performed, the alarm limits in the data base will be compared to

the filtered readings. If the value is outside the normal range, it will be checked against the validity limits for that kind of sensor. If there is a limit violation, the reading, the slope and the type of limit violated will be posted to an agenda of items to be reported to TEXSYS. After all of the sensor readings that were updated are checked, the agenda will be examined and the information on it will be sent to TEXSYS. The data on the agenda can be prioritized so that the expert system can focus on those systems of highest importance. (See figure 4 for a diagram of this process.)

It shall also be possible for the expert system to ask for sensor values, both raw and filtered, to be reported at each update, regardless of the limit checks. This continuous reporting shall be selectable by sensor, by subsystem (e.g., cold plate evaporator #1, condenser #3, etc.) or by sensor type (all pressures in fluid/vapor loop A) allowing the expert system to automatically receive all information needed to support the reasoning of the current focus of attention of the expert system. For example, if an evaporator was reaching full heat absorption capacity, which could lead to dryout of the evaporator, the surface temperature will start to rise and the duty cycle of the inlet valves will approach 100%. The TDAS limit checks would notice the increasing temperature and the trend analysis would notice the increasing valve duty cycle and report these data to TEXSYS. TEXSYS would then request that all sensor readings associated with the evaporator be continuously reported to it until the problem was diagnosed, reconfiguration of the evaporator accomplished and the evaporator returned to normal.

TEXSYS shall also be able to modify the alarm limits used by TDAS. This is needed due to the operational characteristics of the thermal bus system during transition periods such as start-up, loop temperature setpoint change, and shutdown. During these operations, certain of the sensor readings will vary widely and may show readings that would be abnormal in the steady state. Alarm limits of the system therefore can be widened or changed in value without compromising safety, thus reducing the number of false alarms. This will allow improper performance to be spotted, reported and diagnosed with false alarms minimized.

The data reporting shall be performed synchronously, once each second (the fastest sensor update rate), and the amount of data that will be reported to the expert system each frame will vary. The largest size frames will occur as the temperature readings occur each 10 seconds. The data will be reformatted to allow rapid insertion into the knowledge base of the expert system. In addition, archival information will be stored and later retrieved by TDAS from a data base of sensor readings. This archival information will be analyzed, by TEXSYS at low priority, to spot long term trends.

The second form of interaction that TEXSYS will have with the DACS will be in TEXSYS' ability to control the thermal bus. TEXSYS shall have the capability for modification of the control loop constants used by the DACS. All of the control parameters (such as setpoints, gains, etc.) will be reported to TEXSYS at system startup. In addition, the direct control of crossover valves and other items not in control loops can be done by TEXSYS via TDAS.

Future Expansion Plans

These basic capabilities shall be implemented in the first version of the TEXSYS/DACS interface. Several additional techniques are being evaluated with domain experts for inclusion in later revisions of the TDAS software. First, the use of spectral analysis by Fourier transforms to look for cyclic information in the data is being studied. There are potentially damaging situations that can arise from undamped periodic pressure surges in the bus, and if the data rates are high enough to detect such surges this capability may be added. Data from past thermal bus tests are being assembled from data archives for this analysis. Secondly, noise analysis (also by spectral techniques) is being considered as a method for spotting incipient sensor system failure. Some sensor failures are preceded by increasing noise in the sensor readings. Again, the data from previous tests are being assembled to look for noise signatures characteristic of imminent failures.

Finally, the ability to build in contextual and situational sensitivity to allow TDAS to report additional relevant information immediately upon the detection of a significant sensor reading is under consideration. For example, if a pressure drop is reported in a pipe, the upstream and downstream pressures would be needed to determine if the drop is due to a leak or a faulty sensor. In addition, by knowing the current operational state of the bus (startup, shutdown, etc.) the data gathering capabilities can be further modified to allow the automatic capture of related sensor information when abnormal data is detected.

Conclusion

The use of incremental, layered interface software described here will allow us to decrease the risk in the program and to easily experiment with alternative approaches to individual problems. The flexibility in this approach will allow the system to be modified for subsequent demonstrations in the SADP program. These demonstrations will include integration of TEXSYS to cooperate with a power system expert system, a hierarchical expert system complex and finally a distributed expert system architecture. In addition, integration with other Space Station testbed hardware, such as the prototype Data Management System will be easier.

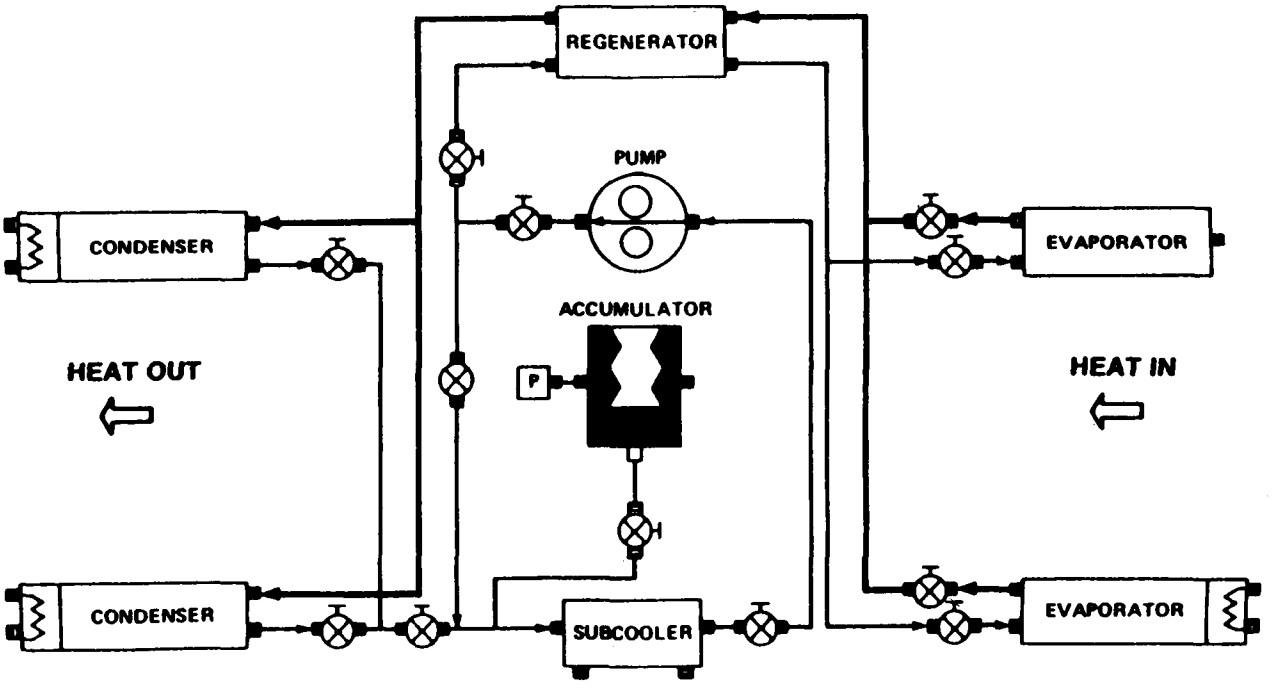
Acknowledgment

We would like to acknowledge the assistance of the following at JSC: Hal Hiers, Intelligent Systems Branch and Paul Marshall, our domain expert from Crew and Thermal Systems Division (EC). At the Ames Research Center we are indebted to Carla Wong, Chief, SADP Project Office; Mary Schwartz and Bill Erickson, SADP knowledge engineers. Finally, at LEMSCO, Robert Norsworthy, Hsi-Jen Chao and Robert Faltisco are working at turning these ideas into reality.

The LEMSCO work described in this paper was performed under NASA Contract NAS 9-15800 and NAS 9-17900.

References

1. National Commission on Space, *Pioneering, the Space Frontier*, Bantan Books, New York, New York, 1986.
2. Bull, J. S. , Brown, R., Friedland, P. , Wong, C. M. , Bates, W., Healey, K. J., Marshall, P. , "NASA Systems Autonomy Demonstration Project: Development of Space Station Automation Technology", AIAA-87-1676, Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program, Arlington, Virginia, March 9, 1987.
3. Hayes-Roth, Frederick; Waterman, Donald; Lenat, Douglas, *Building Expert Systems*, Addison-Wesley, Reading, Ma., 1983.



THERMAL CONTROL SYSTEM SCHEMATIC
TWO-PHASE AMMONIA SYSTEM

Figure 1

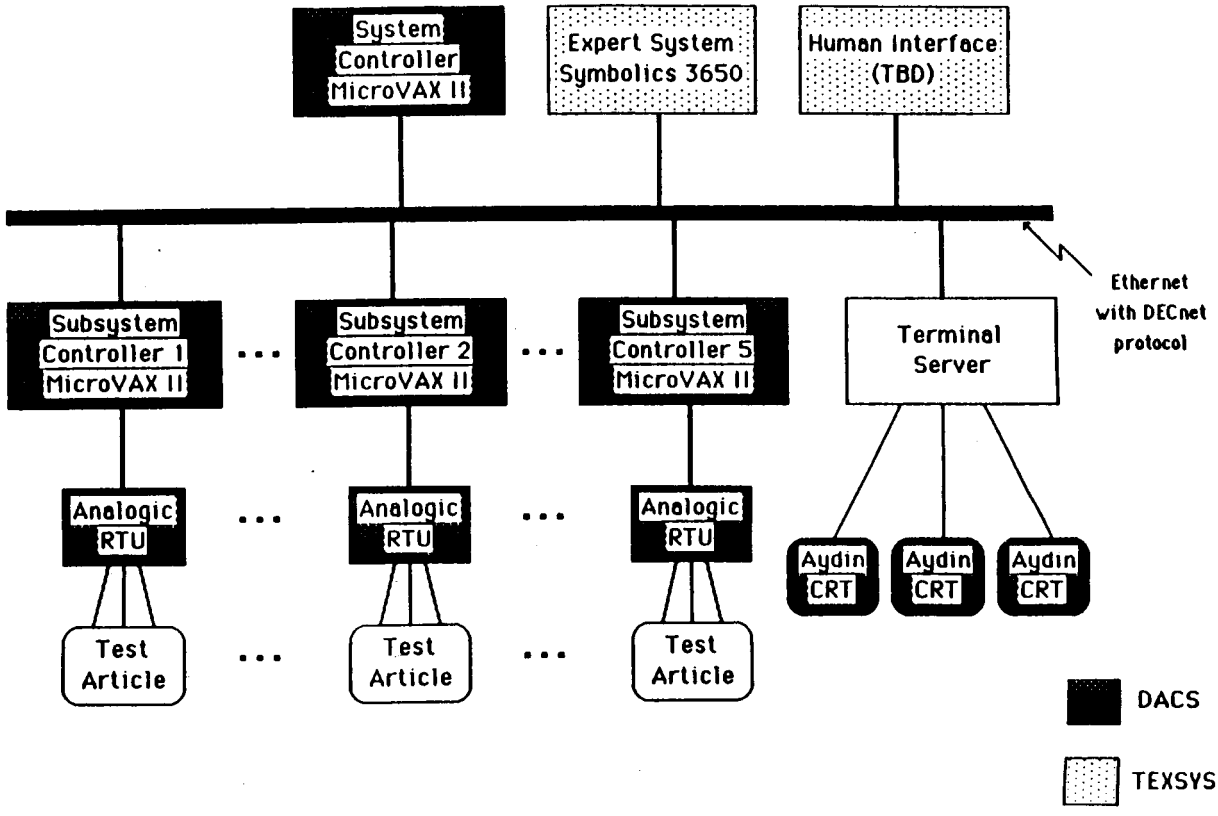
ANALOG SENSORS

	Update Rate (sec/sample)	Number	Length (bits)
Temperature	10	72	32
Flowmeters	1	8	32
Pressure	5	20	32
Liquid level	10	2	32
Pump RPM	1	1	32

DIGITAL INPUTS

Pump RPM	1	1	32

FIGURE 2.- REAL-TIME CONTROL INPUTS



Hardware Configuration

Figure 3

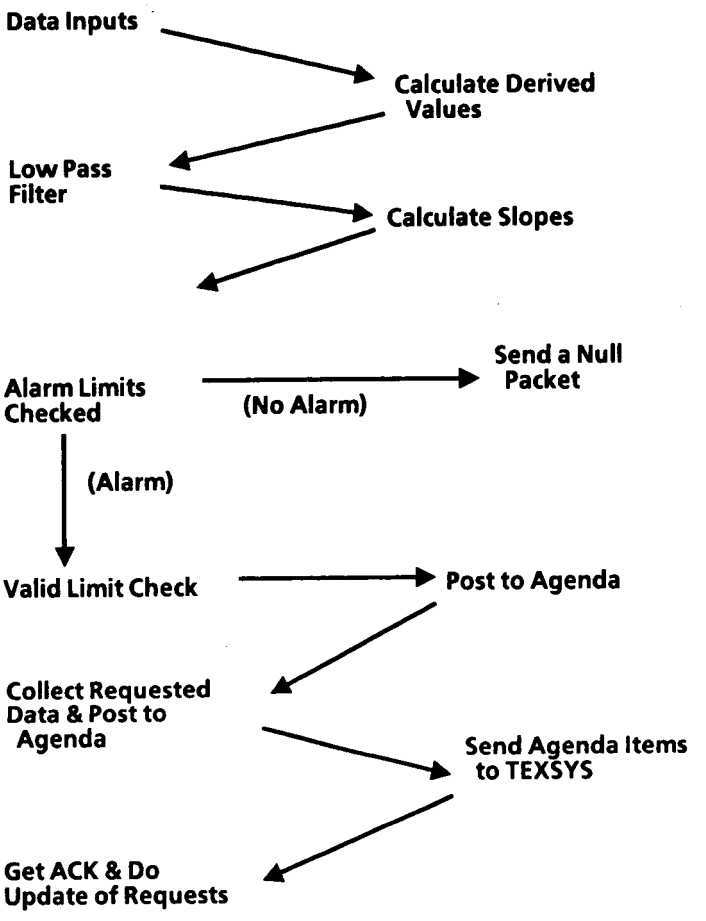

Data Inputs
Calculate Derived Values
Low Pass Filter
Calculate Slopes
Alarm Limits Checked
(No Alarm)
Send a Null Packet
(Alarm)
Valid Limit Check
Post to Agenda
Collect Requested Data & Post to Agenda
Send Agenda Items to TEXSYS
Get ACK & Do Update of Requests

FIGURE 4.

IMPLEMENTING CLIPS ON A PARALLEL COMPUTER

Gary Riley
NASA/Johnson Space Center
Artificial Intelligence Section
Mail Code FM7
Houston, Texas 77058

ABSTRACT

The 'C' language integrated production system (CLIPS) is a forward chaining rule-based language developed by the Artificial Intelligence Section (AIS) of the Mission Planning and Analysis Division (MPAD) at the Johnson Space Center (JSC) to provide training and delivery for expert systems. Conceptually, rule-based languages have great potential for benefiting from the inherent parallelism of the algorithms that they employ. During each cycle of execution, a knowledge base of information is compared against a set of rules to determine if any rules are applicable. Parallelism can be employed to speed up this comparison during each cycle of execution. Parallelism also can be employed for use with multiple cooperating expert systems. To investigate the potential benefits of using a parallel computer to speed up the comparison of facts to rules in expert systems, a parallel version of CLIPS was developed for the FLEX/32, a large-grain parallel computer. The FLEX implementation takes a macroscopic (or high-level) approach in achieving parallelism by splitting whole sets of rules among several processors rather than by splitting the components of an individual rule among processors. The parallel CLIPS prototype demonstrates the potential advantages of integrating expert system tools with parallel computers.

INTRODUCTION

Expert system building tools have shown a great deal of utility in solving knowledge intensive tasks that would often daunt conventional approaches using procedural languages. These tools provide languages that allow solutions to be expressed in paradigms that "closely" resemble the human solution process. Knowledge engineers can express heuristics using rule paradigms as opposed to coding nested if/then statements in a procedural language. The inference engine of the expert system is used to determinine which information has satisfied the conditions of the appropriate rules. The control routines for matching information (facts) against rules are provided by the tool, not the programmer. In addition, many expert system building tools are provided on computers hosting extremely powerful development environments that promote the interactive and incremental development of programs.

The use of high-order languages, however, does not come without cost. Typically, expert systems written in high-order languages run one to two orders of magnitude slower than expert systems directly coded in procedural languages. Speed is very often traded for increased productivity during development and easier maintenance. Sometimes, this tradeoff is acceptable, but many applications requiring real-time speed that could benefit from expert system technologies might not be able to accept this tradeoff.

The AIS of JSC's MPAD has been active in both the design of expert system building tools and the use of parallel computers. Several expert systems have been developed which require real-time or near real-time speed, including NAVEX[1] and MCCSSES[2]. Parallel processing is one of the ways in which expert system speed performance can be increased[3]. This background presented the opportunity and motivation to investigate the use of parallel processing in expert system building tools.

CLIPS

CLIPS is a forward chaining, rule-based language developed by the AIS at JSC to solve both training and delivery problems not fully addressed by most commercially available expert system shells[4]. A forward chaining, rule-based language such as CLIPS has three primary components: a set of rules, a knowledge base consisting of facts, and an inference engine. Facts represent chunks of information such as the altitude of the Space Shuttle or the temperature reading of a particular sensor. Rules basically are if/then statements of heuristic knowledge. The if

portion of a rule is a series of patterns which must have appropriate matches with facts in the knowledge base for the rule to be activated. The then portion of a rule is a series of actions to be taken when the rule is executed. Two possible actions (among many) could be to add new facts to the knowledge base or to remove existing facts from the knowledge base. The inference engine is the mechanism that determines which rules apply and also compares the facts in the knowledge base to the rules and determines which rules are applicable given the current state of the knowledge base. It then selects one of the applicable rules and applies the actions found in the then portion of the rule. For a more complete description of CLIPS, see references [5] and [6].

FLEX/32 PARALLEL COMPUTER

The FLEX/32[7] is a large-grain parallel computer capable of housing up to 20 computer modules and 10 shared memory modules in 1 cabinet. Cabinets also can be connected together. Computer modules available are based on the Motorola 68020 and the National 32032 and may be used in any combination. The FLEX/32 is a multiple instruction stream/multiple data stream (MIMD) computer. Each processor can run independent of the others and can access either shared or local memory. The FLEX/32 (used by the AIS at JSC) has six National 32032 processor modules and two shared memory modules. The processor supporting UNIX has 4 megabytes of local memory, while the other five processors have 1 megabyte of local memory. Each common memory module has 256 kilobytes of memory.

The operating system used on the FLEX/32 is the UNIX System V Operating System. This provides all of the language support normally associated with this operating system. In addition, Flexible Computer offers two languages for parallel programming: Concurrent C [8] and Concurrent FORTRAN. These two languages have been extended to allow parallel processing constructs.

APPROACH TO PARALLELISM

Two levels of incorporating parallelism into CLIPS were considered: macroscopic and microscopic parallelism. A macroscopic approach would attempt to preserve the low-level implementation of the CLIPS inference engine and to incorporate parallelism on a "high" conceptual level. A microscopic approach, by contrast, would attempt to incorporate parallelism in the low-level implementation of the CLIPS inference engine.

A macroscopic approach would provide the quickest means of incorporating parallelism into CLIPS. Source code changes using this approach could be kept to a minimum by utilizing most of the code used for the sequential version of CLIPS. This was desirable because a sequential version of CLIPS was being maintained on a VAX 11/780 for use on that and other sequential computers. This sequential version experiences frequent change for both maintenance and improvement. A macroscopic approach would allow easier integration of changes made in the sequential version with the parallel version. Use of a macroscopic approach also would allow the final product to be a fully developed expert system tool and not a research prototype. A "start from scratch" approach inevitably would not contain of all of the features the sequential version of CLIPS provides. Finally, the source code for CLIPS already was well understood and available.

A microscopic approach would allow the investigation of the best possible techniques for incorporating parallelism. Converting a program developed on a sequential computer to take advantage of the architecture of a parallel computer would not be able to take advantage of other algorithms that may better exploit the architecture of the parallel computer. Recoding the inference engine to take advantage of parallelism at a very low level would allow the very best techniques to be applied.

PARALLEL CLIPS OVERVIEW

A macroscopic approach incorporated parallelism into CLIPS. This approach limited the number of changes to the CLIPS source code and allowed the ongoing changes to the sequential version of CLIPS to be integrated more easily with the parallel version of CLIPS.

The steps taken for the assertion of a new fact are shown for the sequential version of CLIPS in figure 1. First, the fact is filtered through the pattern matcher. The pattern matcher determines which patterns in the if portion of the rules have been matched by the fact that is being asserted. Rules with matched patterns then are given the information that a fact has matched one of their patterns. If this additional fact causes all of the conditions of the rule to be satisfied, the rule is placed on the agenda (in this case, local to a single processor). After all new assertions have taken place, a rule will be selected from the local agenda and its actions will be applied.

An assertion in the parallel version adds an additional level above local fact assertions. The FLEX/32 implementation of CLIPS splits the set of rules among several processors to achieve parallelism. Figure 2 shows the steps taken to assert a fact in the parallel version. A master processor provides the user interface capabilities and acts as a driver for the other processors. The master processor picks a single rule to be applied from the global agenda (the set of all applicable rules). The master processor then informs

the processors containing groups of rules of the actions of the rule that are to be applied. A given action of the rule then is performed by all of the rule group processors in parallel before the next action of a rule is undertaken.

In the case of a fact assertion, the processor containing the rule is informed that it may begin executing its actions. The fact to be asserted is posted to global memory, and the other processors are notified that a fact has been posted to global memory. Each individual processor then asserts the fact exactly as if it were running on a sequential computer using the steps shown in Figure 1. After the assertion takes place on the processor, the local agenda selects one applicable rule (if it has any available) to be posted to the global agenda.

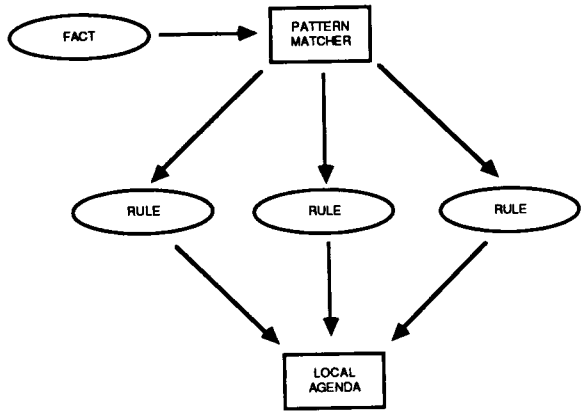


Figure 1: Local Fact Assertion

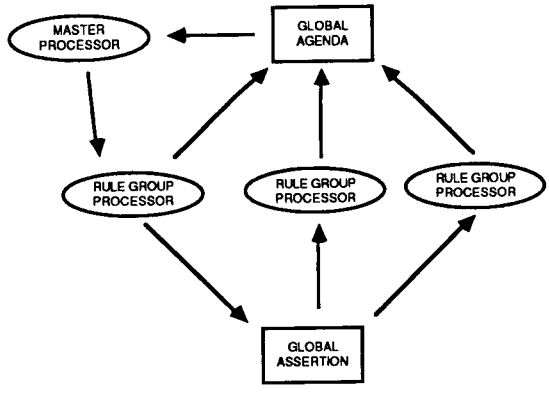


Figure 2: Global Fact Assertion

Retractions are handled similarly to assertions, with the fact to be retracted being posted to global memory and other processors being informed of the task by the rule group processor that contains the rules and is executing the actions. The rule group processor containing the rule waits for all of the other processors to finish before beginning the next action. Other actions that take place in the then portion of a rule (variable bindings, function calls, etc.) are handled only by the processor with the executing rule. Once the rule has finished executing, control is returned to the master processor where another rule is selected from the global agenda to be executed, repeating the basic cycle until no rules remain on the global agenda.

IMPLEMENTATION

The main problem in the implementation of parallel CLIPS was the communication between the master and slave processors. Initial attempts used processes to create and control the slave processors and their tasks. For example, if the main processor wanted the slave processors to assert a fact, it would create a process running on each of the slave processors to handle this task. The main processor then would wait for slave processors to finish. This method turned out to be relatively easy to code using the high-level, parallel constructs of Concurrent C; however, it also was quite inefficient. Sample problems actually ran slower as the number of additional processors was increased. Process creation is expensive, especially when the task to be performed is of a small time duration. Further, multitasking on a single processor also does not seem to work as well as one might expect. Running a slave process on the same processor as the master process caused inefficiency in multitasking. The FLEX/32 arbitration for multitasking does not appear to be very efficient. This conclusion was bolstered further by the results of other parallel programs.

The second implementation attempt corrected two of the errors experienced in the first attempt. Process creation and multitasking were avoided during run time of the expert system. The processor with the master process was not given a slave process. All other processors had a slave process. This slave process ran constantly, waiting for a "message" which informed it that it had a task to perform. When it received the "message" and processed it, the slave process then would send a "message" back to the master process, informing it that the task had been completed.

An attempt was made to use the message passing facility of exceptions provided by Concurrent C; however, it proved too difficult to configure the channels in the appropriate manner for message passing. The final implementation used a set of flags in shared memory. Each processor had an active flag and, in addition, all processors shared a task flag. A processor requiring other processors to perform an action would set the task flag to the appropriate task to be performed. It then would set the active flag of the other processors to active to signal them to begin

execution of the task. The controlling processor then would monitor the active flag. When the active flag was set to inactive, the controlling processor knew that the subordinate processor has completed its task. Information passing was controlled by copying information to global memory and by having each processor copy the information down to its local memory.

The problems encountered during implementation showed that many ways exist to implement a problem given a concurrent language on a parallel computer. Unfortunately, the best way to implement a problem often has to be determined empirically.

RESULTS

Two problems were used to demonstrate potential speed benefits of the parallel algorithm used in parallel CLIPS. The first of these problems was a "goal" problem. This problem was a 30-rule version of the monkeys and bananas problem described in *reference [9] modified to handle more goals and* situations. Eighty-six rules fire to solve the problem for the initial conditions used. The other problem used was a "data" problem. This problem has 13 rules: 1 startup up rule and 12 data-intensive rules. The data-intensive rules were combinatorial in nature in that each rule potentially could add tens to thousands of rule activations to the agenda with the addition of each new fact (depending upon the number of facts already in the knowledge base). To prevent all of these rule activations from actually occurring, a pattern was added to the end of each of the if portions of the rules which had no corresponding matches among facts in the knowledge base. Although this pattern prevented the rules from being activated, it still allowed the computational work in computing the partial matches to be finished. The startup rule asserted nine facts and was the only rule that fired.

The problems were run on CLIPS V3.11 on a VAX 11/780 using VMS, CLIPS V3.11 on the FLEX using UNIX, and parallel CLIPS (based on V3.11) using one to four processors under the multitasking multiprocessing operating system (MMOS). The results are shown in table I.

Table I: Timing Test Results

Version	Data Problem	Goal Problem
CLIPS VAX	15.2	3.3
CLIPS FLEX	21.9	6.4
Parallel CLIPS (1P)	18.1	5.7
Parallel CLIPS (2P)	9.3	5.3
Parallel CLIPS (3P)	7.3	4.4
Parallel CLIPS (4P)	5.5	4.2

The "goal" problem demonstrated only modest speedup as more processors were added. This demonstrates that speedup will occur only for problems in which the problem is divided evenly among the processors. That is, for each fact assertion and retraction, each set of rules on a processor has approximately the same amount of work to perform. This could best be achieved with a set of rules that numbers in the hundreds rather than in the tens.

The "data" problem specifically was tailored to demonstrate a "best case" situation for parallel CLIPS. Only one rule is fired and this rule asserts several facts. For each of the facts asserted, a great deal of work has to be done and this work is very evenly divided among the processors. Two processors ran the problem 1.9 times faster, and four processors ran the problem 3.3 times faster than a single processor. These numbers represent 95 percent and 80 percent, respectively, of maximum possible speedup.

Although rule sets run slightly faster for most examples and much faster for some examples, it is important to remember that the inference engine is not actually working faster. Parallel CLIPS speeds up the system by making the set of rules appear smaller by distributing them among several processors.

AREAS FOR IMPROVEMENT

CLIPS uses the Rete pattern matching algorithm which provides an efficient method for finding all of the facts that match the patterns in the if portions of the rules[10],[11]. It is important to remember that optimizations used in the Rete algorithm may be affected by splitting up rules among processors. Common elements of both patterns and rules can be shared, making the system more efficient. To split the rules among several processors will remove some of the efficiency that is gained by sharing. The version of CLIPS used for parallel CLIPS (version 3.11) uses the Rete algorithm. However, it does not take advantage of common sets of patterns shared between rules (join sharing). Starting with version 4.0, versions of CLIPS incorporate this optimization. The "data" problem used cannot take advantage of join sharing; however, most problems can take advantage of join sharing to a greater or lesser extent. For example, the "goal" problem has 7 of its 30 rules which can benefit from join sharing. Join sharing especially benefits large expert systems with many sets of similar rules. A version of parallel CLIPS based on version 4.0 would allow investigation of the tradeoffs encountered between sharing commonality among rules on a single processor and splitting rules among several processors.

The next logical step in testing the benefits of parallel CLIPS is to develop a suitable problem for testing large expert systems. This problem should consist of

at least 100 rules and should not be dependent on extensive input/output (I/O) or external functions. The initial state or condition should be hardwired so the problem can just execute without human intervention.

The parallel implementation could make use of an action queue to store a list of assertions and retractions to be performed by the rule groups. Each processor could retrieve the next action to be performed from this queue when it has completed its current action. This would ease some of the strict synchronization of rule execution and also would allow processors to proceed at their own pace rather than at the pace of the slowest processor out of all the groups.

Programming constructs should be provided which allow rules to be specifically assigned to certain processors by the programmer. In the current implementation, CLIPS distributes rules among processors with a round robin distribution scheme. The ability to assign rules specifically to processors would be useful when attempting to fine tune a parallel expert system for speed.

CONCLUSIONS

The early results from parallel CLIPS are very encouraging. Parallel CLIPS could be used not only as a program for investigating parallel inference engines, but as a program for the actual delivery of an expert system. Parallel CLIPS is still a prototype, and more development work is required to remove the remaining rough edges. In addition, more suitable problems need to be found to investigate the speed improvements possible with parallel CLIPS.

ACKNOWLEDGEMENTS

The author would like to thank Joe Giarratano and Chris Culbert for their comments and suggestions on this paper.

ACRONYMS

AIS	Artificial Intelligence Section
CLIPS	C language integrated production system
I/O	input/output
JSC	Johnson Space Center
MIMD	multiple instruction stream/multiple data stream
MMOS	multitasking multiprocessing operating system
MPAD	Mission Planning and Analysis Division

REFERENCES

[1] Maletz, M. and C. Culbert, "Monitoring Real-Time Navigation Processes Using the Automated Reasoning Tool (ART)." In Proceedings of the First Annual Aerospace Applications of Artificial Intelligence Conference, AAAIC-85, Dayton, OH, September 1985.

[2] Clemons, P., C. Culbert, and L. Wang, "Development of an Expert System to Assist Monitoring Mission Control Center Software Status." In Proceedings of the First Annual Conference on Robotics and Expert Systems, ROBEXS'85, Houston, TX, June 1985.

[3] Boarnet, M., "Requirements for the Next Generation of Expert System Builders." NASA Technical Memo FM7(86-27), NASA/Johnson Space Center, Houston, TX, February 1986.

[4] Giarratano, J., C. Culbert, G. Riley, and R. Savely, "A Solution to the Expert System Delivery Problem." Submitted for publication.

[5] Culbert, C., "CLIPS Reference Manual." NASA Technical Memo FM7(87-131), NASA/Johnson Space Center, Houston, Texas, July 1986.

[6] Giarratano, J., "CLIPS User's Guide." NASA Internal Note 86-FM-25 (JSC-22308), Mission Planning and Analysis Division, NASA/Johnson Space Center, Houston, TX, October 1986.

[7] Flexible Computer Corporation, "FLEX/32 MultiComputer System Overview." Flexible Computer Corporation, Dallas, TX, June 1985.

[8] Flexible Computer Corporation, "ConCurrent C Reference Manual." Flexible Computer Corporation, Dallas, TX, March 1986.

[9] Brownston, L., R. Farrell, E. Kant, and N. Martin, Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming. Addison-Wesley Publishing Company, Inc., Reading MA, 1985.

[10] Forgy, C. L., "On the Efficient Implementation of Production Systems." Ph.D. Dissertation, Carnegie-Mellon University, Pittsburgh, PA, 1979. (Available from University Microfilms International, Ann Arbor, MI)

[11] Forgy, C. L., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem." Artificial Intelligence 19, 1982, pp. 17-37.

N88-17260

REAL-TIME ARTIFICIAL INTELLIGENCE ISSUES IN THE DEVELOPMENT OF THE ADAPTIVE TACTICAL NAVIGATOR

Peter E. Green*

Intelligent Machines Group
Worcester Polytechnic Institute
Worcester, Massachusetts 01609

Douglas P. Glasson†
Jean-Michel L. Pomarede‡
Narayan A. Acharya†

The Analytic Sciences Corporation
55 Walkers Brook Drive
Reading, Massachusetts 01867

ABSTRACT

The Air Force Avionics Laboratory is sponsoring the development of the Adaptive Tactical Navigation (ATN) system. ATN is a laboratory prototype of a knowledge-based system to provide navigation system management and decision-aiding in the next generation of tactical aircraft. ATN's purpose is to manage a set of multimode navigation equipment, dynamically selecting the best equipment to use in accordance with mission goals and phase, threat environment, equipment malfunction status, and battle damage. ATN encompasses functions as diverse as sensor data interpretation, diagnosis, and planning.

Real-time issues that have been identified in ATN and the approaches used to address them are addressed in this paper. Functional requirements and a global architecture for the ATN system are described. Decision-making within time constraints is discussed. Two subproblems are identified; making decisions with incomplete information and with limited resources. Approaches used in ATN to address real-time performance are described and simulation results are discussed. A communicating expert objects paradigm for the global architecture, an evidence scheduled blackboard for low level diagnostic procedures, and rules for scheduling the data acquisition for a causal network that performs high-level reasoning are presented.

1. INTRODUCTION

Tactical aircraft of the 1990's will have a wide variety of advanced avionics subsystems which support equipment status assessment, onboard resource management and pilot decision aiding. These systems represent the next generation of onboard systems technology. Many of them will utilize knowledge-based systems that augment or provide a supervisory function over (already-complex) current generation navigation, guidance, control, sensing and threat-warning systems.

The Adaptive Tactical Navigator (Ref. 1) is an onboard intelligent system that provides equipment management and pilot decision aiding for an advanced multisensor (i.e., radio-, communication- and sensor-aided) aircraft navigation suite. Figure 1 depicts the functional organization of the ATN system which forms a four-level hierarchy. Three expert specializations comprise the Equipment Management function forming the lower two levels of the hierarchy:

- Navigation Source Manager: These are shown in individual replications for GPS/INS, SITAN and Inertial configurations in Fig. 1. These experts use design engineering models to monitor equipment performance and to detect and isolate failures or degradations.
- System Status: This expert diagnoses system health based on reliability data, mission environment and lower-level diagnoses.
- Moding: This expert configures viable component combinations (including non-standard "jury-rigs") based on current equipment status and determines appropriate handoff strategies for mode changes.

The Decision-Aiding functions are accomplished by the top-level experts:

- Event Diagnosis: This expert evaluates planned navigation events (e.g., a waypoint encounter and designation) and diagnoses anomalous or out-of-spec events to support pilot moding decisions.
- Mission Manager: This expert stores mission plan and environment (threats, ECM) data and determines which available equipment configurations are appropriate to the current and forecasted mission situation.
- Pilot-Vehicle Interface (PVI) Management: This expert manages communication between the ATN and the pilot.

Also indicated in Fig. 1 are the main communication paths among the functions and the processing characteristics of each level. As indicated in the figure, there is a broad mix of deterministic and stochastic processing load and message generation among the

*Professor of Electrical Engineering

†Department Staff Analyst, ATN Program Manager

‡Member Technical Staff

†Member Technical Staff

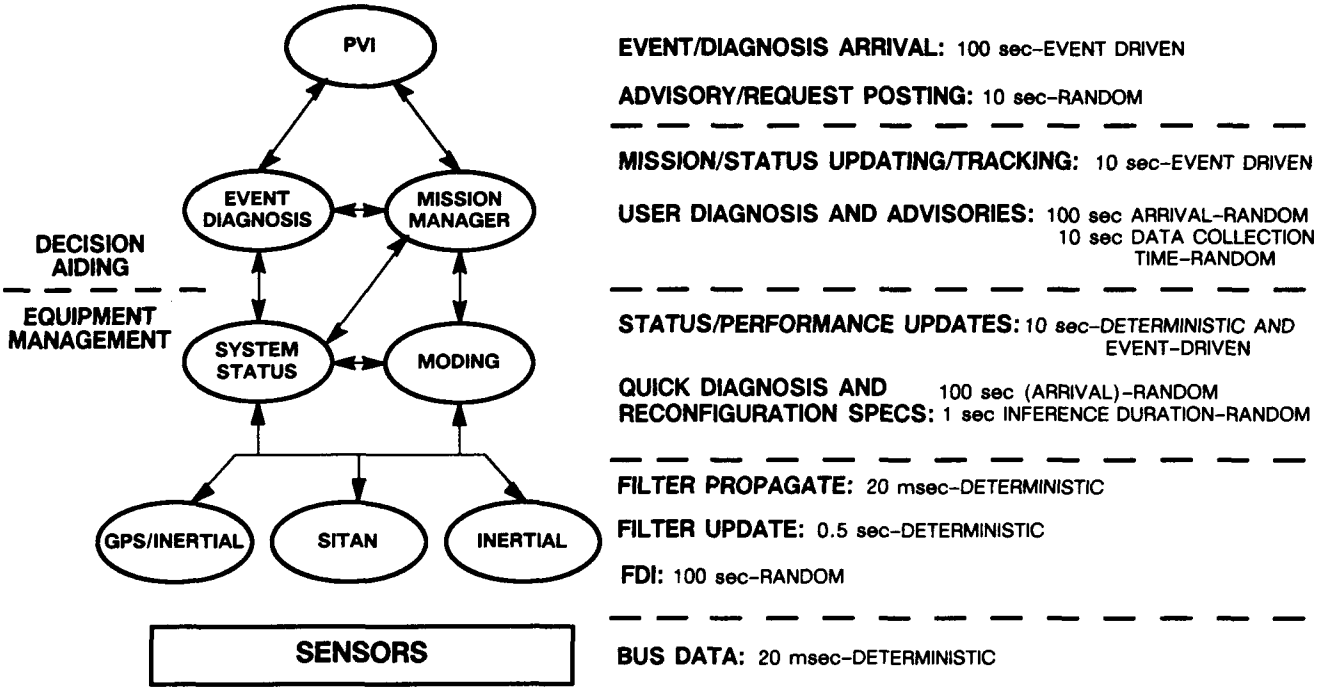


Figure 1. ATN Functional Organization

functions. Computations and communications at the bottom of the hierarchy are clock-driven at high data rates. At higher levels, the processing transitions to lower frequency, random event-triggered computations and communication.

Real-time issues encountered in the design of ATN are discussed in Section 2. In Section 3 specific techniques used in the ATN Global and module-level designs are described. These techniques provide efficient module scheduling and focus of control, effective management of limited computational resources and methods for prioritizing communication and interaction with the crew.

2. REAL-TIME ISSUES

In the design of real-time, onboard system such as ATN it is not possible to provide adequate resources for all the possible actions that may be desirable at any given time. Mechanisms must be devised to ensure that resources are allocated to the tasks that are the most important at a given time. Also, the urgency of the situation may warrant a decision based on incomplete information.

In ATN (and other onboard systems) the real-time limitations include:

- Computer Resources: ATN has a number of loosely-coupled experts which must compete for execution on various processing elements. For example, in identifying multiple sensor failures, the Navigation Source Manager must run monitoring procedures in numbers that far exceed available processing capacity. It is also necessary to ensure that low priority tasks (such as event logging) do not delay high priority tasks (such as moding recommendations).

- Information Availability: Desired information may not be available until certain times in the mission; even then, the information may not be available. Aimpoints used for navigation updates or diagnosis may be obscured. Critical information from other aircraft may be denied due to jammed communication links. GPS satellite links may be obscured by terrain or jammed. Updates that create emissions (e.g., radar ground map) may entail unacceptable risk of detection or homing missile attack.

- Pilot Attention: In a single-seat attack aircraft the crew can spend little time on navigation functions -- especially in hostile situations. During low-level ingress and attack phases of the mission crew attention is directed out-of-cockpit for situation assessment and target location. Pilot attention allocation to navigation ranges from moderate-to-low during ingress to totally unavailable during the final attack phase.

These real-time performance requirements and constraints are not addressed by traditional AI paradigms. Simple rule-based approaches are inadequate since all required elements of a rule's antecedent (complete information) are required for the rule

to fire. In some cases, hypothesis spaces can be represented as a tree; decision-making reduces to efficient search. Unfortunately, in a real-time situation the tree of hypotheses can grow exponentially due to the evolution with time of the world model. Finally, unlimited processing alone cannot ensure effective interaction with the crew. These interactions must be managed in a manner appropriate to the mission phase, current situation and state of evidence.

These important real-time issues were recognized at an early phase of the ATN System development (Ref. 2). A system design philosophy was adopted for subsequent phases of ATN to isolate specific real-time operation issues and to identify or develop design approaches to address them (Ref. 3 and Refs. 4, 5, and 11).

As the design of the current ATN system was formulated, a two-level, real-time design strategy was structured. This approach delineated global architecture and module-level design. At the global level, efficient methods were sought for prioritizing, scheduling and managing communication of a community of specialized experts. At the module level, paradigm-specific approaches for efficient processing, hypothesis generation/management and information prioritization were developed. Examples of global and module-level designs reflecting the ATN design philosophy and the current state of the ATN design are discussed in the following section.

3. ATN DESIGN APPROACHES

Selected elements of the current ATN design are highlighted in this section. Again, real-time performance is addressed at global and module levels. The global architecture is described in the first subsection. Module designs for the Navigation Source Manager, Event Diagnosis and Pilot Vehicle Interface (PVI) Manager are then outlined. These three designs are representative of the module-level real-time issues and design approaches in ATN.

Coordinating System Behavior: Global Architecture

The Global architecture of the ATN system is the Communicating Expert Objects (CEO) architecture (Ref. 8). The CEO architecture is an outgrowth of the HEARSAY paradigm (Refs. 10 and 11). In HEARSAY, hypotheses are posted on a global blackboard; a global scheduling procedure reviews the state of the blackboard and decides which knowledge source to execute next. Typically, the scheduling algorithm is complex and the blackboard review can be time-consuming. In the CEO approach, hypotheses are distributed among Expert Objects that communicate with each other by exchanging messages as shown in Fig. 2. Each message generated by a CEO includes a priority level based on the importance of the message and the "rank" of the sender. As will be discussed subsequently, scheduling is accomplished by a relatively simple process of determining which CEO has accumulated the greatest amount of high priority requests (with CEO rank and various safeguards as additional scheduling factors).

In the ATN application, the CEO approach offers significant efficiency advantages. Messages and message priorities can be designed a priori to

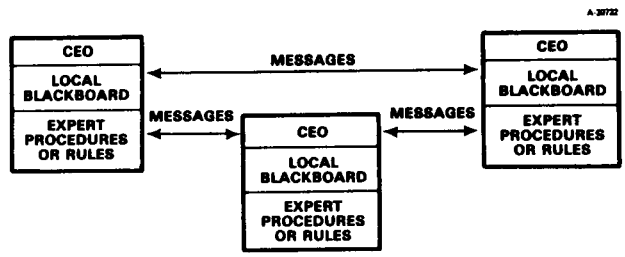


Figure 2. Communicating Expert Objects Architecture

achieve desired system behavior; i.e., to generate appropriate exchanges of messages to resolve stereotypical situations. As a result, the runtime scheduler processing is relatively simple and efficient (i.e., prioritize according to accumulated requests within each CEO). Simulation results (Ref. 12) have demonstrated the efficiency of this paradigm. Salient features of the CEO approach taken in ATN are outlined in the following paragraphs.

The Adaptive Tactical Navigator uses an implementation of a particular CEO methodology known as Activation Frames (Refs. 8 and 9). An AF (Activation Frame) forms a community of AFOs (Activation Frame Objects) as shown in Fig. 3. Each AFO is an expert in a limited problem domain and is the guardian of a set of private hypotheses. Each AF is a process which creates the environment in which all of its AFOs execute. Multiple AFs might coexist on the same processor or on multiple processors connected by a network.

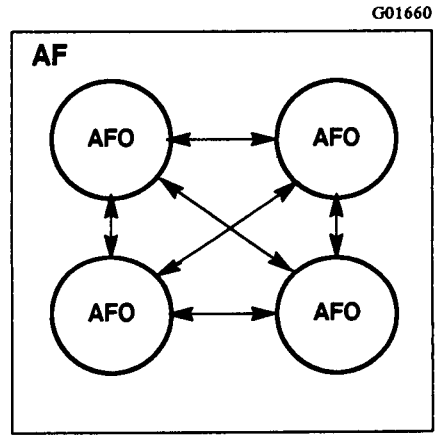


Figure 3. An AF as a Community of AFOs

In implementation, an AF is a process executing within an operating system. Communication between AFOs is provided by AF services using a message passing mechanism. Message passing among AFs is implemented by operating system level message passing mechanisms including, in the case of multiple processors, network protocol processing.

The flow of control within an AF is shown in Fig. 4. The scheduler selects the next AFO to be activated; the procedural code of the activated AFO is then executed. The AFO can then use different AF services (typically message sending and message receiving) during the execution of the procedural code. When the AFO returns control, the messages sent during its execution are actually delivered to the receiving AFOs.

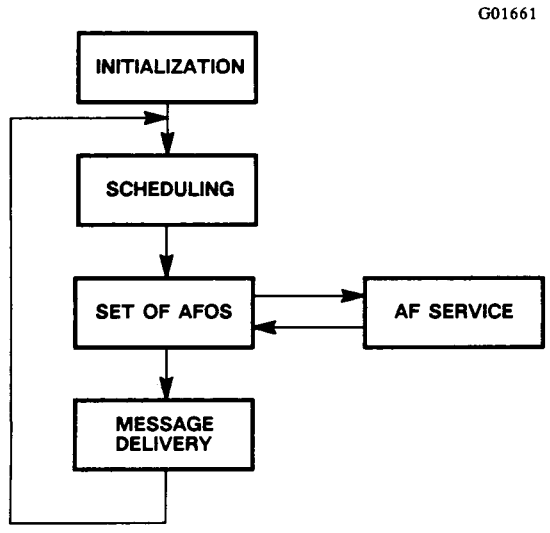


Figure 4. Flow of Control Within an AF

Each AFO has an input message queue and an output message queue. Message sending and receiving is depicted in Fig. 5. When an AFO wants to send a message, the message is put on its output message queue by an AF service. When an AFO wants to receive a message, the message is taken off the AFO's input message queue and made available by an AF service. The actual passing of the message from originating AFO to destination AFO (either within or outside the AF) is done by a delivery procedure after the AFO returns control to its governing AF.

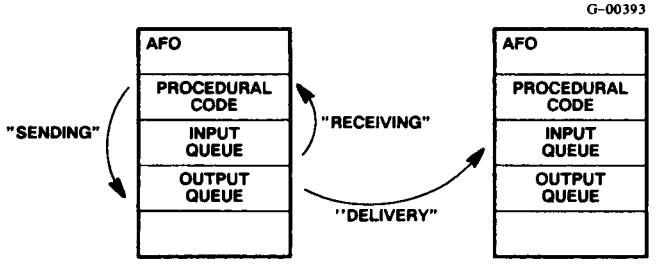


Figure 5. Message Passing by the AF

In the current scheduling scheme, each message is provided with a measure of its importance, the message activation level. Each AFO has an AFO activation level and an AFO activation threshold that are used by the scheduler to determine which AFO is next to execute. The AFO's activation level is the sum of all the message activation levels of messages on its input message queue. An AF schedules its AFOs for execution based on the difference between their activation levels and activation thresholds. The AFO whose activation level exceeds its threshold by the greatest amount is the next to execute.

Managing Compute-Intensive Processing - Navigation Source Manager

The principal goal of the Navigation Source Manager (Fig. 6) is to detect and identify sensor failures soon after they occur. Detection of failures is accomplished through analytic redundancy methods adapted to the identification of single sensor failures (SSFs). Identification of multiple sensor failures (MSFs) is based on detection of SSFs in combination. The task of SSF detection is delegated to a Scheduler which controls the execution of the Failure Estimation software. The Scheduler acts on requests from a Resource Allocator whose function it is to conduct a judicious search through the tree of MSFs. The left half of Fig. 6 shows the three parts of the Failure Detection and Identification (FDI) software of the Navigation Source Manager.

The methods of analytic redundancy (Refs. 13 and 14) provide tools for the comparison of outputs of dissimilar sensors. Time windows of sensor outputs, augmented with navigation system data, are combined in parity functions designed for specific sensor combinations. Derived from a knowledge of dynamics and measurement models, a parity function has the property that its value ideally remains zero only if no failures have occurred. Starting with linearized models, parity functions can be derived as linear combinations of sensor and control data; the coefficients of these are computed off-line and stored as part of the property list of the Scheduler.

The Resource Allocator routinely requests the Scheduler to conduct SSF tests for sensors in current use. If the outcome of these tests is inconclusive, the Resource Allocator must determine a stratagem for testing combinations of two SSFs, then three SSFs, and so on. Since FDI methods are compute intensive, an exhaustive search through the potentially large tree of MSFs is impractical. With six measurement channels in use, for example, exhaustive search for three failures may require analysis for 41 parities. The Resource Allocator must therefore, use external evidence of probable failures to condition its search toward a quick resolution. Information used to this end includes a priori sensor health estimates, damage and malfunction advisories, descriptors of ECM, weather and terrain environments, and pilot observations. An example of a constrained search is shown in Fig. 7. At the two failure search level, a maximum of 10 parities are run versus 45 for an exhaustive search. Upon receiving requests for specific tests, the Scheduler consults its property list to select appropriate parity functions and related data. These data include parity computation times that are used to determine an efficient and manageable schedule of computations.

The Failure Estimation software functions in three steps. First, parity values are computed from raw sensor outputs and system data. Parity values are then smoothed to estimate signal and noise levels for each function. Finally, differences between

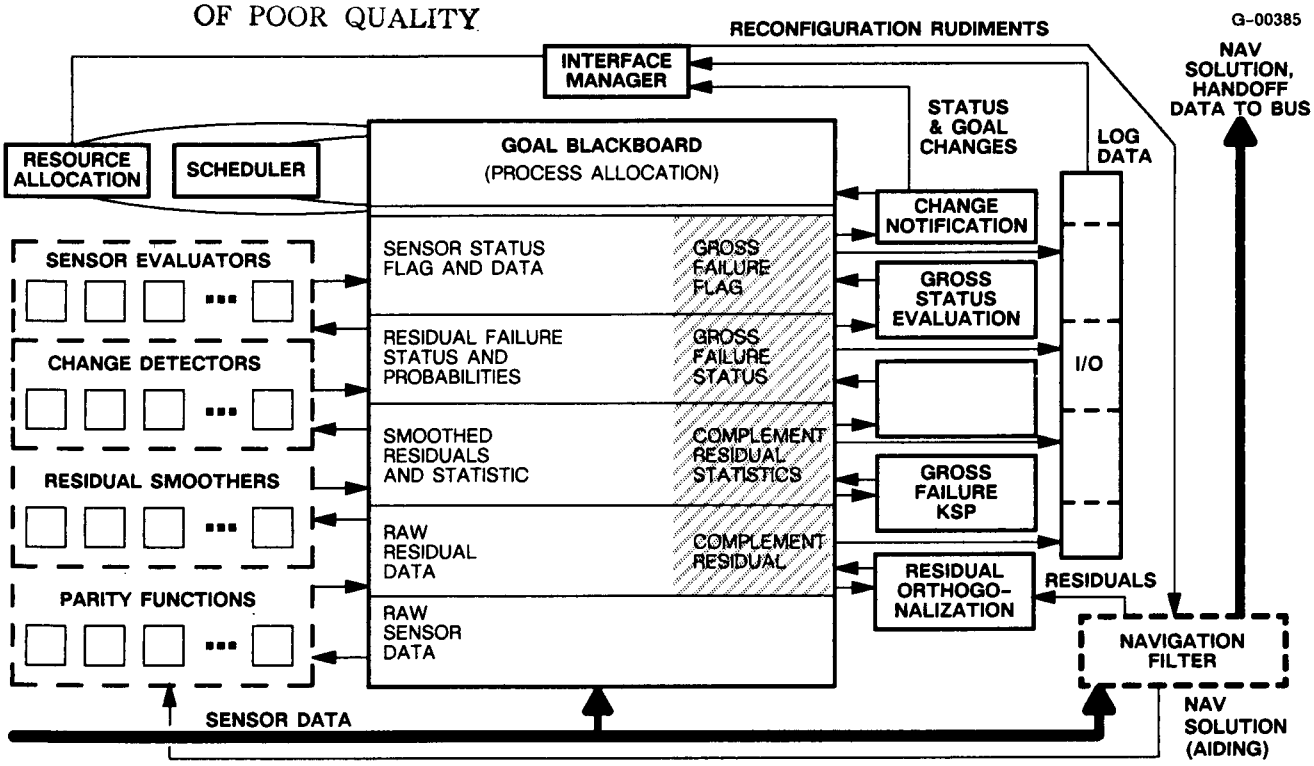


Figure 6. The Navigation Source Manager

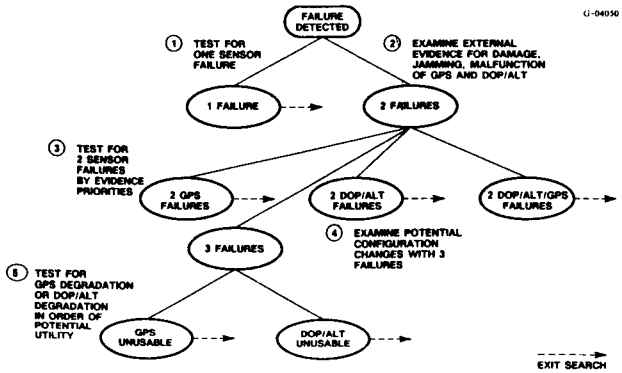


Figure 7. A Constrained Search for Sensor Failures

signal and noise levels are interpreted in terms of a quantitative health figure.

Results of the FDI process are sent to the Navigation Source Manager interface for dissemination to other interfaces of the Expert Navigator. The three types of results envisioned are detailed in messages labeled "equipment.health", "equip.fail.diagnosed" and "equip.fail.unresolved." The first two signal successful completion of an FDI cycle, and provide sensor health figures; the third message simply warns that the FDI tests were inconclusive or could not be completed in the allotted time.

Managing Evidence and Observations: Event Diagnosis

In ATN, diagnosis of the state of health of the current navigation mode is distributed over several modules that are each "experts" in some particular area. The Navigation Source Manager uses parity functions based on engineering models to detect classes of sensor failures. The Event Diagnosis module, in contrast, reasons using pilot or wingman observations combined with evaluations of equipment health.

Constraints on the Event Diagnosis module are that it must deal with information that can be volunteered by the pilot at any time or that may take time to obtain (such as wingman information communicated by radio). In addition, this information may be vague or uncertain (e.g., "possible map error"). Within these constraints, the Event Diagnosis Module must maintain and update an evaluation of evidence of system health.

Several ways of managing evidence have been proposed in the AI literature. Such methods are probability theory (Refs. 7 and 15) confidence factors (Ref. 16), Dempster-Shafer theory (Refs. 17 and 18), endorsements (Ref. 19), fuzzy logic (Ref. 20) and incremental evidence (Ref. 6). Several techniques are reviewed and compared in Ref. 21. In ATN, the technique of probability propagation in causal networks (Refs. 7 and 22) was chosen for the rigor of the mathematical theory of probability and the locality of computation as developed by Pearl. An example of a causal network in the Event Diagnosis module of ATN is given in Fig. 8. In such a network, nodes represent random variables and links have conditional probabilities of the destination random variable

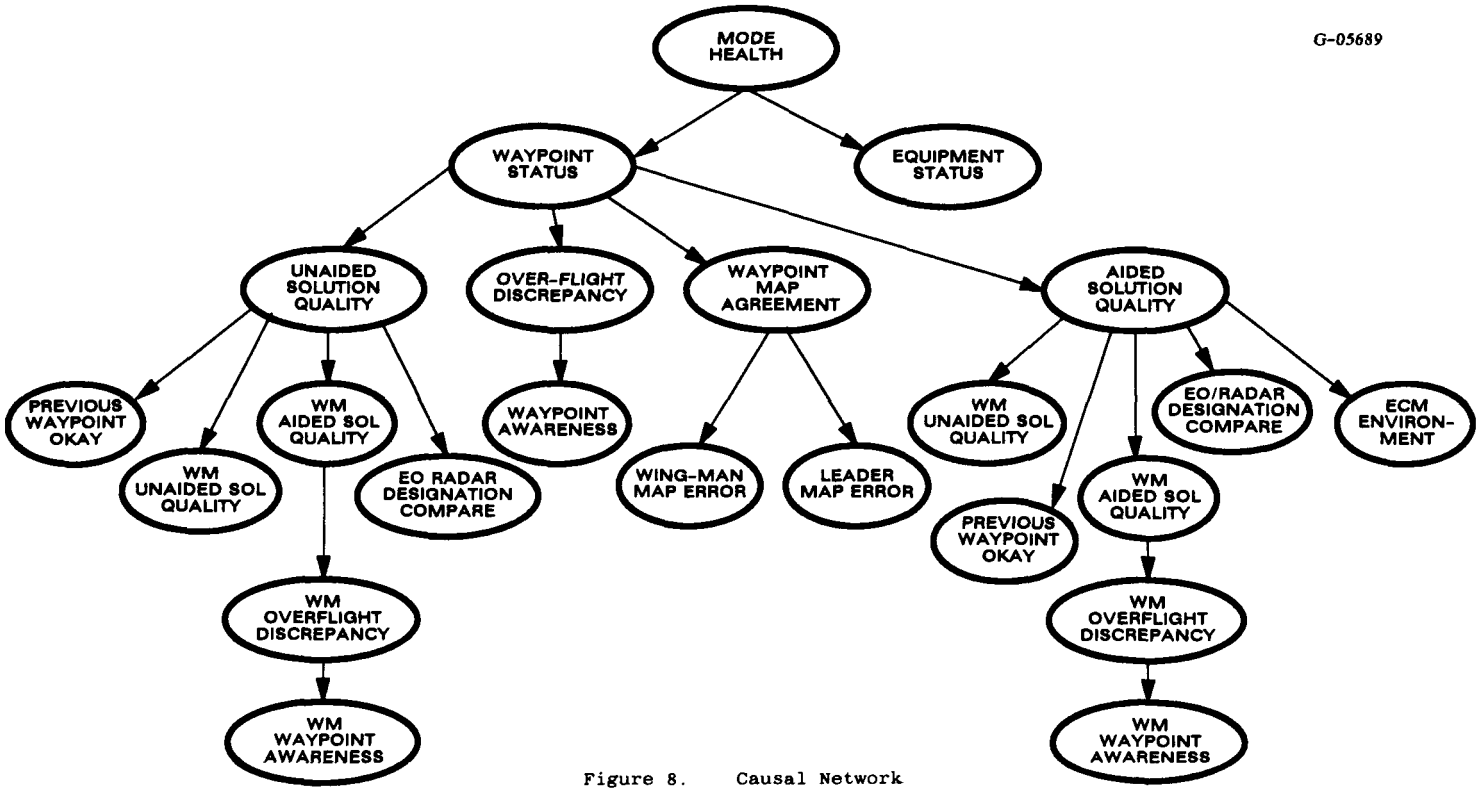


Figure 8. Causal Network

value given the source random variable value. An important assumption of such a tree is that 2 nodes, A and B, separated by a third node C are conditionally independent given C.

In using a causal network, all probability distributions are interpreted in the Bayesian or subjective sense of measures of belief. An a priori belief is assigned to the root node (typically the hypothesis under consideration in the problem). Using the conditional probabilities attached to the links, an a priori distribution can be propagated to all the leaf nodes (typically variables that can be observed). Conversely, if an observation is made (i.e., the value of a leaf is determined), a posteriori probabilities can be propagated up the tree to give an a posteriori distribution of the root node. Thus local calculations update belief in the value of the root node and combine evidence modeled by leaf node observations. For example, in the causal network of Fig. 8, the root node represents the health of the current navigation mode. Leaves represent observables such as the ECM environment or the leader is opinion of map quality.

Causal networks provide a method for combining pieces of evidence in a timely fashion and determining current measures of belief in various hypotheses. What they lack is a method for prioritizing observations. To address this need, a small production system was designed for the Event Diagnosis module to prioritize observations. This system was kept small to ensure fast execution (typically, in modules that involve pilot interaction, events occur on the order of seconds). This information prioritization system incorporates heuristics such as "information from other ATN modules or the JTIDS community costs no time to obtain" or "if JTIDS is not available, information from wingman over the radio will take much longer than information from the leader" or "the first request to the wingman should be an alpha-check." Such heuristics provide the Event Diagnosis module a method of efficiently gathering information as well as a method of incorporating and evaluating the information.

Managing Crew Interaction: Pilot Vehicle Interface (PVI) Manager

The purpose of the Pilot Interface (PVI) Manager is to manage communication between the ATN and the pilot. The PVI manager functionality is depicted in Fig. 9. Communication from the ATN to the crew is controlled by the Request Manager and the Communication Priority Manager. The Request Manager receives and prioritizes requests from the ATN, sends them forward to be posted and matches appropriate responses. The Communication Priority Manager arbitrates usage of the ATN icon window on a heads-up display (HUD) between requests for pilot information (suspect map error?) and ATN advisories (recommend downmode). Behavior of these two submodules is determined by a programmable Display Moding and Symbology submodule. Communication from the pilot to the ATN via voice and keypad is filtered by the Pilot Input Manager.

Appropriate content, priority and time frame of pilot interaction vary significantly during the mission. To appreciate the range of variation, consider the generic air-to-ground mission profile shown in Fig. 10.

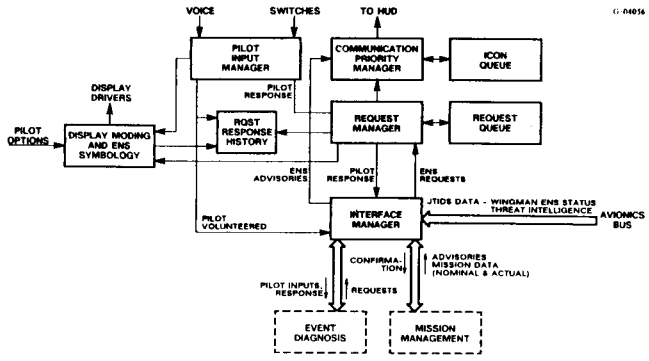


Figure 9. Pilot Vehicle Interface Manager

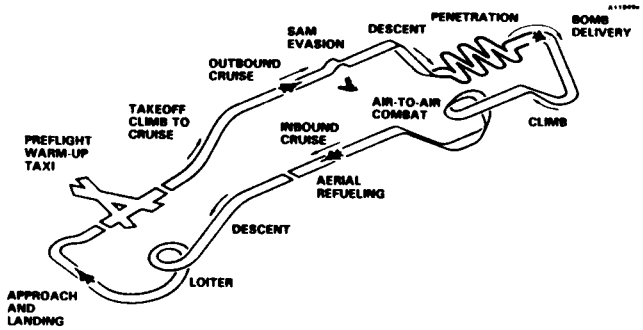


Figure 10. Air-to-Ground Mission Profile

Crew priorities and attention allocation to navigation during the mission can be characterized by five segments:

- Ground Alignment/Climb/Cruise - The INU is initially aligned and its quality is assessed from alignment status. Navigation awareness during this early mission phase is moderate and crew workload is relatively low.
- Low-level Ingress - Here detectability and navigation robustness are primary concerns. Navigation accuracy requirements are not stringent. Crew Workload is relatively high and pilot attention available for navigation diagnosis is limited.
- Pre-IP/IP - Navigation awareness peaks as final preparations for the attack phase are made. Navigation accuracy, as it affects bombing system performance, is a primary concern.
- Post IP/Attack - The crew assumes that the navigation system is working as confirmed at the IP. No time is available for diagnosis or manual moding as the attack flight profile is executed.
- Egress - Navigation requirements are relatively relaxed. Reliable navigation is required for selected points in this phase such as tanker rendezvous.

In view of the wide variations of navigation priority and available crew attention, it is clear that the display behavior should adapt to the current phase. To support this desirable behavior, the Display Moding submodule (AFO) provides a pilot programmable database of timeouts for ENS request/advisory icons and thresholds for alternative displays.

A baseline set of display timeouts and the post IP display moding logic for the ATN demonstration are summarized in Table 1. As indicated in the table more time is allocated for responses during the early and post-attack phases than during the ingress and Pre IP phases. By convention, a timed-out response will be taken as a positive response (e.g., pilot

TABLE 1.

BASELINE DISPLAY TIMEOUTS AND MODING

BASELINE TIMEOUTS - SECONDS

MISSION PHASE / ICON TYPE	GROUND-CRUISE	INGRESS	PRE-IP	EGRESS
EVENT DIAGNOSIS REQUEST	20	10	10	20
MODING ADVISORY	20	10	10	20
WAYPOINT PROMPT	60	30	20	60

POST-IP DISPLAY MODING

- ON BOMBING SYSTEM ACCURACY DEGRADATION BELOW PILOT-PROGRAMMED THRESHOLD, SWITCH TO PILOT SPECIFIED ALTERNATE HUD DISPLAY
- PILOT CAN
  - UTILIZE NEW DISPLAY MODE WITH ALTERNATE DELIVERY
  - PERFORM MANUAL OVERRIDE TO NOMINAL DISPLAY

agrees with recommended mode change). In addition, the display moding rule for the attack phase follows the principle that no time is available for diagnosis. If a significant failure occurs post-IP, go to the alternative delivery profile.

4. CURRENT STATUS AND PLANS

Detailed design of the ATN Demonstrator system was recently completed. As illustrated by the examples presented in this paper, special emphasis was placed on efficiency of combination and use of evidence within each module to achieve real-time operation. These designs will be simulated using the tools and techniques described in Ref. 12 as a means of tracking performance of actual software relative to allocated processing budgets.

ATN will be implemented on a small number of general purpose laboratory processors which communicate via medium-speed data links. The global message passing mechanism and module scheduling mechanisms will be provided by the Activation Framework Shell (Ref. 9).

It is anticipated that ATN will run in real-time, even on this small collection of processors. It is also anticipated that Pilot interaction will be managed in a reasonable manner via the programmable PVI. Further refinement of ATN system behavior will be accomplished through subjective laboratory testing of the demonstrator and a subsequent cockpit simulator test program.

REFERENCES

1. S. Berning, D.P. Glasson, and G A. Matchett, "Functionality and Architectures for an Adaptive Tactical Navigator System," Proc. NAECON, Dayton, OH, May 1987.

2. Jones, H.L., and Pisano, A.L., "Adaptive Tactical Navigation," Report No. AFWAL-TR-85-1015, The Analytic Sciences Corporation, Reading, MA, April 1985.

3. Glasson, D.P., and Pomarede, J.L., "Adaptive Tactical Navigation-Phase II Concept Development," Report No. AFWAL-TR-86-1066, The Analytic Sciences Corporation, Reading, MA, September 1986.

4. P.E. Green, "Issues in the Application of Artificial Intelligence Techniques to Real-Time Robotic Systems," Proc. 1986 ASME Computers in Engineering Conference, Chicago, IL, July 1986.

5. P.E. Green, "Resource Limitation Issues in Real-Time Intelligent Systems," Proc. SPIE Conf. on Applications of Artificial Intelligence III, Vol. 635, Orlando, FL, April 1986.

6. P.E. Green, "Working Paper on the Incremental Evidence Technique," Technical Report EE86RTAIRG03, Worcester Polytechnic Institute, Worcester, MA, September 1986.

7. J. Pearl, "Fusion, Propagation, and Structuring in Bayesian Networks," Technical Report CSD-850022, Cognitive Systems Lab., CS Dept., UCLA, Los Angeles, CA, March 1986.

8. P.E. Green, "AF: A Framework for Real-Time Distributed Cooperative Problem Solving," Collected Papers of the 1985 Distributed AI Workshop, Sea Ranch, CA, pp. 337-356, November 1985.

9. S. Wyss, "A Software Methodology for Distributed Real-Time Intelligent Systems," MS Thesis, Worcester Polytechnic Institute, Department of Electrical Engineering, Worcester, MA, 1986.

10. L. Erman, F. Hayes-Roth, V.R. Lesser, and R.D. Reddy, "The HEARSAY-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," ACM Computing Surveys, Vol. 12, pp. 213-253, 1980.

11. P.E. Green, "Resource Control in a Real-Time Target Tracking Process," Proc. Fifteenth Asilomar Conference on Circuits, Systems, and Computers, pp. 424-428, Pacific Grove, CA, November 9, 1981.

12. J.M. Poole, D.P. Glasson, and T.S. McDermott, "Real-Time Intelligent System Analysis by Discrete-Event Simulation," Presented at the Eastern Simulation Conference, Orlando, FL, April 1987.

13. Weiss, J.L., Pattipati, K.R., Willsky, A.S., Eterno, J.S., and Crawford, J.T., "Robust Detection/Isolation/Accommodation for Sensor Failures," NASA Contractor Report No. 174797, Alphatech, Inc., September 1985.

14. Lou, J.L., Willsky, A.S., and Verghese, G.C., "Optimally Robust Redundancy Relations for Failure Detection in Uncertain Systems," Automatica, Vol. 22, No. 3, pp. 333-344, 1986.

15. Duda, R.O., Horb, P.E., and Nasson, N.J., "Subjective Bayesian Methods for Rule-Based Inference Systems," Technical Note 124, AI Center, SRI International, Menlo Park, CA, 1976.

16. Shortliffe, E.H., "Computer-Based Medical Consultation: MYCIN," Elsevier, NY, 1976.

17. Shafer, G., "A Mathematical Theory of Evidence," Princeton University Press, Princeton, NJ, 1976.

18. Kyberg, E., "Bayesian and Non-Bayesian Evidential Updating," Artificial Intelligence 31, 1987.

19. Cohen, P.R., "Heuristic Reasoning About Uncertainty: An Artificial Intelligence Approach," Potman, London, 1985.

20. Zadeh, C.A., "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems," Fuzzy Sets and Systems 11, 1983.

21. Prade, H., "A Computational Approach to Approximate and Plausible Reasoning with Applications to Expert Systems," IEEE Transactions on Pattern Analysis and Machine Intelligence 7-3, 1985.

22. Prakash, P.S., Shafer G., "Propagating Belief Functions with Local Computations," IEEE Expert, 1986.

SOFTWARE FOR INTEGRATED MANUFACTURING SYSTEMS Part I

A. W. Naylor and R. A. Volz
The Robotics Research Laboratory
The University of Michigan

Abstract

For several years the University of Michigan has been developing a broad, unified approach to programming manufacturing cells, factory floors, and other manufacturing systems. It is based on a blending of distributed Ada, software components, generics and formal models. Among other things the machines and devices which make up the components, and the entire manufacturing cell—machines, devices, software—is viewed as an assembly of software components. The purpose of this project is to reduce the cost, increase the reliability and increase the flexibility of manufacturing software.

This paper gives an overview of the approach and describes an experimental generic factory floor controller that has been developed using the approach. The controller is "generic" in the sense that it can control any one of a large class of factory floors maing an almost arbitrary mix of parts.

1 Introduction

The basic difficulties with current software for integrated manufacturing system is that it is too expensive, too inflexible, and needs greater reliability. For the past five years the University of Michigan has been developing an approach to this software which attempts to address these difficulties. This paper reviews this approach and then discusses an experiment which uses the approach.

2 The Approach

The approach is based on five assumptions or beliefs.

1. Manufacturing software should be in the mainstream of modern software.

   It is unrealistic to expect to solve the problems of manufacturing software if we try to develop solutions that are peculiar to manufacturing. Manufacturing software is—after all—software and most of its problems are problems shared by software in general. Manufacturing software must take advantage of the tools and techniques being developed by modern software engineering. For example, manufacturing software should be written in modern general purpose languages and not tailored "manufacturing languages."

2. Software should be created as an assemblage of software components.

   In other words, we should use object oriented programming. For example, the programmer should be able to view a robot, vehicle, material handling system, or a factory floor as a software component. The programming should be concerned with two things: the interface to the component and how the component works, that is, its semantics. Further, there should be orderly ways to assemble components to create new, larger components, example, create a cell component from machine and robot components.

   The advantages are that (a) components can be reused and replaced thereby decreasing cost and increasing flexibility. Further, the object-oriented approach will increase software reliability.

3. This should be done in a largely common—eventually distributed—language environment.

   The use of object-oriented programming really requires a common language environment. However, this does not mean that portions of a large software system cannot be written in other languages. For example, NC machines will undoubledly be programmed using parts programming languages. These will be encapsulated into software components which externally present a public interface in the common language environment.

   Since manufacturing systems can involve hundreds or even thousands of programmable devices and these will be able to communicate with one another, we are inevitable faced with distributed systems. Rather than writing many separate programs which communicate with one another, we believe the entire system should be written as one (of course, highly structured) program

in a distributed language. The advantages are that (1) it relieves the programmer of writing communication software, (2) allows the programmer to think about the program at a level which largely suppresses the processor boundaries, and (3) allow the language translation system to check for bugs across the entire software system.

4. Explicit formal semantic models are required.

   Much of manufacturing software is concerned with real-time control of manufacturing systems, and real-time control inevitably requires a model for the controlled system. For example, the control software for a factory floor requires an understanding of how the factory floor works, that is, an understanding of its semantics. Thus, in addition to using software components, we must also be able to model their semantics.

5. Generics will amplify software reusability.

   By "generics" we mean skeletons for software components which can be instantiated as actual components. The instantiation process requires that information be supplied which allows the skeleton to be fleshed-out into an actual component. for example, one can imagine a generic material handling system which requires information describing the vehicles and the path layout. This would allow the same software to be used with different fleets of vehicles and different path layouts.

6. The experiment.

   We have developed a generic factory floor controller. It expects to be given a model of the factory floor, process plans, and orders. Based on this information, the generic factory floor controllers determines the appropriate sequence of commands to the factory floor. This is done in real-time. The basic control algorithm is a search algorithm which explores possible future scenarios and selects the best next step, and then carries out the cycle again.

SOFTWARE FOR INTEGRATED MANUFACTURING SYSTEMS PART II

R A. Volz and A. W. Naylor
The Robotics Research Laboratory
The University of Michigan

July 28, 1987

Abstract

Part I presented an overview of the Michigan unified approach to manufacturing software. This paper considers the specific characteristics of the approach that allow it to realize the goals of reduced cost, increased reliability and increased flexibility. It examines why the blending of a components view, distributed languages, generics and formal models is important, why each individual part of this approach is essential, and why each component will typically have each of these parts. An example of a specification for a real material handling system will be presented using our approach and compared with the standard interface specification given by the manufacturer. Use of the component in a distributed manufacturing system will then be compared with use of the traditional specification with a more traditional approach to designing the system.

This paper will also provide an overview of the underlying mechanisms used for implementing distributed manufacturing systems using our unified software/hardware component approach.

1 Introduction

Part I of this paper identified the following five concepts as the keys to our approach to manufacturing software:

1. Manufacturing software should be built in modern extensible general purpose languages.
2. Manufacturing software should be object oriented and created as assemblages of components.
3. Explicit formal semantic models are required.
4. Generics will amplify software reusability.
5. The above should be carried out in a largely distributed language environment.

In this paper, we explore the motivations for the use of these key concepts further and discuss an example of applying them to a material handling system.

The goals of our manufacturing software research are:

1. To develop techniques for building manufacturing software in a more reliable, less costly manner than present techniques.
2. To develop techniques for cost effective maintenance of manufacturing software.
3. To develop techniques for producing reusable software.
4. To develop techniques for producing portable software.
5. To develop techniques supporting a components industry.

Each of the five key concepts supports one or more of these goals.

2 Use of Modern Extensible General Purpose Languages

William Boller of Hewlett Packard[1] has recently stated, with respect to manufacturing software, that "Complexity is the root of all evil." Managing complexity is one of the most important things that must be done to develop reliable software. Managing complexity has also been one of the principal goals of software engineering research during the past two decades, and significant results have been obtained [1,2], including:

- Modular approaches to program development that provide a conceptually clear view of the system being implemented — This aids software production and maintenance.
- Powerful program verification techniques that, while not totally automatic — no existing technique is for programs of any size —, do automatically detect a very large fraction of program errors, thus reducing the cost of program development.

[1] "The Factory of the Future," *The Economist*, May 30, 1987.

- Modular approaches to program development that reduce compilation costs.
- Highly expressible and extensible capabilities.
- Portability of programs from one system to another.
- Techniques for managing concurrent/parallel real-time tasks.

Obtaining these same advantages for manufacturing software is important, and far more likely to be achieved a standard language is adopted than if a new one is built from scratch.

Among the language mechanisms used to achieve these results are:

- data encapsulation and hiding,
- data and program abstraction,
- strong typing,
- separate compilation (both of different modules and of module specifications and implementations), and
- explicit control of representations — particularly for numerics.

Placing all of these into a special purpose language is a very difficult, time consuming and error prone task. Yet omitting them would be to forgo some of the capabilities needed to achieve our goals [3,4,5].

3 Use Object Oriented Software

Our world is made up of objects, and we are accustomed to thinking about the management of our life in terms of the objects around us and operations that may be performed on them. E.g., I am editing on my computer terminal. I drive my car to work each day. Etc. It is natural to carry this mode of thinking over to our problem solving and system building activities, in which case, it is called object oriented design [6]. This approach helps develop a conceptual clarity of the system being built and organize its complexity.

When coupled with the representation of the object by a specification — the public interface to the object that presents the only ways (operations) by which the object may be accessed — and a body that implements the object, the object oriented approach is the natural mechanism for developing plug compatible components and a whole new approach to the relationship between suppliers and manufacturers.

With a components industry for manufacturing equipment and software in place, manufacturers would specify in a formal way the requirements for the manufacturing equipment they need and the component suppliers would supply manufacturing hardware and software components which would "plug" into the rest of the manufacturers system. Several things derive from this view. First, the industrial manufacturer designs the package specification to provide the view of the manufacturing device necessary for the application at hand. Component suppliers are then given the compiled specification and must provide not only the required hardware, but a body to the component package which is compatible with the manufacturer compiled specification as well. Since the component is now formally specified and can be automatically machine checked for compliance with the specification, several vendors might bid against each other for the job. Second, since the body must reside in the control computer, the supplier must take responsibility for the applications level communication across the network. The supplied software component is directly plugable into the manufacturer's computer. This is exactly the opposite of current practice in which the manufacturer assumes the responsibility for custom designing the hardware and software interfaces for integration of the system.

Third, since suppliers will have a fixed and standard framework within which they must deliver components, it will both be easier to develop custom products and easier to formulate standards when a class of devices has reached maturity.

4 Formal Models

We need models of the factory floor and process plans in order to develop control algorithms. Since we realize the factory floor and process plans as assemblages of software/hardware components, we are, in effect, concerned with formal semantic models for such components. The modeling methodology used is described in more detail in [7,8]. One component may include *models* of other components. The models may then be used in a predictive simulation manner to examine the likely outcome of a possible control strategy before it is actually applied.

Finally, the modeling methodology can be used to represent the process plans that the cell is to implement as well as the actions of the components. The uniform modeling of process plans and software/hardware components simplifies the software structure and allows one to view the process plans as just another component in the system. And, the formal models of process plans can be converted to actual components that drive the operation of the system. At present the translation from the formal models to actual software is performed manually, but conceptually (at present, and in the future actually) they could be converted automatically.

5 Generics

Generics can be used in a variety of ways. The most obvious was stated in Part I, to obtain software reuseability through what amounts to parameterization of the types and functions used in a component. However, generics can be used in other ways as well. They can be used to provide an individualized interface to a component, as will be illustrated below. That is, each user of a component, such as a material handling system,

can instantiate his/her own "view" or interface to the system. In this way, the interface to the system can be simplified.

One can also consider dynamic extensions to generics that would allow a user to create instances of generic components at run-time. In this case, each real component would contain the parameters necessary to complete a generic instantiation of it. The user would just reference the generic component and name a specific real component (for example a specific vehicle from a pool of vehicles in a material handling system) from which an actual instance of the component would be created. Resource managers, in particular, would find it useful to operate in this manner.

6 A Distributed Language Environment

Sec. 2 above described a number of advantages available from modern software engineering tools. These capabilities, however, are centered at the language level. That is, they are achievable for *single programs*. In the manufacturing world, however, we are clearly working across machine boundaries. Even for modest sized systems, there will be multiple control computers that will have to communicate with each other. In order to achieve the full advantages of modern software engineering, then, one should look to distributed program execution, that is, execution of a *single program* across a network of processors. One then obtains the advantages of conceptual clarity, modularity and automatic program verification currently possible with single programs on single machines. The single program view of a distributed system would allow verification to be done across the entire system instead of, as is now the case, only on the subsets of a program residing on a single processor. In addition, it reduces the programmer's view of interprocessor communication to interprocess communication, which is the programmer's natural view of communication; special application level communication protocols become unnecessary, and any lower level protocols become transparent to the programmer.

Our approach to this need has been to adopt a standard programming language intended for real-time operation and develop a distributed version of it. Because it is basically a good language, is subject to intense standardization efforts, and is ostensibly intended for distributed execution, we selected Ada. To achieve distributed execution, we have built a pre-translator that takes a single Ada program as an input and whose output is a collection of pure Ada programs, one for each targeted processor. This is somewhat akin to the way embedded SEQUEL is handled in the DB2 database management system.

Our distributed Ada system [9] allows us to distribute library packages and library subprograms statically among a set of homogeneous processors. We write a single program and use a **pragma** (essentially a complier directive) called SITE to specify the location on which each library unit is to execute. For example, if a simple transport system were controlled by computer number 2 and the cell control using it were on computer 1, a sample of relevant code might look as follows:

```
pragma SITE (2);
package VEHICLE is
        procedure MOVE_FORWARD;
              :
end VEHICLE;
        :
pragma SITE(1);
with VEHICLE;
procedure CONTROL is
  :
begin
  :
  VEHICLE.MOVE_FORWARD;
  :
end;
```

Our translation system would replace the local call to the procedure VEHICLE.MOVE_FORWARD with the appropriate remote call. Similarly any references in CONTROL to data objects defined in package VEHICLE would be translated into appropriate remote references as would task entry calls. Note that the user need only use the normal procedure call mechanism to cause the vehicle to move.

7 Material Handling System Example

[7] describes a generic factory control system that has been built and simulated using the ideas described above. In this section, we explore one component of such a system in more detail, a material handling system.[2] We suppose a material handling system (MHS) that is used to move pallets from one location to another, has a number of vehicles to carry out the moves, and can be utilized by several different parts of the system.

From a hardware/software component, i.e., object, perspective we think of the *relevant* objects in the system and the functions performed on them *by the parts of the system that need to use the MHS*. In the simplest view of this example, the relevant objects (from the perspective of the user of the MHS) are the *MHS* itself, the *pallets* that are to be moved, and the *locations* to/from which they pallets are moved. The vehicles used are not relevant to the user, and thus should remain hidden from the view provided to the MHS user. Since there are potentially multiple parts of a factory system, e.g., multiple cells, that could have need to more or less independently make use of the MHS, the MHS should support a concept of multiple users. However, for any one user, the view of the MHS should not have to be cluttered with unnecessary detail about the other users. Generics allows us to achieve this.

[2] We have actually implemented a more complete abstraction of a material handling system than that described here.

We show here a simplified (only in the sense of a reduced set of operations supported by the MHS) generic interface to the component:

```
generic
package GENERIC_MHS is
    type PALLET is private;
    type LOCATION is private;
    L1,L2,L3,L4,LN: constant LOCATION;
    type MOVE_ID is private;
    type ACKNOWLEDGE is (OK, BUSY, FULL);
    type MOVE_STATUS is (WAITING, MOVING, DONE);
    MHS_NONRESPONDENT: exception;
    procedure ALLOCATE_PALLET(P: out PALLET; ACK: out ACKNOWLEDGE);
    function WHERE_PALLET(P: PALLET) return LOCATION;
    procedure REQUEST_MOVE(P: PALLET; L: LOCATION;
            M: out MOVE_ID; ACK: out ACKNOWLEDGE);
    procedure MOVE_STATUS(M: MOVE_ID; MS: out MOVE_STATUS);
    private
        .
        .
end GENERIC_MHS;
```

A cell controller using the MHS might look something like the following:

```
pragma SITE(1);
with GENERIC_MHS;
procedure CELL_CONTROL is
    package LOCAL_MHS is new GENERIC_MHS;
    use LOCAL_MHS;
    P1, P2: PALLET;
    MS: MOVE_STATUS;
    M1, M2: MOVE_ID;
    ACK: ACKNOWLEDGE;
begin
    ALLOCATE_PALLET(P1,ACK);
        .
        .
    MOVE_REQUEST(P1, L1, M1, ACK);
        .
        .
    ALLOCATE_PALLET(P2,ACK);
        .
        .
    MOVE_REQUEST(P2, L2, M2, ACK);
        .
        .
    MOVE_STATUS(M1, MS);
        .
        .
end CELL_CONTROL;
```

There are a number of points to notice about this example. First, the instantiation of the generic MHS provides a clear and straightforward interface to the MHS, expressed in terms a user would find convenient in dealing with the MHS component. The command names have been chosen to have an implied semantics indicative of the operation to be performed. Reading the control program is straightforward. The types provided are just those needed to talk about the objects associated with the MHS. Irrelevant details are hidden.

This example is also presented in terms of a distributed system. The cell controller is indicated as being located on site 1. It is not stated where the MHS is located, and the only fact about its location that is relevant to the cell controller is the fact that it might be on a different computer. In this case, the function and procedure calls to the MHS object will involve remote calls to the site at which the actual MHS controller is located. This possibility is manifested in the generic MHS through the exception NON_RESPONDENT. When the user (CELL_CONTROL in this case) instantiates a copy of GENERIC_MHS, that copy will appear on the same computer as CELL_CONTROL. Hidden in the implementation of the local copy, LOCAL_MHS, is a periodic checking of the communication line and a timeout on the return from the remote procedure calls. If the communication line fails or the actual MHS does not respond within its prescribed time, the implementation of LOCAL_MHS will raise the exception NON_RESPONDENT, and CELL_CONTROL can deal with this as necessary. Only the abstraction representing failure of the actual MHS is appropriate for CELL_CONTROL to be concerned with; of course, other kinds of failures could equally well be represented.

The translation system supporting distributed program execution replaces all calls to remote components with the appropriate communication routines and implicitly manages communication routing.

Also note that by focussing on an object oriented view of the components the potential for standardization is increased. It is now easy to think in terms of standardizing the interface to a single component type, such as an MHS, without having to consider any other component types in the system. The types, procedures, functions, exceptions and call profiles become the formal expression of the standard. Moreover, the syntactic compliance to a standard can be automatically checked by the system compiler.

8 Conclusion

A coherent approach to manufacturing software is one of the most important building blocks needed for U.S. industry to truly develop integrated manufacturing systems. We have described a concept by which coherent manufacturing can be accomplished. However, the theory is not yet complete. Indeed, much remains to be done. Extensions to the formal modeling system are needed to more fully handle generics and distribution of components. The process of instantiation of generics to real components must be extended to allow dynamic instantiations. Distributed languages must be studied in a more general context of multiple forms of memory interconnections, multiple possible binding times, and various degrees of homogeneity (e.g., see the major dimensions of a distributed language defined in [10]).

Yet, we have accomplished enough to demonstrate the viability of the major underlying ideas. A primitive version of a distributed Ada translation system *is* working, and a limited generic real-time factory controller *is* operational, with real factory components replaced by simulation. We believe that when it is fully developed, the approach presented here can become the heart of future integrated manufacturing systems.

References

[1] B. Liskov and J. Guttag. *Abstraction and specification in program development.* MIT Press, Cambridge, MA, 1986.

[2] J.C. Cleaveland. *An Introduction to Data Types.* Addison-Wesley, Reading, Mass, 1986.

[3] E. Denert. *Trends in Information Processing Systems. 3rd Conference of the European Cooperation in Informatics*, chapter Software Engineering: Experience and Convictions, pages 16–35. Springer-Verlag, October 1981.

[4] S.N. Woodfield, H.E. Dunsmore, and V.Y. Shen. The effect of modularization and comments on program comprehension. In *5th International Conference on Software Engineering*, pages 215–23, March 1981.

[5] L. Varga. Specifications of reliable software. *Tanulmanyok Magy. Tud. Akad. Szamitastech. And Autom. Kut. Intez. (Hungary)*, (113):309–25, 1980.

[6] Grady Booch. *Software Engineering with Ada.* Benjamin/Cummings, second edition, 1987.

[7] A.W. Naylor and R.A. Volz. Design of integrated manufacturing system control software. *IEEE Trans. on Sys., Man, and Cybernetics*, submitted 1987.

[8] A.W. Naylor and M.C. Maletz. The manufacturing game: a formal approach to manufacturing software. *IEEE Trans. on Sys., Man, and Cybernetics*, SMC–16:321–334, May-June 1986.

[9] R.A. Volz, P. Krishnan, and R. Theriault. An approach to distributed execution of Ada programs. In *NASA Workshop on Telerobotics*, to appear 1987.

[10] R.A. Volz, T.N. Mudge, G.D. Buzzard, and P. Krishnan. Translation and execution of distibuted Ada programs: is it still Ada? *IEEE Transactions on Software, Special Issue on Ada*, to appear 1987.

COMMUNICATION AND CONTROL IN AN INTEGRATED MANUFACTURING SYSTEM[†]

Kang G. Shin, Robert D. Throne, and Yogesh K. Muthuswamy

Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

Abstract

Typically, components in a manufacturing system are all centrally controlled. Due to possible communication bottlenecking, unreliability, and inflexibility caused by using a centralized controller, a new concept of system integration called an *Integrated Multi-Robot System* (IMRS) was developed. The IMRS can be viewed as a distributed real-time system.

This paper presents some of the current research issues being examined to extend the framework of the IMRS to meet its performance goals. These issues include the use of communication coprocessors to enhance performance, the distribution of tasks and the methods of providing fault-tolerance in the IMRS. An application example of real-time collision detection (as it relates to the IMRS concept) is also presented and discussed.

1 Introduction

Conventionally, components in a manufacturing system are all centrally controlled; that is, control tasks for the system may be distributed over a network of processors or reside in a uniprocessor but are all executed under directives of one central task. The work by Maimon [1] [2] is primarily concerned with dynamically determining how to utilize the resources within a workcell to achieve a certain objective, where an activity controller provides for centralized control of the workcell. The work at the National Bureau of Standards on their Automated Manufacturing Research Facility (AMRF) system [3] [4] [5] deals with real-time control of a workcell using strictly hierarchical control. Their system is data-driven and based on state tables at each level of hierarchy. At each level, these state tables are updated on the basis of (1) commands from the next higher level, (2) results of processes at the next lower level, and (3) sensor inputs at the current level. While information can be exchanged across one level, control is strictly vertical. The state table approach allows for recovery from various undesirable events (so long as these events are accounted for in one of the states), but the overall sequence of operations is hidden from the user.

Due to possible communication bottlenecking, unreliability, and inflexibility caused by using a central controller, we have proposed a new concept of system integration, called an *Integrated Multi-Robot System* (IMRS) [6] [7]. An IMRS is defined as a collection of robots, sensors, computers, and other computer controlled machinery, such that

- each robot is controlled by its own set of dedicated tasks, which communicate to allow synchronization and concurrency between robot processes,
- tasks execute in parallel,
- both centralized and decentralized control concepts are used, and
- tasks may be used for controlling other machinery, sensor I/O processing, communication handling, or just plain computations.

In the above definition (and in what follows) the term "process" refers to an *industrial* (but not computational) process, which could be decomposed into several *subprocesses*. Each subprocess may be accomplished by executing a software *module* in a computerized controller. Each module can be decomposed further into computational *tasks*.

The goal of an IMRS is to outperform its counterparts by better utilization of physical space and computer capabilities, increased throughput, greater flexibility, improved fault-tolerance, and the capability of handling diverse manufacturing processes. In order for an IMRS to effectively utilize the available resources, it must make maximum use of the possible parallelism between processes and tasks. In an IMRS there are five different classes of interaction between subprocesses[7]:

- *Independent Processes*: the work of each subprocess is independent, and the actions taken by each subprocess

[†] This work was supported in part by the NASA Johnson Space Center under Grant No. NCC–9–16 and the US Airforce Office of Scientific Research under Contract No. F33615–85–C–5105.

to accomplish its goal are also independent. Indirect influence through state variables is the only way the subprocesses of an independent process may be related.

- *Loosely Coupled Processes*: the subprocesses perform independent work, but the actions taken by each subprocess depend on the actions of the other subprocesses, e.g., two robots sharing the same workspace or set of tools.
- *Tightly Coupled Processes*: the work of the subprocesses depend on each other, and the actions taken to carry out subprocesses also depend on each other. Carrying a long steel beam with two robot arms is a typical example of this class.
- *Serialized Motion Processes*: the works of the subprocesses depend on each other, yet the actions taken to accomplish each subprocess are independent, e.g., assembly.
- *Work Coupled Processes*: the processes monitor each other. Should one process crash due to a computer or device failure, the other computer or device will attempt to take over the responsibilities of the failing device or computer.

Using the above classification, a logical communication architecture called *module architecture* and those primitives necessary for an IMRS are identified in [7]. The module architecture for an IMRS is an n-ary tree that is formed by *task creation*. When a task is created, it becomes a child task of the task that created it. This parent/child relationship always exists, but the amount of communication between the two will be different according to the class of process the tasks are controlling. Under most circumstances, communication channels among child tasks will be directly established, with the parent task playing a minor role. This is defined as *horizontal communications*. However, in some cases the parent must tightly control its child tasks. This is defined as *vertical communications*. Note that these two approaches represent decentralized and centralized controls, respectively. A *proprietor* or *administrator* task is used to provide exclusive access to shared resources (e.g., the right to change a state variable) and resolve conflicts among different concurrent tasks.

We assume that processors controlling devices in one workcell communicate over a common bus or a local area network[1], while GM's Manufacturing Automation Protocol (MAP) [8] is used for communication between workcells. MAP is a protocol for local area networks based on the OSI (Open Systems Interconnection) Reference Model developed by ISO and CCITT. It is a seven layer communication protocol which uses a token passing bus based on the IEEE 802.4 standard [9] [10] as the physical layer. The application layer of MAP specifies the use of the Manufacturing Message Specification (MMS) [11] [12] for communication with manufacturing and process control devices. For time critical applications, the upper four layers of the seven layer ISO protocol are removed, leaving a three layer protocol called the *MiniMAP*. Thus, MiniMAP does not conform to the OSI standard since it is incapable of peer open system communication. MiniMAP is suitable for unintelligent devices such as sensors that do not need to communicate outside their interconnection network. MAP/EPA is composed of both the full seven layer MAP and the three layer MiniMAP.

In the context of the IMRS concept, we discuss in subsequent sections various issues in system integration, such as architectures for high performance intertask communications, the distribution of device controllers among the networked computers, and graceful degradation in case computers and devices fail. Communication bottlenecks should be avoided with any distributed system, particularly with a real-time system. In addition, the assignment and scheduling of tasks on a processor in such a system is of paramount importance. In order to minimize communication bottlenecks and allow for real-time task management, the use of a communication coprocessor is discussed in Section 2. The distribution of tasks on a distributed system has been studied previously. Section 3 discusses some of the issues involved in task distribution in a real-time control environment like an IMRS. Section 4 discusses fault-tolerance in the IMRS, particularly the problems with work coupled processes. Section 5 discusses an application of the IMRS concept to real-time collision detection and avoidance. Finally, Section 6 summarizes the paper.

2 IMRS Communications

An IMRS can be considered a distributed real-time system, with each of the workcells considered as a node. A workcell refers to a set of processes which are grouped together either due to their functional relationship or due to physical proximity of the devices they use. Communications within the same workcell are usually more intense and time-constrained than the communications taking place between different workcells.

We will consider the use of a port-based communication architecture for the IMRS because of its many advantages such as modularity, flexibility, and programmability (see [7] for more on these). In this architecture, each task is associated with some ports to communicate with other tasks. These ports are logical entities and may be mapped onto physical ports on processor nodes on which their associated tasks are located. It is natural to decompose each node's function of the IMRS into communications and applications. For the high performance required for the IMRS, the former will be handled by a dedicated processor called a *communication processor* (CP) and the latter by an *application processor* (AP). The idea of using hardware support for interprocess communication has been proposed elsewhere [13], though not in the context of real-time control. The AP may either be one physical processor or multiple processors. The CP is responsible for all the

[1]That is, a network consisting of only the processors and devices within one workcell.

communications associated with the tasks residing at the node and the AP is responsible for the necessary computation, e.g., execution of a robot's motion.

The processes within a workcell are accomplished by executing a set of tasks, possibly on different processors. A contemporary workcell consists of a number of coprocessors which execute different tasks and also has a CP which is responsible for communicating with the other workcells.

The inter-node access protocol will play a key role in the overall system performance. Notable among popular performance parameters are: *response time*, *throughput*, *availability*, and *fairness*. Response time is composed of nodal computation time at each layer, queueing delays at each layer and at each node, and the actual propagation time along the network. For example, with the IEEE 802.4 token passing scheme (used by GM's MAP), an upper bound exists for the time a node will have to wait to transmit some of its data. The throughput of the network basically depends on the buffering capacity of the destination node or of any intermediate nodes (i.e., gateways) the messages must pass through. Effective throughput is a function of the number of retransmissions required due to transmission errors. The availability of the network depends on the reliability of the components used in the network. For example, if the node with the token fails, the network will be unavailable until error recovery procedures reconfigure the system and generate a new token. Finally, the fairness of the network depends on the load demanded by each user and the optimization the network provider is trying to achieve. For example, if the network provider optimized mean response time in the network, then it is better to allow transmission of users' packets equally. On the other hand, if the network provider optimized throughput, then it is better to allow transmission of packets from users who have the maximum demand. For more information, see [14].

Unlike the inter-node communications, the organization and communications within a workcell node are determined by a number of other issues related to the *message handler* (MH). The MH is a task responsible for interfacing each task on the workcell with its environment. Each task is associated with a MH task, and the aggregation of all the MH tasks at a node resides in the CP and acts as the communication interface for all the tasks associated with that node. This aggregation will henceforth be referred to as MH for simplicity.

The tasks queue up their requests to the CP (either to send or receive messages) on independent queues. The MH task *scans* all these queues and selects a request to service based on some criteria, for example, priority of the requesting task, or the deadline associated with the message to be sent. The task priorities may either be determined *a priori* or dynamically. After sending a message, some tasks might get blocked. Also, when a message arrives from some other node, some tasks might get unblocked. When a task currently executing on the AP gets blocked after sending a message, the MH should decide the next task to be scheduled on the AP. Similarly, when a task gets unblocked, scheduling decisions have to be made by the MH. The methodology of scanning the requests by MH, and an appropriate algorithm to select a request to service will be an important research area.

In addition to acting as the interface for the tasks at a node, the MH is also responsible for maintaining the required degree of fault-tolerance. Failures might be due to device or processor failures. The MH maintains a task map at the node. It is also responsible for unblocking processes that have been blocked by the failure(s) of devices or processors.

The interconnection between the various processors at a workcell depends very much on the pattern and intensity of the communications taking place and also on the stringency of the deadlines associated with the various tasks. A possible interconnection is to connect all the processors in a ring. In this case the time for message passing between any two processors will not be the same, but this allows for expandability. The entire workcell can be visualized as a hierarchy of levels. At the lowest level we have the tasks and the message handlers associated with them. At the next level is the CP associated with that workcell node. The interconnection between the various levels and also the interconnections within the same level have to be determined. Another issue is whether to implement the MH in hardware or software, i.e., whether additional processing power should be provided to each task to implement the MH, or can it be done by the CP at the higher level. This would depend on the fault-tolerance sought for the system as well as the message traffic pattern and intensity.

The protocols used at various levels must be studied. A traditional seven layer protocol at the device controller level may result in deadlines being missed due to the time overhead involved. The sensitivity of the deadlines to various parameters like protocols and interconnection is an important issue and will determine the overall architecture.

3 Distributing IMRS Tasks Among Processors

The distribution of the tasks on the processors will be a key element in determining the overall system cost, performance, and reliability. By examining the parallelism between tasks we get some indication of which tasks can be assigned to the same processor without performance degradation. In addition to the classification of processes, one of the distinct features of an IMRS is to allow *both* vertical and horizontal communications. If the control tasks are distributed over many processors, a hybrid of horizontal and vertical communications between tasks may prove to be beneficial.

For example, serialized tasks can be assigned to the same processor, while assigning independent tasks to the same processor may result in a serious performance degradation. However, since some tasks may depend on state variables modified in another processor, delays in reliably updating these variables must also be included when assigning tasks to processors. If the network throughput is too low, assigning all tasks dependent on one or two key state variables to a single processor (even if the tasks are independent or loosely

coupled) will improve system performance. From these arguments it appears beneficial to group many tasks on a few large (and powerful) processors, but this could lead to a decrease in system performance and reliability.

The system throughput might increase if processors were physically located near the devices to be controlled, each processor having a direct access to the device (i.e., through an I/O port). In this way, a control task for a device could be assigned to its "local processor" and would have to contend with smaller delays over the physical network. There are, however, several drawbacks to this idea. If we depend on having a processor at each device, the potential reliability of networking the computers is seriously diminished. If our real-time performance depends on the presence of such processors, and a local processor fails, we may not be able to have another processor assume the control task and meet the real-time constraints. In addition, if the device only communicates through the local processor's I/O ports, and the processor fails, we may not be able to communicate with a (working) device.

Allowing tasks to communicate directly (horizontally) without a central controller (i) reduces the chances of a bottleneck by exchanging messages among children (instead of always going through the parent), (ii) increases reliability because the subprocesses do not rely on one central control task, and (iii) allows more parallelism because each child task is not blocked as often as in the vertical case, where each child must always wait for a directive from the parent. Tradeoffs in using vertical and horizontal communications for various industrial processes must be analyzed.

Most methods for allocation of tasks in a distributed system are concerned with minimizing a cost function consisting of the sum of processing cost per task on each assigned processor and interprocessor communications (IPC). As was reviewed in [15], these methods are based on graph theory or integer programming or heuristic solutions. Real-time constraints are difficult to impose using the graph theoretic approach, while the integer programming methods allow constraints that all of the tasks assigned to a processor complete within a given time. However, this constraint does not account for task queueing and precedence relations among tasks.

Efe [16] presents a module clustering algorithm minimizing IPC cost without considering constraints, and then moves modules from overloaded to underloaded processors by a module reassignment algorithm. Ma *et. al.* [17] developed an algorithm based on integer programming and the branch-and-bound method. A *task exclusive* matrix defined mutually exclusive tasks that could not be placed on a single processor and *task redundancy* was introduced for system reliability. Chu and Lan [18] chose to minimize the maximum processor workload in the allocation of tasks in a distributed real-time system. Workload was defined as the sum of IPC and accumulated execution time for each processor. A *wait-time-ratio* between two assignments was defined in terms of the task queueing delays. Precedence relations were used to arrive at two heuristic rules for task assignment, which were used in conjunction with the wait-time-ratios to generate a heuristic algorithm for task allocation. Lo [19] proposed the concept of *interference costs* which were inferred when two tasks were assigned to the same processor. This additional cost was used in an effort to reward concurrency.[2]

A criterion to measure task assignments in the IMRS using some of the ideas mentioned above must be developed. It should include task redundancy and mutual exclusion to provide reliability, as well as requirements to group certain tasks to be executed on a single processor. An IMRS should take advantage of as much parallelism as possible, so we will need to include some type of interference penalty. Since we have to deal with a real-time system, we will need to account for queueing delays in the network and within a processor. Finally, the IMRS deals with five basic task classes, and the cost function will have to deal with tasks within the different classes separately.

Once an appropriate cost function is determined, an algorithm to distribute the tasks to the processors must be developed. It is unlikely that a polynomial time algorithm will be found, so faster heuristic suboptimal algorithms may have to be developed.

4 Fault-Tolerance

One of the primary reasons for using a distributed system is to improve the fault-tolerance of the system. The IMRS deals with fault-tolerance through work coupled processes or tasks. These tasks monitor each other so that if the processor or device executing one task fails, the other task on the healthy processor can attempt to compensate. In order to compensate for tasks on a failed processor, the states of those tasks must be known. The update rate between work coupled tasks will affect both network traffic and the load of the associated message handlers. If the state of each work coupled task is updated too often, the network may get congested with state update messages, while if the state is not updated often enough, then recovery of the failed process will be more difficult. Finally, work coupled tasks should not be assigned to the same processor, since failure of that processor will make recovery impossible.

For work coupled tasks to be effective, the system must have the ability to determine that a processor or device has failed. Hence we must first determine methods of detecting the failure of a processor.[3] One such method is sending *heartbeat* messages between processors and assuming the failure of a processor if a response is not received within a prescribed time. A critical issue is the number of such messages and the rate at which they are sent. Depending on the system architecture and timing constraints, it may prove beneficial to have such heartbeat messages sent at different rates for

[2]That is, the assignment of two tasks which could be run simultaneously if assigned to different processors would tend to produce a lower objective function if such an assignment were made.

[3]We assume a *fail stop* system, where a processor stops when it fails.

different processors. These rates would be determined by (1) the minimum allowable recovery time of any of the tasks on the failed processor, (2) the minimum state update rate of any of the tasks on the failed processor, and (3) the assignment of tasks to processors. In addition, the destination of each heartbeat message will depend on these factors, since heartbeat messages could also serve as state update messages.

It may be useful to have specific *health management* tasks to maintain system health, rather than having individual processors acting independently. For example, the health management tasks would be responsible for initiating all heartbeat messages, maintaining tables of healthy processors and the tasks running on those processors, and coordinating recovery when a processor failed. While we may be able to save time and resources by having a single health management task, these benefits would have to be realized at the expense of a centralized system. We would certainly want to have redundant copies of the health management tasks, and may want to have two or more copies of the same task running simultaneously on different processors.

Once a processor is determined to have failed, we must devise mechanisms for ensuring that none of the tasks on the working processors remains "blocked" while waiting for a reply from a task on the failed processor. To accomplish this we can have the message handler maintain lists of incoming and outgoing messages and issue "fake" messages [20] to the blocked tasks. In addition, since the IMRS communicates through ports, the lists of users of a port must be updated to reflect the current state of the system. If a task realizes that one of the work coupled tasks it is monitoring has failed, it should assume that task. Should it then also try to set up a new work coupled task to monitor itself on another processor? One solution might be to have many "overlapping" work coupled tasks assigned when the system is initialized. However, the extra network traffic caused by this solution could be high. Instead, we could have a hierarchical system of work coupled tasks, in which states are updated less often at lower levels of the hierarchy. Such a system for establishing checkpoints in order to achieve resiliency was proposed in [21]. In addition, we would have to determine how many overlapping work coupled tasks would provide the desired degree of fault-tolerance. Similarly, suppose a processor fails and all of the tasks executing on it are assumed by other processors. Now the first processor is restarted. We need to determine a mechanism to dynamically reassign tasks to the processor when it is restarted. Certainly, we do not want to have to shut down the network (and hence the manufacturing) just to reload one processor. The requirements for such a system are presented in [22]. In case there is a sufficient number of failures that not all of the tasks can be run in real-time, these tasks must be executed in a preplanned degraded mode.

5 Real-Time Collision Detection in an IMRS

To discuss their feasibility, the IMRS concepts and solutions must be applied to some realistic examples. Due to its importance, real-time obstacle detection and avoidance has been selected as an application example.[4] This example requires the IMRS to communicate effectively with external sensors, such as vision systems, acoustic range sensors, and various types of proximity sensors. To maintain a high degree of fault-tolerance, each of these sensors should be linked to the computer network. We expect the sensors to provide overlapping coverage, so that if some of the sensors fail information from the other sensors can be used to continue.

Initially, we will assume that the "obstacles" are AGVs conveying parts between workcells. We do not want all devices on the factory floor to stop whenever an AGV nears a device or workcell, only those workcells and devices which potentially could collide with the AGV should be stopped or slowed. Define a *workcell safety volume* as the volume enclosing the workcell which cannot be safely entered while the devices in the workcell continue normal operation. Note that it may be possible to safely enter a workcell safety volume if the devices within a workcell are slowed down or their operations are changed. A *device safety volume* is similarly defined as the volume surrounding a device which cannot be safely entered while the device remains in normal operation.

Associated with the notion of these safety volumes, assume that there are two levels of collision detection, *workcell volume warning* and *device volume warning*. The former provides warning that with an obstacle's current trajectory[5] it may intersect a particular workcell's safety volume, or a group of workcells' safety volumes. This is early warning that the devices in the workcell may have to stop or otherwise alter their normal operation. Similarly, device volume warning provides warning that a particular device's safety volume, or a group of devices' safety volumes, may be violated. If a device's safety volume is violated, the device *must* take immediate actions to avoid a collision.

Define *collision detection* (CD) tasks as those tasks assigned to track obstacles and determine whether any safety volumes will be violated. These tasks must estimate the earliest violation of any device's or workcell's safety volume in terms of some parameter. In addition, since there may be many obstacles present in the environment, the CD tasks must determine, for each message received from the sensors, whether a current obstacle is one which it is already tracking, whether the current obstacle presents a threat to any of the devices or workcells the CD task is monitoring, or whether the obstacle is a new threat.

[4] However, we will address only those issues related to the IMRS.

[5] Actually, since we will be sampling at discrete times, we will probably include all possible trajectories between the current time and the next sample time. For example, we may assume the moving obstacle can instantaneously change direction, and will search for all intersections within a given radius around the obstacle.

We define *device stopping* (DS) tasks as those tasks which determine how a device (or group of devices) can safely stop and how long (in terms of some parameter) the device (or group of devices) require to stop. For example, if two robots are carrying a heavy panel it may not be safe to have each robot just stop as quickly as they can (individually). This uncoordinated action may cause them to drop the panel or even damage themselves. Instead, we may want them to stop as fast as possible while not deviating from their preplanned path (to avoid any further collisions).

An important issue is how many CD and DS tasks should there be, and what their relationships should be with the other tasks. One option would be for each workcell to have its own CD task, and if an obstacle comes within a prescribed minimum distance, the CD task would spawn subtasks for the individual devices within the workcell. However, the overhead associated with setting up new tasks may be prohibitive. Also, we may not know that there is enough computing power available to run each of these tasks in real-time. A better idea might be for these tasks to be preassigned to processors but remain "inactive" until required. As more processing by the CD tasks is required, the other tasks would be forced to slow down. Since we would probably want the devices to slow as an obstacle came near, this may not be much of a problem if the CD tasks and the device controlling tasks were assigned to the same processor.[6]

Another issue here is in dynamic priority assignments. As an obstacle comes near, we may want the CD tasks to have the highest priority. When a collision becomes imminent we want the task controlling the stoppage of a device to have the highest priority (and not be interruptible).

6 Summary

We are currently investigating various issues of system integration, the solutions of which will extend the framework of an IMRS to meet its real-time performance and fault-tolerance goals. While many of the issues presented are currently being studied in the literature, few solutions deal with the special requirements of the IMRS. The use of a communication coprocessor to speed up communications, provide real-time task scheduling, and maintain tables and lists for fault-tolerance has been discussed. Issues related to the task distribution in an IMRS have been addressed. The use of work coupled tasks to recover from failed tasks, as well as the use of heartbeat messages to determine failed processors has been examined. The solutions of these problems will benefit not only the IMRS, but also other distributed real-time systems.

[6]There may be a problem if, for example, robots were following a prescribed trajectory. In this case, we may not be able to follow the trajectory without sufficient computational power. We may be able to follow the same path, though.

References

[1] Maimon, O. Z., and Nof, S. Y., "Coordination of robots sharing assembly tasks", *Journal of Dynamic Systems, Measurement, and Control*, Vol. 107, December 1985.

[2] Maimon, O. Z., "A multi-robot control experimental system with random parts arrival", *IEEE Conference on Robotics and Automation*, St. Louis, MO, March 1985.

[3] Simpson, J. A., Hocken, R. J., and Albus, J. S., "The automated manufacturing research facility of the National Bureau of Standards", *Journal of Manufacturing Systems*, Vol. 1, No. 1, 1984, pp. 17-31.

[4] Jones, A. T., and McLean, C. R., "A proposed hierarchical control model for automated manufacturing systems", *Journal of Manufacturing Systems*, Vol. 5, No. 1, pp. 15-25.

[5] Haynes, L. S., Barbera, A. J., Albus, J. S., Fitzgerald, M. L., and McCain, H. G., "An application example of the NBS robot control system", *Robotics and Computer-Integrated Manufacturing*, Vol. 1, No. 1, 1984, pp. 81-95.

[6] Shin, K. G., Epstein, M. E., and Volz, R. A., "A module architecture for an integrated multi-robot system", *Technical Report*, RSD–TR–10–84, Robot Systems Division, Center for Research and Integrated Manufacturing (CRIM), The University of Michigan, Ann Arbor, MI, July 1984. Also appeared in the *Proc. 18th Hawaii Int'l Conf. on System Sciences*, January 1985, pp. 120-129.

[7] Shin, K. G., and Epstein, M. E., "Intertask communications in an integrated multi-robot system", *Technical Report*, RSD–TR–4–85, Robot Systems Division, Center for Research and Integrated Manufacturing (CRIM), The University of Michigan, Ann Arbor, MI, May 1985. Also appeared in *IEEE Journal on Robotics and Automation*, Vol. RA-3, No. 2, April 1987, pp. 90-100.

[8] Manufacturing Automation Protocol (MAP) Reference Specification (Draft), February 25, 1986.

[9] IEEE Standards Board. IEEE Standards for Local Area Networks: Token-Passing Bus Access Method and Physical Layer Specification. New York: IEEE, 1985.

[10] Stallings, W., "IEEE Project 802 : Setting standards for local-area networks", *ComputerWorld*, February 1984.

[11] "Manufacturing Message Specification – Part 1: Service Specification", *ISO 2nd DP 9506*, May 21, 1987.

[12] "Manufacturing Message Specification – Part 2: Protocol Specification", *ISO 2nd DP 9506*, May 21, 1987.

[13] Ramachandran, U., "Hardware support for interprocess communication", *Computer Sciences Technical Report # 667*, University of Wisconsin, Madison, WI., September 1986.

[14] Muralidhar, K. H., "Performance management – measures, analysis, control, and optimization", *Proc. 11-th Conference on Local Computer Networks*, October 1986, pp. 20-25.

[15] Chu, W. W., Holloway, L. J., Lan, M. T., and Efe, K., "Task allocation in distributed data processing", *Computer*, November 1980, pp. 57-69.

[16] Efe, K., "Heuristic models of task assignment scheduling in distributed systems", *Computer*, June 1982, pp. 50-56.

[17] Ma, P. Y. R., Lee, E. Y. S., and Tsuchiya, M., "A task allocation model for distributed computing systems", *IEEE Transactions on Computers*, Vol. C-31, No. 1, January 1982, pp. 41-47.

[18] Chu, W. W., and Lan, L. M. T., "Task allocation and precedence relations for distributed real-time systems", *IEEE Transactions on Computers*, Vol. C-36, No. 6, June 1987, pp. 667-679.

[19] Lo, V. M., "Heuristic algorithm for task assignment in distributed systems", *Proc. 4-th International Conference on Distributed Computing Systems*, May 1984, pp. 30-39.

[20] Knight, J. C., and Urquhart, J. I. A., "On the implementation and use of Ada on fault-tolerant distributed systems", *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 5, May 1987, pp. 553-563.

[21] Birman, K. P., Joseph, T. A., Raeuchle, T., and Abbadi, A. E., "Implementing Fault-Tolerant Distributed Objects", *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 6, June 1985, pp. 502-508.

[22] Kramer, J. and Magee, J., "Dynamic Configuration for Distributed Systems", *IEEE Transactions on Software Engineering*, Vol. SE-11, No. 4, April 1985, pp. 424-435.

DEVELOPMENT OF MOIRE MACHINE VISION

Kevin G. Harding
Industrial Technology Institute
P.O. Box 1485
Ann Arbor, Michigan 48106

ABSTRACT

Three-dimensional perception will be essential to the development of versatile robotics systems capable of handling complicated manufacturing tasks in the factory of the future and in providing the high accuracy measurements needed in flexible manufacturing and quality control. The program described here will develop the potential of moire techniques to provide this greatly needed capability in vision systems and automated measurements, and demonstrate Artificial Intelligence (AI) techniques to take advantage of the unique strengths of moire sensing. Moire techniques provide a means of optically manipulating the complex visual data in a three-dimensional scene into a form which can be easily and quickly analyzed by computers. This type of optical data manipulation will provide higher productivity through integrated automation, producing a higher quality product while reducing computer and mechanical manipulation requirements and thereby the cost and time of production. The objective of this program will be to develop this nondestructive evaluation technique such as to be capable of full-field range measurement and 3-dimensional scene analysis.

BACKGROUND

One of the most powerful senses available to humans is vision. Vision allows us to collect and analyze vast amounts of empirical data at an astounding rate. Many years of research have been devoted to developing sophisticated image processing systems for medical and military applications. Modern computing capabilities have greatly aided these systems by handling the vast amounts of information at faster and faster speeds. These systems often use large computing systems, yet still often require highly skilled operators. For a vision system to be practical for industrial application it must be versatile, fast (less than 0.1 seconds typically), inexpensive, and it must require a minimal amount of human operator support[1,2]. Given these requirements and the current limits of small computer systems, the amount of data which can be processed is limited.

There are currently three basic approaches to three-dimensional machine vision; range finding including structured lighting, stereo or binocular vision, and gray scale methods[3-16]. There are varying degrees to the range finding approach. A simple version is to focus a beam of light on the object at a given distance. As the object surface moves closer or more distant, the spot on the object surface will enlarge, the size of the spot being directly proportional to the change in surface height. The most popular versions use the triangulation method where a beam of light is projected onto the object's surface at some angle and the image of this spot or line of light is viewed at some other angle. As the object distance changes a spot of light on the surface will move along the surface by (change in spot position) = (change in distance) x (tan(incident angle) + tan(viewing angle)). If a line is projected onto the surface by imaging or by scanning a beam, the line will deform as it moves across a contoured surface as each point of the line move as described above[5-9]. Other range finding systems often use selected point measurements to supplement some other system[10]. Other systems use multiple lines or patterns such as reticles to cover more area at a time[4,11,12].

Stereo or binocular vision methods work on the same principle as human vision by obtaining parallax information by viewing the object from two different perspectives (as our two eyes do)[4,13-15]. The two views must be matched up and then the difference in the position of a correlated feature gives the depth information from the geometry of the two view points (similar to the triangulation methods). These methods are full-field, thereby keeping the data acquisition time to a minimum, and they do provide all the 2-D feature information (in fact they depend on these surface features). There has been some success using special purpose correlation hardware in a method similar to those developed for military surveillance industrial. Generally the software manipulation is necessarily intensive[4,15].

Shape form shading methods work on the principle that a uniform field of light incident on a uniformly reflecting surface will vary in intensity depending on its angle of incidence. That is, as the angle of incidence increases, a given portion of the light field

will be spread over a wider area of the surface, thereby decreasing how brightly that area is illuminated[4,16]. This system does not provide an absolute measurement relative to the machine coordinates (the other two approaches do provide this information) since it is insensitive to step changes. An auxiliary system such as a rangefinder would need to be added.

Precision full-field part measurement is an even more complicated task. Manual point-by-point measurements are often long and laborious. Without full-field object data, some imperfections can be missed altogether. There are some coordinate measuring machines on the market which can be programmed to measure one or a few specific parts[17]. This approach saves human labor and human error but does not necessarily work any faster. Therefore, measurements for quality control are often limited to a spot check system. A versatile automated contouring system capable of measuring either large or even small areas at a time would provide the opportunity for better and more complete inspections. For many applications, speed is a very important factor as well. An approach which simplifies the required computing capabilities while providing added data processing speed is to use some form of optical preprocessing of the visual image. Optical data manipulation and encoding acts on an entire visual scene simultaneously in a parallel fashion. Parallel processing of this type is very fast and can be used to arrange or sort information into a form which can more easily and more quickly be analyzed with a computer than by direct digitizing techniques. Unwanted information can be disposed of and the desired information can be highlighted, sorted, or encoded for easier manipulation. In particular, moire techniques can transform the out-of-plane shape information of an object into a two-dimensional intensity pattern in such a way as to separate the third dimension completely from the other two without losing information.

TECHNICAL DISCUSSION

Two main technologies support this program: moire interferometry for optical processing and data acquisition, and iterative model-driven constraint directed reasoning for scene analysis.

Moire interferometry is a full-field, noncontact measurement technique[18,19]. A moire pattern is made by forming a subject grating, by projecting, shadowing, or contacting a grating onto the object to be measured, and comparing this grating to some reference grating by overlaying the two grating images. If the reference grating is a straight line grating, the beat pattern between the two gratings will form a contour map of the object's surface in the same way that a topographic map delineates the contours of the land. A diagram of a simple projection moire system is shown in Figure 1. In this case, the grating is imaged onto the surface, then the surface is imaged back to a reference plane, from an angle different from the illumination angle, and the

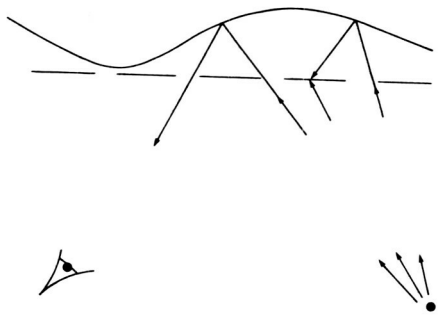


Figure 1. Diagram of moire setup.

resulting image is viewed through the reference grating. As an illustration of the pattern produced, Figure 2 shows a contour moire of a turbine blade. For this example a grating was placed in front of the statue and simply shadowed onto the statue, then the shadow was viewed through the original grating. This shadow moire effect can often be seen in everyday situations such as the patterns seen when an object is viewed through a screen door or through sheer woven draperies.

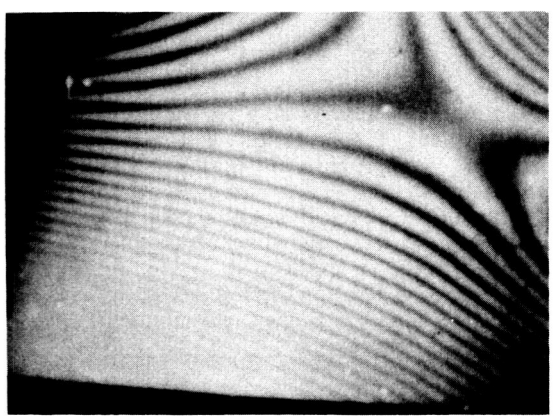


Figure 2. Moire pattern on a turbine blade.

If the reference grating is made by recording the image of the object grating, then the moire pattern can be used to show only differences between the reference object or reference state of an object and some new object or state. In this way, moire can be used to show only deviations from a good part. When applied to an on-line inspection system this difference moire approach greatly reduces the amount of information to be analyzed to determine if a part is within tolerances or to simply identify the part.

The sensitivity of moire contouring is given by the same relationship as other triangulation methods, ie.

$$Z = p/(\tan(i) + \tan(v))$$

where p = the grating period on the object
i = the angle of incidence of the projected pattern
v = the viewing angle
Z = the sensitivity per fringe

(the projected beam has been assumed to be approximately collimated, though this is not necessary in practice)

Typically, with digitizing methods a fringe can be measured to one tenth of a fringe. With such methods a sensitivity of 0.001 inches over a full field of a few square feet is not unreasonable with higher sensitivity over small areas. For example, over a one square foot area, the sensitivity could exceed 0.0005 inches, and over one square inch changes as small as 10 microinches have been recorded using moire techniques[18]. The particular sensitivity that would be practical would depend on many factors such as part geometry, system resolution, etc.. The sensitivity of the moire pattern can be tailored to fit the requirement by simply adjusting the grating period.

Since moire is full-field, the contour of an entire area of an object can be mapped out at one time and recorded during a single video frame. This allows anomalies as well as large-scale shape features to be viewed and measured to the same precision and at the same point in time (important in situations where thermal drifts or other factors may be warping the part with time). The instantaneous contour plot can be viewed immediately with high-accuracy numerical results available after a reasonable amount of analysis time. Such real-time viewing of a component is a common practice in fabricating high-quality optical components to tolerances of a few microinches. Many facilities and interferometer manufacturers have incorporated computer systems to digitize and analyze fringe data much like those from moire contouring[18,20-23]. As an example of such digitizing methods, Figure 3 shows a high sensitivity moire pattern (about 0.04 inches per fringe) of a machined part with bevelled surfaces. Figure 4 shows the computer generated isometric plot of the surface shape. The actual value of each point on the disk was then available to the computer to the spacial resolution of the computer model. In this case the depth information was available to about 0.001 inches.

As with the line projection techniques, the depth information is effectively encoded by the moire interferometer into a 2-dimensional map that is both easy for the computer system to record, since it is only from one perspective, and is independent of 2-D features on the surface of the object. This ability makes the 2-D features separable from the depth information which means the 2-D outlines can be analyzed separately using well established vision algorithms.

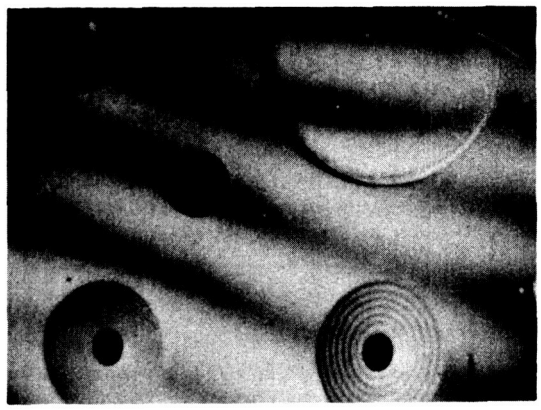


Figure 3. Moire patern on a machined part.

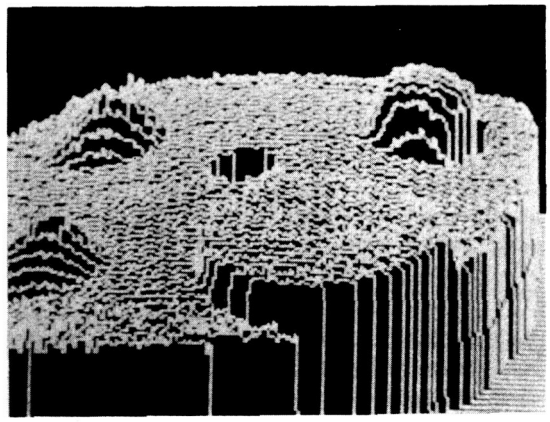


Figure 4. Shape information from moire.

Optically specular surfaces can and are routinely inspected using moire methods, though the method used for contouring specular surfaces is typically different than that employed for diffuse surfaces. It is typically easier to treat diffuse surface and specular surface contouring differently, but the results and the analysis are largely the same.

Peaks and valleys can easily be distinguished and absolute measurements made by a variety of methods. The sign of the slope being measured can be determined by a wide range of methods such as moving one grating with respect to the other (the fringes will move in different directions depending on whether the slope is positive or negative) or by moving the object between two consecutive recordings or by using mismatched gratings. These methods are very familiar to people working in moire interferometry and are well documented in the literature[18,24-31]. Absolute reference points in space can be established by encoding the grating in much the same way that some structured lighting techniques employ.

Scene analysis technology reasons from the 2-D and 3-D information returned by the moire system to identify and locate physical objects in the field of view [39]. Scene analysis is less complex if the system takes advantage of the limited population of potential objects to do model-driven recognition. This approach requires the system to store a model of each object that it may be required to recognize. Industrial applications deal with well-defined mass-produced parts with consistent characteristics, and so lend themselves well to a model-based approach [36].

An object can be represented with a semantic net formalism modeled on ACRONYM's prediction graph [33]. An object is represented as a set of *features*, such as edges and faces, together with *relationships* between those features.

Each frame in the net corresponds to a *feature*, with slots for the characteristics of the feature. A frame that corresponds to a line records the length of the line. A frame that corresponds to a face records such information as its area, perimeter, first moment of area, centroid, minimum and maximum diameters, and dimensions of smallest enclosing rectangle. In some cases, entire components of an object may be usefully modeled as a frame recording their extended gaussian images [40]. The range and gradient data provided by the moire sensor lends itself readily to extracting region-based analysis useful for characterizing faces of an object in this way. Region-growing on the basis of intensity information is difficult, since shadows or other secondary interference can produce spurious differences. Data showing constant gradient and continuous range is a much more robust indicator of a continuous plane face. Furthermore, the range and gradient data allows us to normalize the slot values recorded for a face so that they are invariant with respect to rotation or tilting of the part, within reasonable limits. Thus the use of a range and gradient image permits the construction of a feature-based representation of the object that is much more detailed and discriminatory than would be feasible from monocular intensity data alone.

Often the characteristics of a single feature will not be enough to diagnose the presence of a part, and the *relationships* among features must be taken into account. These relationships are represented as arcs in the semantic net, and include

- adjacency between features (lines to lines; faces to faces; lines bounding faces);
- the angles between adjacent lines or between the gradients of adjacent faces;
- mutual exclusion between features that cannot be simultaneously visible (such as two ends of a brick);
- Euclidean distance in 3-space between the centroids of two features.

The range and gradient data from the moire sensor permits efficient computation of the angular and normalized distance relations between features, data that otherwise would be very inefficient to obtain.

The network has nodes not only for features, but also for *objects*. Each feature that belongs to an object is joined to it with a "part-of" arc. The object node provides a place to store information such as gripping points for the object, indexed by the features that are visible from each orientation.

Our strategy is *iterative*, seeking to identify only one or a few objects in the scene, removing it, and then acquiring and analyzing a new scene. Some scene-analysis strategies seek to identify every item in the scene from a single image [37, 32]. The combinatorial complexity of matching features to models can become very high in this case. Furthermore, once a robot begins to manipulate parts in the bin, it may inadvertently move other parts, so that a repeated analysis may well be necessary anyway. Because moire gives range data, we can immediately identify the topmost features in the field of view and focus our attention on the objects to which they belong. Once the robot removes these objects, we take a new view of the bin and repeat the process. Since moire yields a full-field range image with a single exposure and very little computation, acquiring repeated images is cheap, and the combinatorial simplification obtained by identifying only a few objects at a time reduces the cost of analyzing multiple images.

One disadvantage of an iterative approach is that the robot cannot pick up parts in the order required by a sequential assembly plan, but must take them "as they come." There are solutions to this problem. The robot may buffer the parts for later reordering. Furthermore, if the robot is scheduled opportunistically rather than sequentially [34, 35], the probability increases that it will be able to make use of what it finds on top of the bin.

As a *model-driven* program, the system recognizes objects by comparing features visible in the frame with the prediction graphs stored for expected parts. The system does not aimlessly analyze every region visible in its field, but begins with the highest pixels (those nearest the sensor) to define the "topmost" features. As these are matched with features from the prediction graphs, graphs that are candidates for matching suggest additional features with known relations to the ones already observed that the system searches for to confirm or disprove hypothetical identifications. The use of expectations to drive the analysis of image data can significantly reduce the amount of visual processing necessary to reach an identification.

The system identifies regions with models by *constraint propagation*, along the lines used by [38]. Any single region may match several different models. The system maintains with each region a list of the objects and their features that the region matches.

As more and more regions are matched, the system compares the relationships between adjacent regions with the relationships specified in the prediction graphs for the candidate objects, and refines its estimates of the objects to which the features belong on the basis of those relationships.

For example, assume that two regions are adjacent, and both match features from two objects, A and B. If the relationship between the regions matches the relationship between the corresponding features in A but not in B, the system refines the identification of the regions by eliminating B from the list of candidate objects.

APPLICATIONS

The distinct characteristics of moire machine vision make it particularly suitable to many industrial applications where other systems would have problems. Unlike systems which depend on surface marks or shading effects, moire contouring is very amenable to dirty or hostile environments. Such environments often exist in automated manufacturing. Moire contouring could also be applied in situations where the part may be very hot or vibrating where neither physical contact nor long scanning measurement times are practical. An example of this would be a metal extruding operation or an operation in which the parts have been freshly painted or lacquered. A moire vision system would permit such operations to be better controlled by providing inspection during otherwise inactive waiting times in the manufacturing process. This would reduce waste and increase productivity, while removing human inspectors from a potentially hazardous environment.

The noncontact measurement capability of the moire machine vision system would have great value in many precision measurement applications. As a specific example of the need for a full-field optical measurement system, NASA has encountered a problem in measuring aircraft models for wind tunnel tests. The complexity of these models requires that they be completely measured to an accuracy of about 0.001 inches. To measure the models point by point manually requires two to three days at considerable expense. If any small anomalies are missed, the model may not react as predicted in the wind tunnel or may fail completely. In addition, because of the delicate nature of the surfaces of the models, physical contact may actually damage the model. The moire vision system could measure the entire model in minutes and could even be used as a diagnostic tool during a wind tunnel test to measure the model deflections (a task impossible for a coordinate measuring machine). NASA has been very interested in such technology.

Another specific example of great current interest in the aerospace industry is the inspection of turbine and compressor blades for turbine engines. In this case, there are a number of different inspection requirements. For example, on the leading edge and on a region from 0.03 to 0.1 inches back from the leading edge center the tolerance is typically on the order of 0.0005 inches. On the concave and convex faces of the airfoils, which typically have a cord width of 2 inches or less, the tolerances are typically in the range of 0.002 inches (for some finished turbine blades) to 0.004 inches. These inspections are often performed using mechanical reference slides by measuring the gap between the airfoil and the slide with feeler gages. This process is laborious and highly dependent on the accuracy of the reference slide, the feeler gage and positioning. This is typically purely a Go/No-Go type of test.

With the capability to easily vary the sensitivity of moire interferometry, and the high sensitivity of moire methods over small areas, a system could be developed which would zoom in on the leading edge area with high sensitivity and yet be versatile enough to provide only the 0.002 inch tolerance measurement needed over areas of 2 inches square or more.

Finally, since the moire data is full-field, and available (with the encoding) in one video frame, the data can actually be taken in microseconds with a strobed camera or lighting. This allows freezing of dynamic objects. As the data is built up with scanning in the standard line of light structured light sensors, the time to obtain the data is 0.1 to 0.3 seconds, thus precluding the freezing of dynamic objects.

In the area of robot guidance, a moire assisted vision system offers a distinct advantage in both speed and size over current three-dimensional vision systems or range finder gages. The optical preprocessing would make real-time three-dimensional information available for locating identified parts in a pile (which part is on top of the other) or for distinguishing parts with the same two-dimensional shape but different thickness. Since the third dimension is obtained through the vision system, the sensor can be a compact, light-weight, solid-state camera such as has already been applied to robot guidance and have even been mounted directly on the robot arm in some applications.

SUMMARY

Moire contouring can provide high resolution depth information to a vision system by encoding the information into an easily analyzed 2-D pattern. The depth and surface feature information can be easily separated for simplified analysis. Absolute contouring information regarding the sign of a slope and the relative position of the subject to some reference surface is available with this technique. Full-field data measuring all points on the object, not just where there is a reference point, with variable sensitivity and insensitivity to dirt, stray light or vibrations is available with moire contouring. A moire vision system would solve many of the problems now encountered by 3-D vision systems using

technology which has been well developed for other applications. The application of scene analysis techniques similar to those used for 2-D images, can provide a means of part recognition, and three-dimensional locating tasks required for true flexibility of robotic operations.

REFERENCES

1. A. E. Thomas and K. I. Staut, "Robot Vision," Engineering, May 1980, p. 533.

2. P. Marsh, "Robots See the Light," New Scientist, 12 June 1980, p. 238.

3. Q. Kinnuean, "How Smart Robots are Becoming Smarter," High Technology, Sept/Oct, 1983, p. 32.

4. E. L. Hall and C. A. McPherson, "Three Dimensional Perception for Robot Vision," SPIE Proc. Vol. 442, 1983, p. 117.

5. M. R. Ward, D. P. Rheaume, S. W. Holland, "Production Plant CONSIGHT Installations," SPIE Proc. Vol. 360, 1982, p. 297.

6. G. J. Agin and P. T. Highnam, "Movable Light-Stripe Sensor for Obtaining Three-Dimensional Coordinate Measurements," SPIE Proc. Vol. 360, 1983, p. 326.

7. K. Melchior, U. Ahrens, M. Rueff, "Sensors and Flexible Production," SPIE Proc. Vol. 449, 1983, p. 127.

8. C. G. Morgan, J. S. E. Bromley, P. G. Davey, and A. R. Vidler, "Visual Guidance Techniques for Robot Arc-Welding," SPIE Proc. Vol. 449, 1983, p. 390.

9. G. L. Oomen and W. J. P. A. Verbeck, "A Real-Time Optical Profile Sensor for Robot Arc Welding," SPIE Proc. Vol. 449, 1983, p. 62.

10. J. E. Orrock, J. H. Garfunkel, and B. A. Owen, "An Integrated Vision/Range Sensor," SPIE Proc. Vol. 449, 1983, p. 419.

11. H. K. Nishihara, "Prism: A Practical Real-Time Image Stereo Matcher," SPIE Proc. Vol. 449, 1983, p. 134.

12. M. C. Chiang, J. B. K. Tio, and E. L. Hall, "Robot Vision Using a Projection Method," SPIE Proc. Vol. 449, 1983, p. 74.

13. J. Y. S. Luh and J. A. Klaasen, "A Real-Time 3-D Multi-Camera Vision System," SPIE Proc. Vol. 449, 1983, p. 400.

14. G. Hobrough and T. Hobrough, "Stereopsis for Robots by Iterative Stereo Image Matching," SPIE Proc. Vol. 449, 1983, p. 94.

15. N. Kerkeni, M. Leroi, and M. Bourton, "Image Analysis and Three-Dimensional Object Recognition," SPIE Proc. Vol. 449, 1983, p. 426.

16. C. A. McPhenson, "Three-Dimensional Robot Vision," SPIE Proc. Vol. 449, 1983, p. 116.

17. G. L. Franck and J. L. Henry, "Flexible In-Line Inspection for the Automated Factory," Proceedings 2nd Biennial International Machine Tool Conference, Sept. 1984, p. 7-43.

18. K. Harding, "Moire Interferometry for Industrial Inspection," Lasers and Applications, Nov. 1983, p. 73.

19. K. Harding and J. S. Harris, "Projection Moire Interferometer for Vibration Analysis," Applied Optics, Vol. 22, No. 6, 1983, p. 856.

20. A. T. Glassman, "Automated Interferogram Reduction," presented at SPIE meeting in Washington D. C. April 19, 1979.

21. H. E. Cline, A. S. Holik, and W. E. Lorensen, "Computer-Aided Surface Reconstruction of Interference Contours," Applied Optics, Vol. 21, No. 24, 1982, p. 4481.

22. W. W. Macy Jr., "Two-Dimensional Fringe-Pattern Analysis," Applied Optics, Vol. 22, No. 22, 1983, p. 3898.

23. L. Mertz, "Real-Time Fringe Pattern Analysis," Applied Optics, Vol. 22, No. 10, 1983, p. 1535.

24. F. P. Chaing, "Determination of Signs in Moire Method," J. Engineering Mechanics Division, Proceedings of the American Society of Civil Engineers, EM6, Dec. 1969, p. 1379.

25. M. Idesawa, T. Yatagai, and T. Soma, "Scanning Moire Method and Automatic Measurement of 3-D Shapes," Applied Optics, Vol. 16, No. 8, 1977, p. 2152.

26. G. Indebetouw, "Profile Measurement Using Projection of Running Fringes," Applied Optics, Vol. 17, No. 18, 1978, p. 2930.

27. D. T. Moore and B. E. Truax, "Phase-Locked Moire Fringe Analysis for Automated Contouring of Diffuse Surfaces," Applied Optics, Vol. 18, No. 1, 1979, p. 91.

28. R. N. Shagam, "Heterodyne Interferometric Method for Profiling Recorded Moire Interferograms," Optical Engineering, Vol. 19, No. 6, 1980, p. 806.

29. M. Halioua, R. S. Krishnamurthy, H. Liu, and F. P. Chiang, "Projection Moire With Moving Gratings for Automated 3-D Topography," Applied Optics, Vol. 22, No. 6, 1983, p. 850.

30. K. J. Gasvik, "Moire Technique by Means of Digital Image Processing," Applied Optics, Vol. 22, No. 23, 1983, p. 3543.

31. H. E. Cline, W. E. Lorensen, and A. S. Holik, "Automatic Moire Contouring," Applied Optics, Vol. 23, No. 10, 1984, p. 1454.

32. Boyter, B.A., and J.K. Aggarwal, "Recognition of Polyhedra from Range Data," IEEE Expert, 1, 1986, p.47-59.

33. Brooks, R.A., "Symbolic Reasoning among 3-d objects and 2-d models," AI Journal, 16, 1981, p.285-348.

34. Fox, B.R., and K.G. Kempf, "Opportunistic Scheduling for Robotic Assembly," Proceedings of the IEEE International Conference on Automation and Robotics, St. Louis, 1985, p.880-889.

35. Fox, B.R., and K.G. Kempf, "Complexity, Uncertainty, and Opportunistic Scheduling," Proceedings of the Second IEEE Conference on Artificial Intelligence Applications, Miami, 1985, p.487-492.

36. A.C. Kak, K.L. Boyer, C.H. Chen, R.J. Safranek, and H.S. Yang, "A Knowledge-Based Robotic Assembly Cell," IEEE Expert, 1, 1986, p.63-83.

37. Srihari, S.N.; J.K. Udupa, and M. Yau, "Understanding the Bin of Parts," Proceedings of the IEEE International Conference on Cybernetics and Society, Denver, 1979, p.44-49.

38. Waltz, D., "Generating Semantic Descriptions from Drawings of Scenes with Shadows," In P. Winston, ed., The Psychology of Computer Vision, New York: McGraw Hill, 1975, p.19-92.

39. Besl, P.J., and R.C. Jain, "Three-Dimensional Object Recognition," Computing Surveys, 17, 1, 1985 p.75-145.

40. Horn, B.K.P., "Extended Gaussian Images," Proceedings of the IEEE, 72, 12 December, 1984, p. 1671-1686.

N88-17265

IMPLEMENTATION OF A ROBOTIC FLEXIBLE ASSEMBLY SYSTEM

Ronald C. Benton
Honeywell Corporate Systems Development Division
1000 Boone Avenue North
Golden Valley, MN 55427

ABSTRACT

As part of the Intelligent Task Automation (ITA) program sponsored by the Air Force and the Defense Advanced Research Projects Agency (DARPA), a team of industry, research institute and university members developed enabling technologies for programmable, sensory-controlled manipulation in unstructured environments. These technologies included 2-D/3-D vision sensing and understanding, force sensing and high-speed force control, 2.5-D vision alignment and control, and multiple-processor architectures. This paper describes the subsequent design of a flexible, programmable, sensor-controlled robotic assembly system for small electromechanical devices using these technologies and ongoing implementation and integration efforts. Using vision, the system will acquire parts dumped randomly in a tray. Using vision and force control, it will perform high-speed part mating, in-process monitoring/verification of expected results and autonomous recovery from some errors. It will be programmed off-line with semiautomatic action planning.

INTRODUCTION

The Intelligent Task Automation program objectives are to develop and demonstrate generic technologies that (1) form a basis for advanced intelligent automation systems, (2) have near-term application to unit processes, component assembly, and many aspects of defense batch manufacturing, and (3) establish the technology base for new opportunities to apply intelligent systems to complex military tasks [1, 2, 3]. This program identifies and develops robot-related technologies that should have a high payoff for future manufacturing systems.

Phase I included research, development and feasibility demonstrations that culminated in a conceptual design for each component as well as the system itself. Breadboard hardware and developmental software were constructed to critically address areas of uncertain technical risk and to show feasibility of the entire system. Phase I concluded in December 1985 with the results documented in References 4, 5 and 6.

Phase II of the program began 1 September 1986 and will culminate in mid-1988 with demonstration of an entire system. The functional elements of the Intelligent Task Automation System (ITAS) are shown in Figure 1 along with the enabling technologies developed during Phase I, providing the basis for system design. The ITAS demonstration configuration is shown in Figure 2. The ITAS is a flexible, programmable, sensor-controlled robotic assembly system for small electromechanical parts. It operates in an unstructured environment relative to today's technology since it recognizes, locates and grasps the proper parts (which may be touching or overlapping) to be assembled from a tray. Once parts are grasped, the ITAS performs high-speed mating and fastening actions to assemble the device. During assembly, it performs in-process monitoring and verification of expected results and autonomous recovery from some anticipated errors. The ITAS is programmed off-line with semiautomatic action planning enhancements.

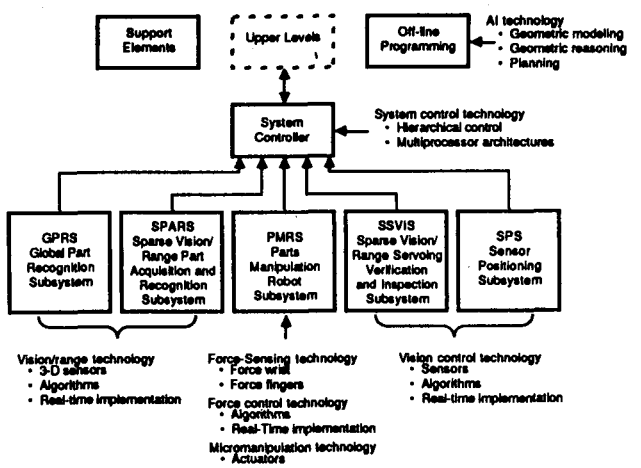


Figure 1. ITAS Functional Elements and Associated Enabling Technologies

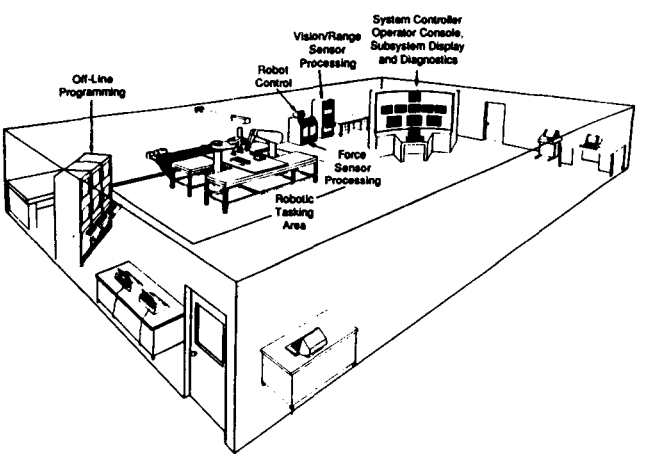


Figure 2. ITA System

To demonstrate the primary assembly task, automatic assembly of a Honeywell 1EN1-6 microswitch will be performed (see Figure 3). This assembly was selected due to the variety of part geometries, materials and mating requirements. A different electromechanical device will also be assembled to illustrate the flexibility of the ITAS.

Figure 3. Tray of 1EN1 Microswitch Parts as Seen by Global Part Recognition Sensor

DEVELOPMENT APPROACH

The ITA Phase II System development cycle follows the classic systems engineering approach of requirements formulation, design, implementation, integration and test. Since most of the effort involves software, the structured analysis/structured design method was selected [7]. The graphics-based SA/SD process consists of developing both a system data/control flow model (by analyzing the requirements of the system) and a system structure model that is a translation of those requirements into a description of the system architecture, system components, interfaces, data and materials necessary for the implementation phase, i.e., a design. The results are documented in so-called "structured" specifications.

A computer-aided software engineering tool set from Interactive Development Environments Inc. (IDE) [8] running on UNIX-based Sun Microsystems work stations was applied to support the SA/SD methodology. Using graphics input, the tool generates a system data base to enable automatic decomposition, consistency checking, dictionary generation, and on-line specification access. To permit specification generation, the IDE tool was linked to a documentation system to automatically extract textual descriptions attached to graphic diagrams and to insert that text in proper format in the specification documents.

This "automatic" documentation tool set was developed to simplify creation of documents for the ITA System; i.e., specifications, manuals, etc. The documents can be large, can be constructed from many parts in complex ways (e.g., graphics and text from the SA/SD tools must be integrated into design specifications), must be controlled (configuration management), and must follow strict style guidelines. Since the SA/SD tools reside on UNIX-based Sun Microsystems work stations and most of the code will be generated there, the documentation tools were built using UNIX tools. Document content and style guidelines are supported by on-line templates for each document type, a set of *nroff/troff* formatting macros and some preprocessing of the documents. Since the templates are ASCII files, both designers and secretaries fill them in using terminals and text editors of their choice, and the documentation system generates a document in the proper format. Finally, to aid in control of the document as a whole, procedures for revision control were developed using the UNIX *SCCS* (Source Code Control System).

Ideally, it would be possible to perform a complete top-down, implementation-independent requirements analysis before considering design candidates. However, in many cases, as in ITA, preliminary and/or partially documented implementations exist that could, at least in part, be used in the new design. Using the structured design tools described above, reverse engineering of these implementations is being performed in parallel to support design activities. Specifically, the objectives of this process are to: (1) generate top-level maps of major modules and calling-tree structures; (2) identify "black box" modules, i.e., code with sufficient maturity and documentation for reuse as is; (3) map calling structures of high risk and/or critical code; (4) identify potential trouble spots; (5) provide input to design decisions on candidate selection, reusability, and high-payoff rework areas.

The process is based on generation of a "battle map" using annotated structure charts. Each module is rated on the level of existing documentation, a subjective "modifiability" index (i.e., how easy it would be to modify the code, if necessary) and the extent of global variable usage. The battle map enables quick identification of both black box modules and trouble spots.

The CIMStation work cell simulation tool from Silma Inc. was installed on an Apollo work station to support system analysis, design, incremental integration and off-line programming throughout Phase II [9]. CIMStation provides a geometric modeler with IGES interface, dynamic robot simulations, multiple process simulation with synchronized signal/wait interprocess communication, and a LISP-based, Pascal-like, device-independent simulation language with device-dependent translators. A rapid prototype of the ITA System design was developed to validate the subsystem functional allocations, verify robot selections, improve the system time-line estimates and identify task-level programming approaches.

Finally, rapid prototyping of critical hardware/software elements, such as the force-controlled robot and vision sensors, is being performed where necessary to support the design process.

SYSTEM REQUIREMENTS AND DESIGN

The ITA System will implement generic technologies developed in vision/range sensing and understanding, force sensing and understanding, sensory-based control and manipulation, and off-line programming (see Figure 1). A primary assembly task (i.e., the 1EN1 microswitch) will be performed to demonstrate its capability, and secondary assembly tasks will be performed to demonstrate its flexibility. The autonomous performance of these tasks along with a demonstration of the ITAS reprogrammability must clearly distinguish between the ITA System's method of performing a task and that of hard automation.

The capability of the ITA System to become part of a larger system must also be demonstrated. In a factory of the future, the ITAS would be one of many work cells at the lowest levels of the hierarchy, such as in a small batch assembly plant. Here, an automated storage/retrieval work cell would generate the kits of parts for each device to be assembled. The ITAS would assemble these kits and send them to other work cells for subsequent assembly operations, final inspection and packaging (or, if necessary, rework). These cells would be interconnected with an intercell materials handling system such as conveyors and/or robot carts. Each group of work cells would be coordinated by the supervisor layer of the factory hierarchy to fill customer orders for small batches of devices.

These cells would be tended by shop floor operators who would support changeover from one type of assembly to another and

correct noncatastrophic equipment failures. Routine maintenance, catastrophic failures, and test/debug of first-time assembly tasks would be handled by manufacturing technicians/engineers. To maximize utilization of shop floor equipment, however, work cell programming would be performed off-line by CAD/CAM specialists with access to computer-based information about the new device to be manufactured and the capabilities of the shop floor equipment. Thus, work cell software and, if necessary, new tools/fixtures would be developed without taking shop floor equipment out of production.

Since the ITAS has three operating modes–on-line (i.e., the shop floor segment of the ITAS is producing assemblies), downtime (all shop floor activities when the ITAS is not on-line), and off-line programming (which occurs independently of the other two modes)–the external performance requirements for each mode were defined. In on-line mode, the system will perform assembly of the 1EN1-6 microswitch and other secondary assembly tasks in an autonomous manner with speed and reliability comparable to a human. It will: (1) recognize, locate and grasp the proper parts to be assembled, which are presented randomly in a tray and may be touching or overlapping; (2) perform high-speed mating and fastening actions to combine these parts into a subassembly using simple and/or multifunction fixtures; (3) provide in-process monitoring and verification of expected results during the assembly; (4) provide annunciation, recording and error recovery from expected (i.e., previously defined as potential) error conditions; (5) provide annunciation and recording of all unexpected results.

Downtime activities include ITAS start-up, shutdown, routine maintenance, changeover and test/debug of new tasks. Thus, the duration and/or frequency of this mode should be minimized. The off-line programming mode will occur concurrently with the other two modes to provide maximum productivity in the shop floor segment of the ITAS. Thus, both the amount of programming time and expertise required off-line and the time spent in the downtime mode for test/debug (to get the ITAS to perform a new assembly task) should be minimized.

The resulting system requirements flow model (context diagram, not shown) clarifies the ITAS boundaries. Part kit trays are provided by an intercell materials handling system (a human operator will be used in the Phase II demonstration); similarly, trays of completed or aborted assemblies are removed by this system. The two classes of users that interact with the ITAS are shop floor users (operators and production engineers) and off-line programmers. Decomposing one level reveals the two major segments of the ITAS–the shop floor, Process 1, and the off-line programming, Process 2. These processes can run concurrently and could be located in different sites with appropriate communication links. Process 3 generates displays to illustrate nonobvious aspects of ITAS operation for Phase II demonstration purposes. The decomposition of Process 1 is shown in Figure 4. The detailed operational scenario presented later illustrates how this flow model represents the ITA System's functional requirements.

The structure of the ITAS (see Figure 1) resulted from allocating one or more processes in the requirements flow model to "black

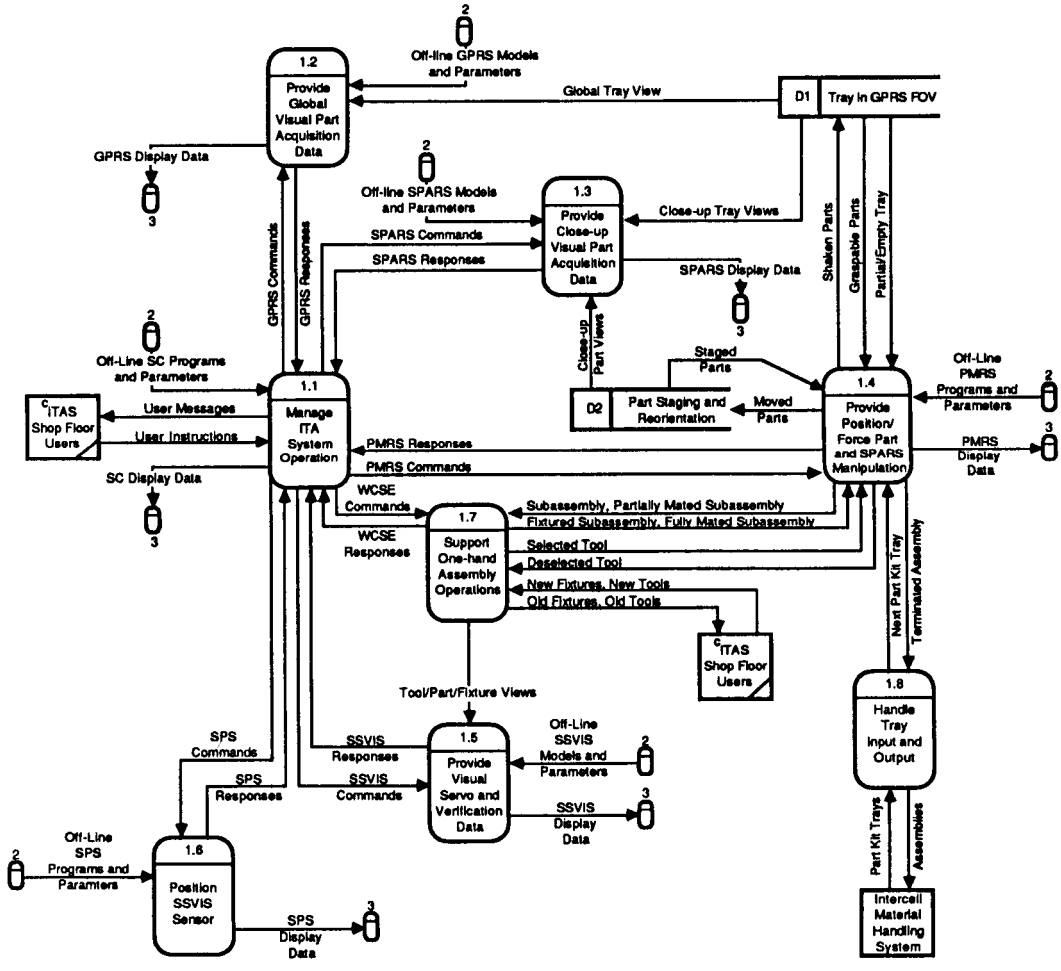


Figure 4. Provide Shop Floor Assembly Functions (Process 1.0)

box" system elements with well-defined interfaces. These system elements, or subsystems, interact to achieve the overall system requirements as shown by comparison to Figure 4:

- A System Controller (SC)--The controller will provide the upper-level control to coordinate three vision subsystems and two robots (Process 1.1).
- A Global Part Recognition Subsystem (GPRS)--This subsystem will contain the primary part recognition sensors to acquire and process gray-scale and dense vision/range images for part identification, location and acquisition strategy generation (Process 1.2).
- An Arm-Mounted Sparse Vision/Range Part Acquisition and Recognition Subsystem (SPARS)--This subsystem will provide the functions to assist GPRS during part recognition and to fine tune part location prior to acquisition (Process 1.3).
- A Part Manipulation Robot Subsystem (PMRS)--This subsystem with force sensing and control will provide the delicate parts manipulation function and will perform part acquisition, mating, and fastening operations along with contact verification of in-process conditions before and after part-mating operations (Process 1.4).
- An Arm-Mounted Sparse Vision/Range Servoing Verification and Inspection Subsystem (SSVIS)--This subsystem will measure relative part misalignment during mating operations and conduct noncontact inspection/verification of in-process conditions before and after part-mating operations (Process 1.5).
- A Sensor Positioning Subsystem (SPS)--This subsystem will position the SSVIS sensor (Process 1.6).
- An Off-Line Programming Subsystem (OLPS)--This subsystem will perform interactive system programming, program validation, program download and data base generation for the on-line segments of the system (Process 2).
- Work Cell Support Elements (WCSE)--This subsystem will provide the work table, the parts presentation means, the fixtures used during assembly and the structure to support the sensors (Process 1.7 and Process 1.8)

The major hardware components of the subsystems described are configured as shown in Figure 5.

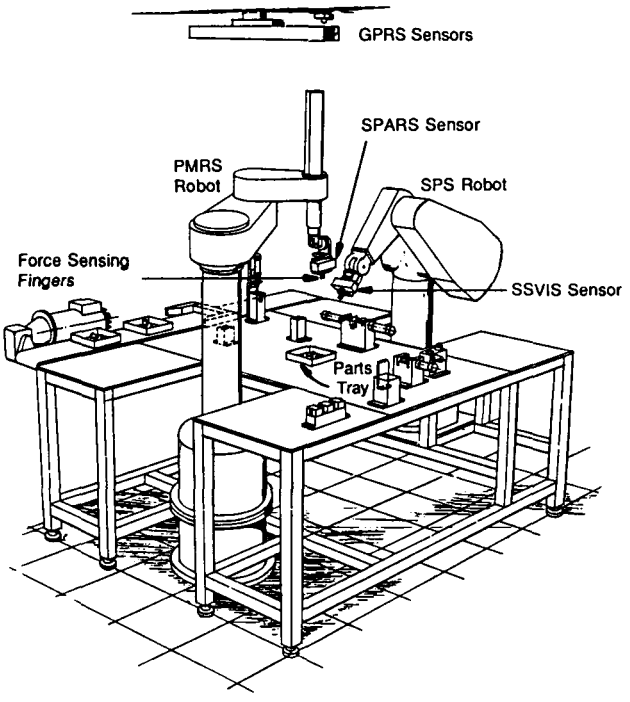


Figure 5. ITAS Robotic Tasking Area

OPERATIONAL SCENARIO

The ITAS requirements and design may be better understood (and verified) by walking through the intended operational scenario for each of the three operating modes.

On-Line Mode

Prior to entering this mode, all system programs and data bases for the desired assembly task (for example, the 1EN1-6 microswitch) will have already been generated as described later. In addition, they will have been tested and debugged on the shop floor segment of the ITAS and loaded along with any tools and fixtures.

The operator then invokes a command such as "ASSEMBLE n 1EN1-6," and the SC (Process 1.1) will begin sequencing the system to assemble the n microswitches. The PMRS robot (Process 1.4) will be directed to acquire from the conveyor (Process 1.8) the 5-inch by 5-inch trays (into which kits of the 17 microswitch parts have been dumped) and to position each tray beneath the GPRS sensor, where PMRS uses force sensing to seat the tray against registration stops (Data Store "Tray in GPRS FOV"). No parts presentation means other than the tray will be used. The SC directs GPRS (Process 1.2) to find the desired part or any of an ordered list of larger parts. GPRS then acquires gray-scale and dense-range (terrain map) images to recognize, locate and determine a grasp strategy (i.e., a PMRS finger selection and corresponding unobstructed grasp points) to acquire the parts in the order needed for assembly. The parts will be touching, overlapping and possibly occluding each other in the tray; thus, if GPRS fails to find the desired part (e.g., the first part needed for the microswitch, the bushing, is large, but the second part, the red O-ring, is small and frequently occluded), the first larger part found is acquired and placed in the staging area (Data Store "Part Staging and Reorientation"), and GPRS is directed again to find the desired part. If GPRS fails to find any of the parts in the ordered list, the SC directs PMRS to shake the tray and reposition it under GPRS. If GPRS still fails, the operator is notified.

Whenever PMRS is available for acquiring another part from the tray (i.e., not doing an assembly operation), the SC asks GPRS what it has found so far in its list of parts. GPRS replies but continues to work on its ordered list until every part is found, it gives up, or the SC sends GPRS a new list. Thus, if subsequent operations find that GPRS' reply is unacceptable, the SC can return to GPRS to see if a better answer has been found. For example, GPRS will tag some of the parts it has found for SPARS (Process 1.3) to verify the identity and/or pose of a given part at a given location in the tray. Then, while GPRS continues to work, SPARS will be positioned over the part by PMRS, where it will gather vision/range data and provide the appropriate feedback. If SPARS rejects GPRS' answer, the SC can go back to GPRS for another part(s) to acquire.

Thus, the PMRS robot will acquire the quick-change fingers (from Process 1.7) directed by GPRS to pick up the part, use its servoed gripper to open the fingers to an appropriate size, use a force-controlled guarded move to check for collisions while placing its

fingers around the part, close the gripper with a grip force appropriate for the part, verify that its finger opening matches the part dimensions, and remove the part from the tray. If any of the above part identity/grasp strategy verifications fails, the SC will return to GPRS for another part(s). PMRS may be directed to place the part in the staging area to regrasp it properly or flip it over for the upcoming assembly steps; SPARS may be required to verify/fine tune the part pose to enable PMRS to regrasp it. Once PMRS has cleared the parts tray, GPRS will be sent a new list of parts.

Meanwhile, the SPS robot (Process 1.6) will be posing the SSVIS sensor (Process 1.5) at the appropriate fixture/subassembly (Process 1.7) for the upcoming part assembly operation. PMRS will blindly bring the two parts to be assembled into the appropriate relative poses necessary for mating. SSVIS will visually measure the relative misalignment between these parts and provide feedback to PMRS (via the SC) to correct the misalignment. When the parts are aligned, SSVIS will verify that the in-process conditions are correct for the assembly to proceed (i.e., correct parts and tools are present). If preassembly conditions are acceptable, PMRS will mate the two parts using high-speed force servoing for compliant, delicate micromanipulation of the parts. Mating operations will terminate when position and/or force limits are reached; data from the termination will be interpreted for contact verification of in-process conditions. PMRS will extract its fingers from the subassembly and return to the tray to acquire the next part that GPRS has found while this assembly was occurring.

With PMRS out of the way, the SPS robot can position the SSVIS sensor wherever necessary (e.g., looking down the bushing bore) to verify post-assembly conditions (i.e., correct parts are present and mated correctly). If expected errors exist, the SC provides error correction based upon scripts generated during previous testing.

Each additional part is assembled into a microswitch subassembly in the above fashion. If an in-process error is detected where the SC has a predetermined error recovery script, the system will attempt to correct it; otherwise, the SC will gracefully halt the system, notify the operator, and provide system and assembly status displays to enable quick recovery. When the assembly is completed, the microswitch subassembly is placed in the tray and the tray is returned to the conveyor. The next tray of parts is then acquired and assembled into a microswitch. This continues until all n microswitches have been assembled.

Errors produced in fixture placement and off-line programming are handled by the ITA System sensors; thus each assembly can be one of a kind. However, when multiple assemblies of one kind are to be produced, system throughput can be improved by updating or "touching up" the SC data base. This is defined as system autotuning. Thus, when the operator replies to the SC that the proper tools/fixtures are in place, the system can be instructed to perform autotuning. GPRS, PMRS and SPS are calibrated relative to permanent objects in the work cell during routine setup (e.g., the GPRS parts tray coordinate frame is calibrated with respect to the permanent tray registration stops). However, when special fixtures/tools are placed in the work cell, the programs generated off-line must be touched up. Here, SSVIS is first used to verify the SPS pose, accommodating errors within its 1-cubic-inch field of view; these results are fed back to the SC through queries to SPS about its new pose. Then the normal assembly steps are performed on the first tray of 1EN1-6 parts at slow speed with guarded PMRS moves. Each part acquired from the tray is staged and regrasped to ensure a nearly ideal pose in the gripper. SSVIS is then used to correct any misalignment, and the results are fed back to the SC via queries to PMRS about its new pose. After the part-mating operation, the force sensor data is analyzed to provide fine-tuned parameters for that force-controlled operation. Subsequent assemblies will now proceed at a faster rate since fewer sensory corrections are required. SSVIS is still used to accommodate part misalignment in the PMRS hand and to provide in-process verification.

Downtime Mode

This mode occurs whenever the shop floor segment of the ITAS is not in on-line mode or powered off. Downtime mode activities begin with the operator starting up the ITAS; i.e., all subsystems are brought up through a "cold-boot" operation to one of listening for commands from the SC. The SC then verifies the communication integrity and the state of health for each subsystem. If maintenance is required (e.g., sensor or robot calibration), it is performed at this point.

To begin assembling a device (or to change over from one type to another), the operator invokes a simple command such as "INIT CELL 1EN1-6" at the SC console. This causes the SC to load its data base for the 1EN1-6 assembly and to invoke GPRS, SPARS and SSVIS to load their data bases; PMRS and SPS are instructed to load the appropriate programs. The SC then prompts the operator to supply any special fixtures and/or tools required for the 1EN1-6 assembly and where to place them. When the operator notifies the SC that the proper tools/fixtures are in place, the cell is ready to begin assembly in on-line mode.

Programs and data for new, first-time assembly tasks (generated off-line) are tested and debugged in downtime mode. The subsystems' programs and data are modified to achieve acceptable performance using the SC console and/or the appropriate subsystem(s) console.

Off-Line Programming Mode

System programming within the system's assembly domain is accomplished by generating or modifying high-level task control programs and by generating data bases (Process 2). The three data bases for the vision subsystems are interactively generated off-line using training data collected from the on-line vision sensors and preprocessors. A work cell simulator is used to develop and test programs off-line for the robots and the SC and to generate the SC data base.

SC and Robot Programming--CAD models of the parts are generated with a coordinate system defined for each part. These parts are then "assembled" on the CAD system to compute their relative poses when properly assembled. Any tools and fixtures required for this assembly and not already in the ITA System work cell simulation are modeled. All new part, tool and fixture models are then passed to the work cell simulation.

The work cell simulation now contains geometric models of all components in the work cell: robots, fixtures and fingers/tools as well as the parts and the desired subassembly (see Figure 6). It also contains a functional simulation of the system with detailed simulations of the SC and the two robots and their controllers and simplified simulations of GPRS, SPARS and SSVIS. Since the robot controllers obtain all their task-specific data from the SC through procedure call parameters, only programs and data base entries for the SC must be written and tested for the desired assembly task. With the addition of user-specified safe approach/departure points and paths, all geometric information required to program the assembly task is available on the simulation. The resulting programs and data structures that are sufficient to run the simulation correctly are then used to generate the SC data base via a software language translation interface.

After the geometric modeling is complete, the user first defines safe approach/departure points and paths for the tray and all assembly fixtures interactively. Thus, poses like TRAY_SAFE and FIXTURE_1_SAFE become known to the simulation, along with

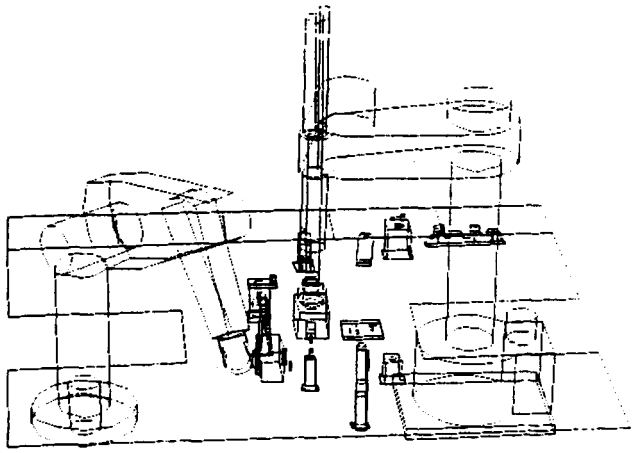


Figure 6. Simulation of ITA System Using CIMStation Tool

lists of "via points" for safe paths like TRAY_TO_FIXTURE_1. The parts are then placed interactively in various poses in the tray.

The user then enters the assembly operators such as LOCATE BUSHING, GRASP BUSHING FROM TRAY, PLACE BUSHING IN FIXTURE_1. As each operator is simulated, the system prompts the user if additional information is required. For instance, LOCATE BUSHING initially causes no prompts; the GPRS simulation simply queries the geometric modeler to provide the pose of the bushing--functionally equivalent to the real GPRS's recognition/location function. However, to determine a grasp strategy, the LOCATE operator lacks information: part-pose label, finger type, finger opening size and grasp position. Therefore, the user is prompted to supply the part-pose label (e.g., BUSHING_PRONE), which enables the simulated LOCATE operator to access its version of the GPRS grasp data base (created separately, as shown below) to obtain the additional data. GRASP BUSHING FROM TRAY then has all the data it needs except grip force, which is entered by the user. PLACE BUSHING IN FIXTURE_1 will prompt the user to "place" the bushing in Fixture 1 by interactively placing the PMRS robot, which is holding the bushing, in the desired pose. Operators like INSERT will require data to be entered such as SSVIS/SPS sensor pose, alignment tolerances, natural and artificial constraints for part mating, and force/position limits. Once the entire assembly sequence has been programmed, the parts are "placed" (using the simulation) in various poses in the tray to test the program and data. The verified assembly task program and data base are then translated and downloaded to the SC.

GPRS Data Base Generation--The GPRS data base is generated interactively off-line using the features extracted from training images of the parts in their various poses in the tray. These training images and features will have been gathered and stored using the GPRS on-line sensors and feature detection processing. First, an image of the part to be trained is selected and a coordinate system is defined for the part that corresponds to the one generated for the SC. Features detected for the part from both gray-scale and terrain map data are interactively selected to generate preliminary recognition models. Recognition strategies (e.g., which sensors to use, which part pose to look for first) are determined by the user. These preliminary models are then refined by exercising them on other training images of the same part in the same pose; during this process the system gathers statistics to adjust the model tolerances appropriately.

SPARS/SSVIS Data Base Generation--The SPARS and SSVIS data bases are generated in similar fashion to GPRS. To gather training data, a copy of the SPARS/SSVIS sensors on a six-degree-of-freedom repositionable fixture emulates the PMRS and SPS robots. First, data are gathered from the parts tray for training the SPARS subsystem. Then, a five-degree-of-freedom positioning fixture that can hold the quick-change fingers/tools to be used by PMRS gathers data for the SSVIS subsystem. Model building then proceeds in a fashion similar to that of GPRS.

SUBSYSTEM DETAILED DESIGN

The detailed design for all of the subsystems described above is currently nearing completion, and implementation of "certified" subsystems and/or their components has begun. Highlights of the designs are presented below.

System Controller

The primary function of the SC is to control, coordinate and monitor the subsystems that make up the ITA System. To maximize the system's effectiveness at its tasks, the SC must also manage the concurrent use of system resources; detect, respond to and correct error conditions whenever possible; and notify the shop floor user in the event of an unrecoverable error. All system, configuration and assembly data will be maintained in a "world model" of the ITAS within the SC. Facilities for development of new assembly applications, for editing existing assembly programs, and for system maintenance and calibration are also part of the SC functionality. The SC will provide test and debugging facilities for use in integrating subsystems into the ITA System during system development.

User Interface--The user interface requirements for each of the SC operating modes were determined using Reference 10 (among others) as a design guideline for sequence control, user guidance, data display, data entry and data protection. Since menus/forms are the most straightforward candidates for implementing the interface, requirements based on these candidates can be readily applied to others. Sample menus were generated to define requirements.

Error Recovery--Three levels of sophistication were envisioned. To recover after detecting an error condition, the SC could notify the operator, execute preprogrammed scripts of assembly operators/primitives, or execute commands from an "expert advisor." Three error scenarios were analyzed to determine the requirements of automatic recovery: (1) GPRS cannot find the desired part in the tray; (2) during preassembly verification, SSVIS indicates that the part is not in PMRS's gripper; (3) during post-assembly verification, SSVIS indicates that the white seal is too cocked in the bushing for the assembly to proceed.

In addition, since assembly-specific errors will be difficult to anticipate while off-line programming the ITAS, error recovery must be quick and easy to program on the shop floor. Both a simple interpretive IF/THEN shell as well as a graphical, expert-system knowledge-base tool [11] will meet the requirements for preprogrammed recovery procedures; however, an IF/THEN shell is the baseline design.

Software Architecture--Given the overall SC requirements implied above, the baseline software architecture shown in Figure 7 was established. Software modularity and concurrency are facilitated by a multitasking and/or multiprocessor environment with synchronized interprocess communications (i.e., a data exchange manager).

SC Platform Selection Criteria--The following SC hardware/software selection criteria were established. The SC *must have:* (1) real-time, multitasking operating system; (2) C language (or equivalent) development capability; (3) fast interprocess message exchange. The SC *should have:* (1) existing software applicable to ITA; (2) mature platform, near-term availability, good vendor support, *reasonable cost*; (3) *intelligent I/O capabilities*; (4) 68020 CPU (or equivalent) and 68881 math coprocessor (or equivalent); (5)

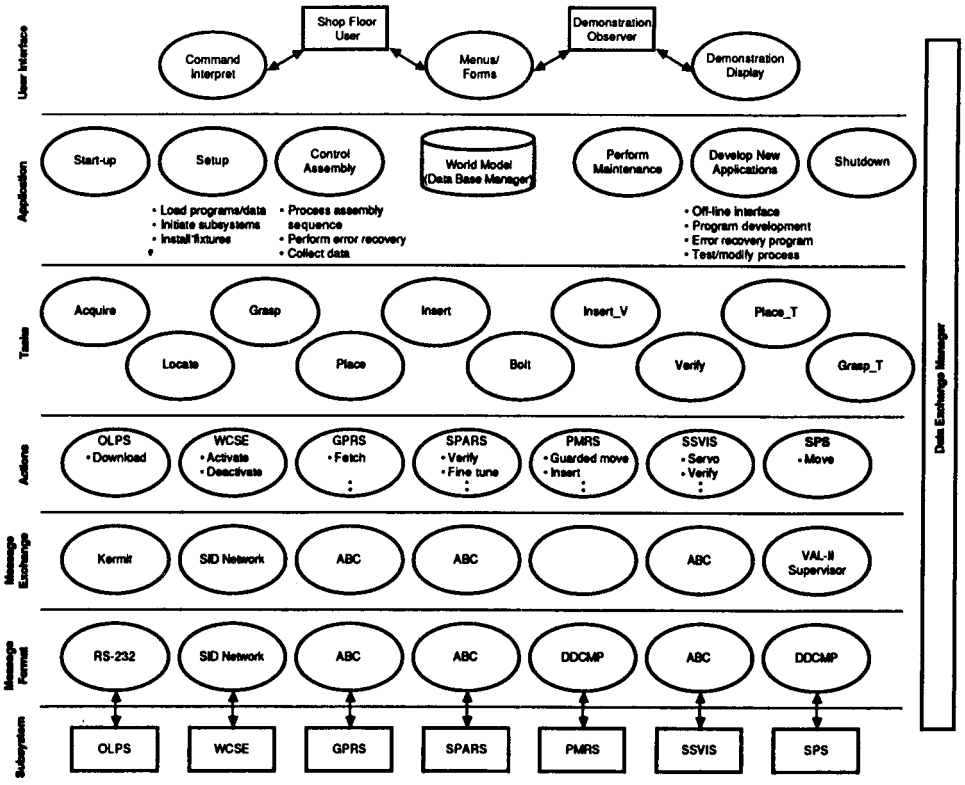


Figure 7. System Controller Baseline Software Architecture

multiuser development capability; (6) 2- to 4-MB (minimum) physical memory; (7) 60-MB (minimum) hard disk.

These criteria were then weighted and used to evaluate candidates as objectively as possible. The selected platform consists of a commercially available area controller [12] coupled with an internally developed communication gateway processor.

Global Part Recognition Subsystem (GPRS)

FETCH Process--This process provides the main functionality. For each part in the "short list of parts," a process, "Determine Acquisition Strategy," manages the following processing steps: (1) acquire imagery appropriate for the part; (2) detect features required to recognize the part; (3) build hypotheses of feature clusters that could be the part and rank them; (4) verify, in ranked order, each hypothesis to select one, if any, that is the part; if a hypothesis is probably the part, but requires verification by SPARS, label the part "quasi- found;" (5) if the part is found, capture that knowledge (to determine, if necessary, whether this part is occluding another part) and remove this part's features from further consideration; (6) if the part is found, try to find an unobstructed grasp access using preferred tool(s).

Feature Detection Process--Based on the Phase I GPRS real-time implementation study, as much of this process as possible should be implemented on an array processor. Thus, the process was decomposed sufficiently to isolate the first primitive processes for implementation; namely, the Difference of Gaussians edge detector. The resulting pseudo-code primitive process specifications have been implemented on the array processor to reduce risks with the Phase II hardware (see below).

Recognition and Grasping Algorithms--Using the Phase I GPRS functional performance results based on processing imagery from 25 tray dumps, performance and/or algorithm deficiencies were analyzed to identify any problem areas requiring immediate attention. Problem areas appear mainly in (1) the hypothesis verification step of the recognition process for small, similar-looking parts, and (2) grasp determination performance, particularly for parts with highly specular surfaces. Otherwise, the baseline algorithm suite appears to be adequate to meet requirements. Since all sensors will be new in Phase II, no further algorithm efforts (other than optimization analyses) were deemed necessary until the algorithms could be tested with data from the new sensors.

Non-Real-Time Phase II Hardware--Funding constraints preclude implementation of real-time vision subsystems in Phase II. However, functional conformance to requirements must be demonstrated within a subjectively "reasonable" time line of 16 to 25 minutes to assemble the microswitch (versus 5.4 minutes in real time), while real-time conformance will be demonstrated using a simulation of the real-time system design. As shown in Figure 8, components were selected and interface details were resolved to configure the Phase II hardware platform.

Close-Range Part Alignment and Recognition Subsystem (CRPARS)

The design activities established that a Close-Range Part Alignment and Recognition Subsystem can be developed that meets the requirements of both the Sparse Part Acquisition and Recognition Subsystem (SPARS) and the Sparse Servoing, Verification, and Inspection Subsystem (SSVIS). This represents a major simplification of the system design and development since two copies of CRPARS, rather than two different subsystems, need to be built and tested.

CRPARS Sensor Redesign--Since redesign of the Phase I prototype sensor was required to miniaturize and ruggedize it for mounting on the end effector of the high-speed PMRS, a review of the Phase II SPARS/SSVIS requirements and Phase I experiences

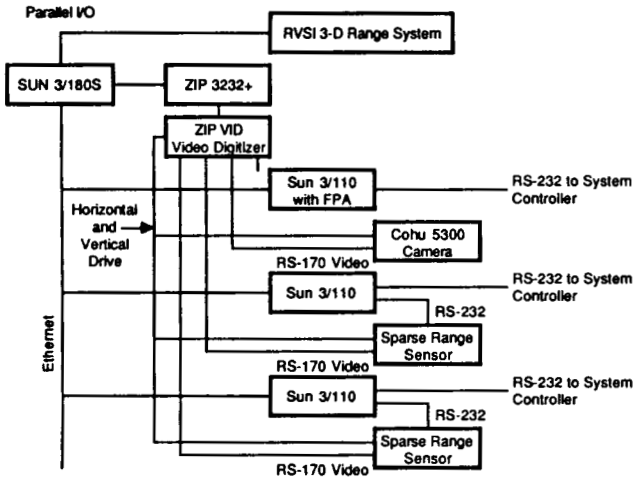


Figure 8. Phase II Vision Hardware Platform

was performed to design the CRPARS sensor. A 1-cubic-inch field of view is required to accommodate off-line programming and fixture placement errors–difficult to achieve with a small number of light stripes. Moreover, Phase I experiments showed problems with (1) obtaining repeatable light-stripe data, especially from specular surfaces; (2) interpreting the data from complex scenes; and (3) training and tuning light-stripe recognition models. Solutions to these problems are to (1) augment light-stripe data with binary or gray-scale data for verification and rough pose measurements; (2) add two horizontal light stripes to the two vertical stripes for SSVIS assembly verification and interpart alignment tasks; (3) use binary or gray-scale data to reposition the CRPARS sensor when light-stripe recognition has failed but when 2-D conditions look promising enough to warrant another verification attempt.

Three of the many candidates were evaluated; namely, stereo gray scale, stereo gray scale with a single light stripe, and monocular gray scale with two to four light stripes. Selection criteria included hardware/software maturity, illumination needs, size/weight, training and calibration complexity, and speed. Monocular gray scale with two to four light stripes (two for SPARS, four for SSVIS) was chosen; stereo gray scale with a single light stripe was a close second choice, but maturity based on Phase I prevailed. A component selection study resulted in the design shown in Figure 9.

CRPARS Algorithms--Algorithms for in-process monitoring and verification of expected conditions were analyzed and compared to the GPRS recognition process to maximize commonality between the two subsystems. Algorithms to measure object pose must be implemented on an array processor to meet real-time requirements. Thus, they were decomposed and analyzed sufficiently to verify that they could be implemented as planned on the selected Phase II array processor (see Figure 8).

Part Manipulation Robot Subsystem

The PMRS manipulator will consist of a five-axis AdeptOne™ SCARA robot. Since the PMRS end effector consists of a JR3 Corp. force-sensing wrist, a LORD Corp. instrumented, servoable gripper, and the SPARS (two-stripe CRPARS) sensor, the evolution of its physical configuration and payload is being monitored carefully. Using component weight estimates and a static torque analysis, the baseline configuration is within the robot's payload limits; its length drives the need for a 12-inch stroke on Joint 3 of PMRS (see Figure 5).

The Phase II PMRS hardware architecture shown in Figure 10 was designed and implemented. The major change from Phase I was replacement of the Weitek Corp. array processor with a more easily

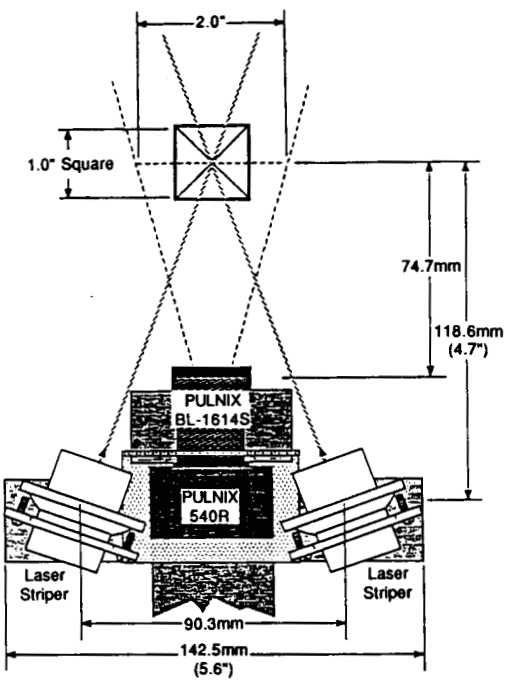


Figure 9. CRPARS Sensor Design with Two Light Stripes

programmable and lower cost Motorola 68020 CPU/68881 math coprocessor single-board computer. A joint interface board for the fifth axis was also added. The LORD Corp. gripper has its own controller, which communicates with the Adept robot controller over an RS-232 line. The Phase I force servo control was ported to the Phase II hardware to obtain a 250-Hz force servo rate to mesh with the 500-Hz joint servo rate. After converting the force matrix multiplication from floating point to integer computation, implementing the forward arm solution in VAL II, and extending control to five axes, force servo processing requires 3 out of the available 4 milliseconds.

The Phase II system performs comparably to the Phase I system with some additional features. Five-axis force servoing allows specification of translation stiffnesses in cartesian tool (end-effector) coordinates, referencing of orientation stiffnesses to individual joints (which seems appropriate for a SCARA robot), and implementation of a true software remote-center-compliance (RCC). Joint limit stops, which gracefully pull the arm back into the active region when the offending force is removed, have replaced the panic shutdown monitor from Phase I; while performance is less than ideal, it is computationally efficient. Real-time gravity compensation, which tracks the movement of Joint 5, subtracts sensor bias and payload weight to leave contact forces; an automatic calibration routine takes measurements at several points and computes parameters using least squares.

Sensor Positioning Subsystem (SPS)

A standard six-axis PUMA 560 from Unimation/Westinghouse appears to meet all the requirements of the SPS, including communication with the SC via its "supervisor" interface using the DDCMP protocol.

Work Cell Support Elements (WCSE)

The WCSE's primary functions are to provide the tools and fixtures necessary to assist the one-handed PMRS assembly operations and to provide the mechanical and structural interface for portions of the other subsystems. Attention has focused on using the detailed

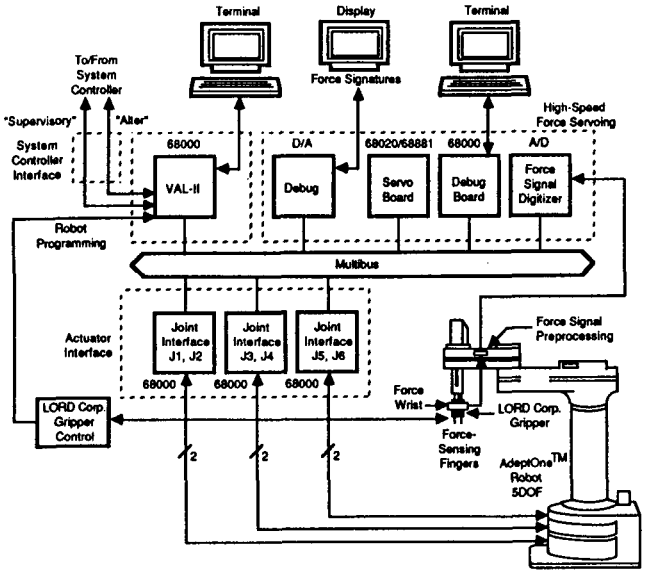


Figure 10. PMRS Detailed Hardware Architecture

assembly scenario for the 1EN1 microswitch to develop preliminary fixture and tooling prototypes for experimentation with the PMRS robot when it becomes available. The design goal is to make these as simple and multipurpose as possible.

Off-Line Programming Subsystem (OLPS)

The OLPS must provide the ability to perform system programming, program validation, program downloading and data base generation interactively for the on-line segments of the ITAS. In the Phase II ITAS, the programmer interface will not be a single one; rather, there will be one interface for the SC, one for the vision subsystems, and one for the robots (see Figure 2). For the SC, the OLPS must provide, as a minimum, the same capability as the "Develop New Applications" process does for the shop floor user. Rather than positioning the robots in position or force-teach modes, however, the simulated robots are used where the baseline implementation assumes use of the CIMStation simulation package. The results of preliminary off-line programming tests for end-to-end error buildup along with inspection of the VAL code generated by the translators were deemed adequate to keep CIMStation as the baseline. For the vision subsystems, the OLPS will consist of a copy of the shop floor interactive training processes developed for GPRS and CRPARS. Future systems would either have copies of the shop floor sensors or, ideally, synthesized training imagery from a CAD-based geometric modeler and sensor simulator. The baseline PMRS and SPS designs assume that suites of data-driven primitives can be developed such that a variety of assembly tasks can be performed without writing new primitives. In the event that this is not a valid assumption, assembly-specific routines will be developed on CIMStation, translated to the appropriate robot controller's language, and downloaded.

CONCLUSIONS

The process of engineering emerging technologies into a functional demonstration system is essential for rapidly maturing experimental implementations, validating their efficacy and, ultimately, transferring useful technology and experience. Application of systems engineering tools and techniques is an efficient approach when carefully tailored to the research nature of such projects.

ACKNOWLEDGEMENTS

This work is funded by Air Force contract No. F33615-82-C-5092 under project managers, Mr. T. Brandewie (AFWAL) and Dr. R. Rosenfeld (DARPA). The author gratefully acknowledges the contributions of many individuals at Honeywell and Adept Technology, including D. Waters, R. Bell, R. Blair, R. Graber, M. Harless, T. Hill, R. Jacobson, T. Keefe, D. Larkin, J. Mannering, J. Maples, G. Marschall, V. Nibbe, C. Raymond and B. Westover.

REFERENCES

1. Levinthal, E. C., "DARPA Program--Intelligent Task Automation," *Robotics and Industrial Inspection, Proceedings of SPIE, The International Society for Optical Engineering*, San Diego, CA, Vol. 360, August 24-27, 1982, pp. 32-38.

2. Rosenfeld, R., "Intelligent Task Automation Manufacturing Systems," *Proceedings, Second Annual Conference on Robotics and Factories of the Future*, Springer-Verlag, San Diego, CA, July 28-31, 1987.

3. Reimann, Walter, "Air Force Assembly Thrust," *Proceedings, The 1985 United States Air Force Computer Integrated Manufacturing*, Dallas, TX, April 9-12, 1985.

4. Benton, R. C., "Enabling Technology for Flexible Assembly Systems," *SAMPE Journal*, January/February 1985, Vol. 21-1, pp. 25-33.

5. Benton, R. C. and Waters, D. E., "Intelligent Task Automation," *Proceedings, The 1985 United States Air Force Computer Integrated Manufacturing*, Dallas, TX, April 9-12, 1985, pp. 195-214.

6. Honeywell Inc., "Intelligent Task Automation--Phase I," Volumes I, II, and III, Report No. AFWAL-TR-86-4122, Air Force Wright Aeronautical Laboratories, AFWAL/MLTC, December 1986.

7. Hatley, Derek J., "The Use of Structured Methods in the Development of Large, Software-Based Avionics Systems," Paper No. 84-2595, American Institute of Aeronautics and Astronautics, 1984.

8. Interactive Development Environments, "Software through Pictures™ Reference Manual," 150 Fourth Street, Suite 210, San Francisco, CA 94103.

9. Craig, John J., "Issues in the Design of Off-Line Programming Systems," Proceedings of the Fourth International Symposium on Robotics Research Conference, Santa Cruz, CA, August 1987.

10. Smith, Sidney L., and Mosier, Jane N., "Design Guidelines for User-System Interface to Software," The Mitre Corporation, Project No. 522A, September 1984.

11. Hutchins, B.; Wolff, A.; Cochran, E.; and Ludlow, P., "Design of a User Interface for Automated Knowledge Acquisition," *Proceedings of the 1986 IEEE International Conference on Systems, Man and Cybernetics*, Atlanta, Georgia, 1986.

12. Adept Technology, "Assembly and Information Management System Reference Guide," Version 1.5, 150 Rose Orchard Way, San Jose, CA 95134, April 1987.

CAD Based 3-D Object Recognition

(Paper not provided by publication date)

SYSTEM INTEGRATION of a TELEROBOTIC DEMONSTRATION SYSTEM (TDS) TESTBED

John K. Myers
SRI International, Robotics Laboratory
Menlo Park, California

Abstract

This paper describes the concept for and status of a Telerobotic Demonstration System testbed that integrates teleoperation and robotics. The system is being developed by the Jet Propulsion Laboratories with technical assistance from SRI International. The components of the telerobotic system are described and the projects performed by SRI International are discussed. The system can be divided into two sections: the autonomous subsystems, and the additional interface and support subsystems including teleoperations. The autonomous subsystems are scheduled to be demonstrated separately at the end of 1987, and the entire, integrated telerobotic system is scheduled to be demonstrated at the end of 1988.

Acknowledgments: Other members of the consulting team at SRI International include Ron Cain, Cregg Cowan, Jim Herson, Tony Sword, and Doug Ruth. The project was supervised by Jan Kremers and David Nitzan. I would like to thank the contract monitor, Wayne Zimmerman of JPL, for excellent management and for significant contribution to most of the ideas expressed in this paper.

1 Overview

The Jet Propulsion Laboratories is constructing a telerobotic demonstration testbed system, known as the TDS, where the current state-of-the-art in robotic and teleoperation concepts can be integrated, tested and explored. Although building a space-worthy robot is not an immediate goal, the system concepts and experience obtained will be useful and may eventually be applied to a deployable system.

In order to demonstrate a realistic application, the task of servicing a defective satellite has been selected. The satellite has a defective electronics module contained in a backplane-type slot. As currently envisioned, the repair operation will proceed as follows: The operator will use teleoperation to fold back a flexible foil space blanket and attach its corners to holding tabs. The operator will then place the system in autonomous mode, and instruct the system to replace the electronics module. To accomplish this task, the system will plan and execute motions to unscrew and remove the door, unplug electrical connectors from the electronics module, remove and replace the module, replug the connectors, and replace and rescrew the door.

Figure 1 shows the layout of the TDS setup. Two Unimation PUMA robots perform the actual work on a satellite mockup. A third robot holds a stereo camera pair for the vision-system and for operator feedback. Three additional cameras are positioned on the walls and ceiling of the room. A rack for quick-change tools and spare parts is positioned in front of the robots. The robots are mounted on lathe beds to provide one additional degree of freedom. Figure 2 shows an early version of the actual testbed. Figure 3 shows a solid-model simulation of the testbed, which will be discussed further in a following section.

SRI International is consulting for JPL on the integration of the different parts of the system. The telerobotic system itself is constructed of four subsystems that comprise the main, autonomous part of the robotic system, and three additional subsystems that provide for operator interaction and support the system. The original layout and most of the actual coding of the system was determined and continues to be developed by JPL. SRI is helping JPL to specify the functions of each of these subsystems, and to develop the communication between them Besides this general consulting, SRI currently has four deliverable tasks.

This paper will first discuss the structure of the TDS. Each of the four autonomous subsystems will be defined and examined, followed by the three additional subsystems. After this, the four deliverable projects that SRI is performing will be examined and discussed.

[1]This paper and the consulting work described herein were supported by Jet Propulsion Laboratories under Contract 957908.

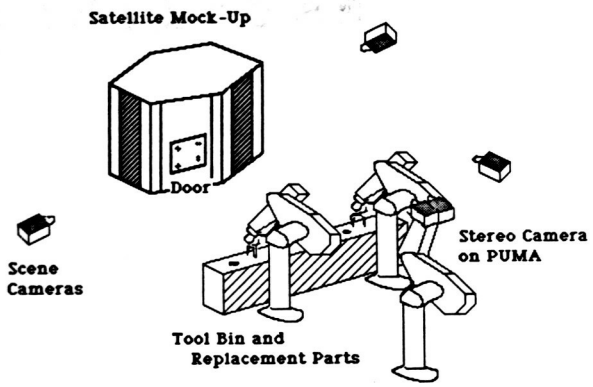


Figure 1: The Layout of the Testbed Setup

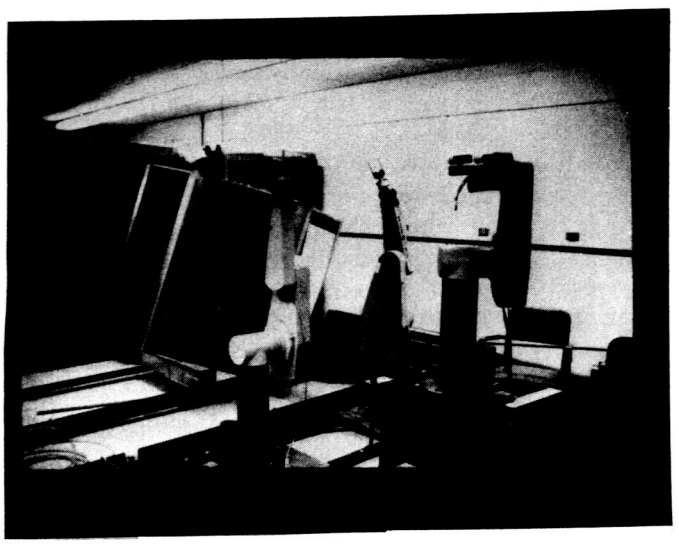


Figure 2: The JPL Testbed

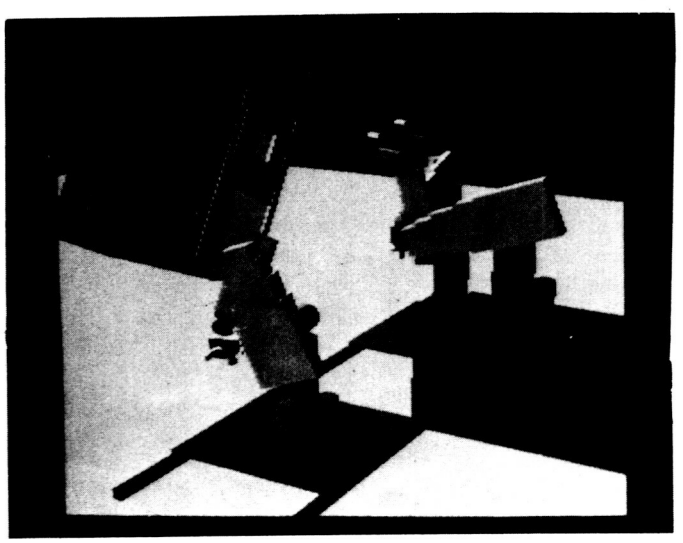


Figure 3: A Simulation of the Testbed

2 The Autonomous Subsystems

The autonomous portion of the TDS consists of the following four subsystems: Artificial Intelligence Planning (AIP), Run-Time Control (RTC), Manipulation and Controls Mechanization (MCM), and Sensing and Perception (S&P). See Figure 4. Each of these subsystems runs on its own separate computer. The AIP uses a Symbolics Lisp Machine, while the other subsystems use MicroVax II computers.

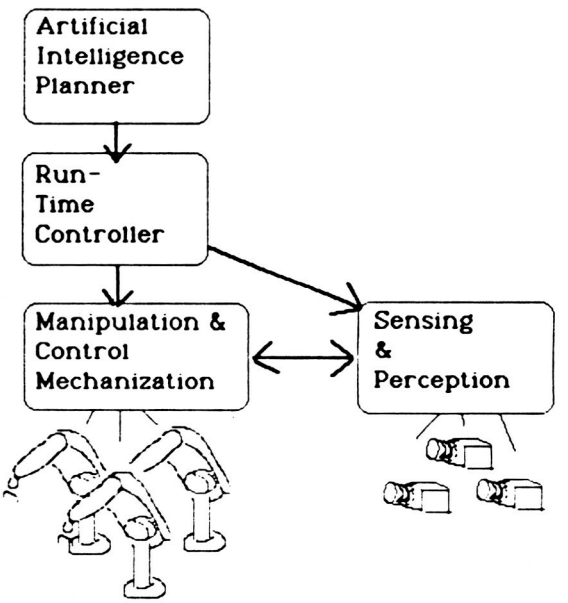


Figure 4: The Autonomous Subsystems

2.1 The Artificial Intelligence Planning Subsystem

The AIP is responsible for planning the actions to perform when the system is under autonomous control. The input to the AIP subsystem is a description of the problem, and a statement of the goal to be accomplished. For instance, in the demonstration task, the input is a description of the state of the satellite, (including the fact that the electronics module is defective), and the goal that the satellite be healthy. The AIP plans the actions to take, and directs the Run Time Control to execute those actions.

Internally, data used by the AIP consists of a rule base describing different conditions and the order in which different procedures must be accomplished. For example: in order to have a healthy satellite, the module must be replaced; in order to replace the module, first the old module must be removed and then the new module must be inserted; in order to remove the old module, the door must be open; in order to open the door, the retaining bolts must be unscrewed; in order to unscrew the bolts, the arm must pick up a nutdriver and then perform the unscrewing action. These rules are represented and stored in a commercial expert system, ART, which is used to reason about what robot system actions must be taken.

The output from the AIP is a series of symbolic actions, such as "Move to above tool bin," "Pick up nutdriver," "Move to above bolt #1," and "Unscrew bolt." These are passed to the Run-Time Control.

2.2 The Run-Time Control Subsystem

The RTC is responsible for instantiating symbolic actions into robotic motions for the system to execute. It takes commands from the AIP, and coordinates the functions of both the MCM and the S&P subsystems. The RTC uses a collision-detection spatial simulator to verify motions. In the future, it will use a collision avoidance module to plan paths around obstructions.

For example, given the symbolic command "Move to above bolt #1", the RTC might perform the following. First, the RTC accesses the location of bolt #1 in a database to instantiate it into a "[4×4]" homogeneous coordinate transformation matrix. If the precise location is not known, or could have changed, the RTC directs the S&P subsystem to verify (or determine) the current location using vision. Next, the RTC uses a predefined "above" distance for that particular bolt to compute the actual location to move to. After this, a "move to above" program is accessed, which may actually contain several individual arm motions, depending on where the robot is at the present time. The RTC executes a predictive collision detection simulation of the proposed motion, to ensure that the robot will not collide with anything when it moves. Finally, the actual instantiated robot system commands are scheduled and sent to the MCM subsystem, and the S&P subsystem if required (e.g., in the previous example).

2.3 The Manipulation and Controls Mechanization Subsystem

The MCM subsystem directs the robots. It takes commands from the RTC, and executes the robot motions on the hardware. The MCM uses force feedback, if required, to modify the motions of the robots. Currently the MCM interfaces with the VAL controllers of the PUMA robots; in the future, the MCM may control the robots directly.

The MCM is responsible for performing atomic (basic, low-level) motions. However, it does have some "reflex macro" motions that are considered to be atomic, but are actually composed of a series of motions. For instance, the robot's nutdriver might be positioned right above the bolt, and the MCM might be instructed to "execute unscrewing motion." The robot lowers the nutdriver onto the bolt head, and "feels around" until the nutdriver is seated. Then, the robot rotates the nutdriver, maintaining appropriate pressure in the direction of the bolt shaft, until the bolt is unscrewed. This "macro" is actually a series of motions. Since this action sequence will always be the same, and is only varied on the fine motion scale based on force sensor readings, the action is considered to be an atomic motion.

2.4 The Sensing and Perception Subsystem

The S&P subsystem is responsible for verifying the locations of objects by using visual feedback. The S&P subsystem has three-dimensional models of all of the viewable parts in the testbed. It uses these models, and an edge image of the scene extracted from the gray-scale image, to perform verification of the position and orientation of parts. The vision system can track moving objects (using a Kalman filter to predict the location of moving objects based on time), use information from multiple camera sources taken at different times, and verify the locations of partially occluded objects. In addition, randomly positioned objects can be searched for and visually acquired; however, this takes significantly longer. See [1] for further details.

The S&P reports its results to the RTC, and also sends results directly to the MCM when requested. An example of a command might be to "verify the location of bolt #1 at approximate location X." The S&P decides the most appropriate camera for viewing that location, takes a picture and computes the edge image, verifies the bolt's location using the visual model of the bolt, and returns the refined location to the RTC.

3 The Additional Subsystems

Besides the autonomous subsystems, there are three additional subsystems that support the TDS and provide important functions. These are: Teleoperations (TELEOP), the Operator's Control Station (OCS), and the System Executive (EXEC). See Figure 5.

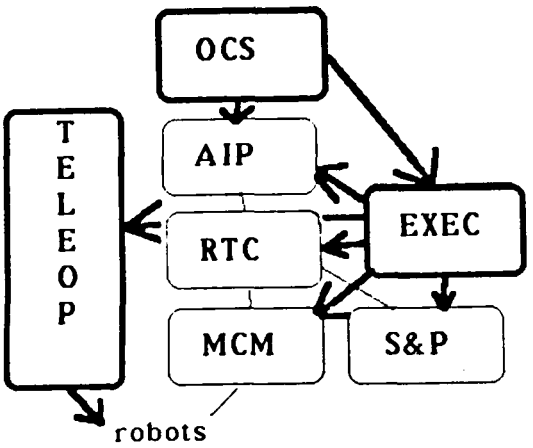


Figure 5: The Additional Subsystems

3.1 The Teleoperations Subsystem

The TELEOP permits the robots to be operated by a person. This allows greater flexibility in actions that can be accomplished, by permitting the operator to perform actions that cannot be done autonomously. The TELEOP controls two Salisbury "hand controllers," which each consist of a handle in a gimbaled cradle attached to the end of a pivoted telescoping shaft. Each of these controllers permits input in six degrees of freedom. The hand controllers' mechanical system are "counterweighted" and use bearings to allow fast and smooth motion. Besides passively providing location as an input to the system, the hand controllers also actively reflect force to the operator's handles as a feedback output. This is used for example to communicate contact at the end of the arm, as detected by the arm's force sensor, and to slow the hand controller down if the operator is slewing too rapidly and the robot arm is lagging too far in its tracking. In the future it will also be used to reflect virtual forces from imaginary "force fields" around modeled objects, and so assist the operator in avoiding collisions [2].

The TELEOP also supports the switchover from autonomous mode to teleoperational mode and back again. In the present design, the two system modes are essentially disjoint. The autonomous system runs by itself and drives the robot arms. At any time, the operator can request or demand a switchover, and the autonomous system either gracefully shuts down, or aborts all actions and returns control to the teleoperator. The teleoperation system then becomes active; the teleoperator can move the arms and remedy any anomalous conditions (such as dropped, wedged, bent or damaged parts), or can perform necessary actions that are beyond the dexterity of present-day autonomous robotics. When the teleoperator is finished, he or she returns control to the autonomous part of the system, and the TELEOP becomes inactive.

An open research issue is the best way of switching from teleoperator to autonomous control. The autonomous subsystems depend on knowing the approximate location of all objects in the system. In both the present-day TDS, and in the eventual space application, this is a reasonable assumption; once the satellite has been acquired and fixed relative to the robots, the robots, tools, and satellite parts become a closed system. After teleoperation, however, the information in the autonomous subsystems' databases may be invalid. Old parts may be unexpectedly moved or missing (e.g., dropped on the floor); the operator could conceivably introduce new parts or sufficiently unfamiliar configurations or modifications of old parts such that they would be unrecognizable by the system.

Assuming that the difficulties are restricted to relocations of old parts, at least four possibilities for solving this problem are being considered. The autonomous system could direct the teleoperator and give instructions as to which parts he or she is allowed to work on. Or, the teleoperator could pick from a menu of standard teleoperation procedures or states to inform the autonomous system of the status of the system. Alternatively, the teleoperator could explicitly tell the autonomous system about each object that was moved. An advanced autonomous system could reinitialize its view of the world by verifying the locations of all expected parts and recognizing the intruding locations of all relocated parts.

Perhaps the best solution would be to convert the system from one that is disjoint between the autonomous and teleoperation modes, to one where the two parts are cooperative and the distinction is blurred. In a futuristic system, the autonomous part of the system would remain on all the time, and "watch over the operator's shoulder" as the teleoperator works. It would observe where the operator is placing parts, so that even during teleoperation, the system would have a full, accurate model of the locations of objects. The autonomous system would also attempt to understand what actions the teleoperator would be performing, and guess what he or she would be trying to accomplish. The autonomous system could then direct additional arms to assist the teleoperator, or take over if a routine task (e.g., unscrewing the bolt) is being performed.

3.2 The Operator's Control Station Subsystem

The OSC subsystem consists of a number of display screens and a keyboard in an ergonomically designed layout. The operator can monitor the status of the system and enter commands for the system to execute. The commands are sent to the AIP, the TELEOP, or the System Executive (to be discussed). The different ordinary and emergency status messages are displayed for the operator; the operator can also obtain views of the work from any of the cameras, and displays of solid models of the testbed describing what the system believes the current location of robots and objects to be. The OCS will be equipped with discrete-word voice input, and voice output for status messages and alarms. In addition, a dual-screen superimposed polarized display will allow three-dimensional viewing to operators wearing polarized glasses. The input for this will probably be taken from the stereo camera pair mounted on the third arm, although it could be computer-generated from solid models of the scene.

3.3 The System Executive Subsystem

The EXEC subsystem is responsible for configuring the system, testing each individual subsystem and the integrated system as a whole, and managing the health of the system. It can suspend and resume the entire system or pieces of the system, and it is also responsible for maintaining an initialization database for the entire system. It manages all of the other subsystems.

The EXEC maintains a library of executable programs for the system. This ensures that the system's software is consistent, and that the different versions of executable files for the different subsystems are kept up-to-date. When the system is initially turned on, the EXEC is responsible for configuring the system. Executable files are downloaded and the system is "brought up" piece by piece. The EXEC also maintains an initialization database, which is downloaded to the different parts of the system once the system is running. These will be discussed in greater detail in following sections.

Once the system is running, the EXEC is responsible for testing each of the subsystems in turn, to ensure that each one is fully configured and capable of running. Each subsystem is also directed to test its hardware, if any, and report the results back to the EXEC. After this, the system as a whole is tested: the AIP is directed to plan a minute, single movement and take a picture. This command is watched as it filters down through the RTC to the MCM and S&P, and as the results are returned via the RTC to the AIP. If everything works properly, then the system as a whole is up and running. Future commands will similarly test the TELEOP and OCS subsystems' operation in the system as a whole. In addition, the EXEC will eventually be able to "watch over the shoulder" of the system as it executes tasks, detect when a computer has become "wedged" or has "crashed," and recover the system from this state.

The EXEC is also responsible for gradual and emergency shutdowns of the system, and the corresponding resumption of execution. There will be several grades of shutdown, depending upon whether the operator wants control when convenient to the system, "soon," or immediately; whether the arms are expected to retreat to a convenient safe position, finish the current process and then stop, or freeze "dead in their tracks;" and whether the computer processes are expected to be able to resume from where they left off, start over, or be completely deleted. System resumption will similarly have to take different forms, depending upon the state of the robots and computer processes.

4 The SRI Projects for the TDS

In addition to general consultation on the design and development of these subsystems, SRI International is providing four deliverables for the TDS. These include: the Network Interface Protocol (NIP), the Robotic Simulator with Collision Detection (RCODE), the System Configuration package, and the System-wide Initialization Database and Editor. The position of these packages in the TDS is shown in Figure 6.

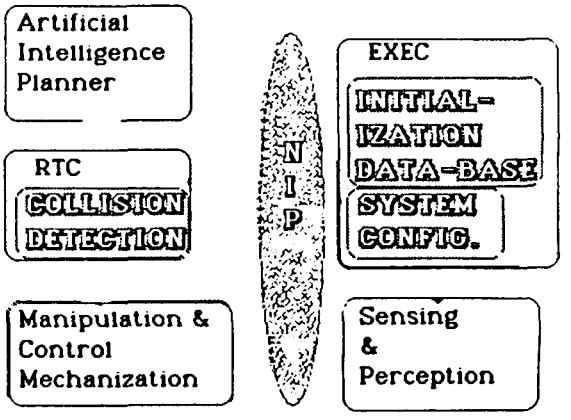


Figure 6: SRI International Contributions

4.1 The Network Interface Protocol

The Network Interface Protocol (NIP) is a package of software that allows communications between different submodules, with the emphasis on robotic applications. It has been delivered and installed on MicroVax machines, and SRI is currently finishing development of a version for the Symbolics. The NIP is currently implemented on top of DECNET for the Ethernet. However, one of its goals is to separate the implementation of the communications channel from the actual communications themselves. So, if a new technology of communications channel is chosen, none of the subsystems have to be modified; only the implementation-specific levels of the NIP itself have to be changed.

The NIP is specifically designed to facilitate robotic communications, which possess slightly different characteristics from file transfer or mailbox applications. Thus, the NIP must support these characteristics. One difference is timing. In robotics, transactions typically consist of a command issued from a "master" machine to a "slave" machine. The transaction remains open until the slave machine replies to the command. However, instead of replying directly, or in a few tenths of seconds, in typical robotic applications the slave (for instance) might execute a motion with a physical arm and not reply back until the motion has finished, which may require seconds or even tens of seconds—and the delay time may be unknown ahead of time.

Another difference is completion. A robotic motion may be prevented from finishing (especially in the case of force servoing) but still be active, so that it is problematical to state even objectively whether the motion has failed or not. Similar real-world effects exist in vision applications, where (for instance, if presented with a textured pattern, perhaps caused by a bad reflection) an unknown, significantly large number of "blobs" or "edges" can effectively cause the vision system to stop returning answers, while the vision system itself believes that it is properly performing its functions. In addition, robotic applications are notorious for finding unexpected ways to crash the computer they are running on. Problems such as trapping on division by zero or stack overflow, or handling a dropped synchronous communications line to a hardware device, must be detectable and recoverable. Thus, robotic communications must be flexible: they must support transactions that remain open over long periods of time, where it is problematical whether the slave will actually return with an answer or not.

However, other characteristics must be supported. The master might not be able to afford waiting for the return, so the communications must have the option to be pollable: the master can continue processing, and periodically check back to see whether a message has arrived yet or not. Some communications are urgent and should not remain in an input queue, so they should be able to trigger interrupt servicing routines in the application program. With some communications, normal processing cannot proceed until an answer is returned, so the NIP must also support waiting for a response, with an optional time-out clock. Other requirements, such as supporting simultaneous conversations, are too numerous to mention here. The NIP provides such a communication package that is tailored for robotic applications.

4.2 The Robotic Simulator with Collision Detection

The Robotic Simulator with Collision Detection (RCODE) presents a spatial occupancy model of the robots and parts in the scene, that is used to determine whether a collision would occur with a certain movement or not [3]. Objects are modeled using a CSG (constructive solid geometry) system, employing the volume primitives sphere, cylinder, box, and half-space, and the construction operator union (intersection and subtraction are not supported, due to the nature of the algorithms). Device-independent wire-frame and Z-buffer shaded-surface graphics are provided by the system; an example of a model of the TDS is shown in Figure 3. Joint-interpolated, straight-line, and user-specified trajectories are supported. Movement simulation is performed using the stepped-move approach; that is, a single, stationary scene is tested, the moving robots' positions are incremented slightly, and the next scene is tested. The system can test an average scene in about 0.2 seconds on a VAX 750, through use of a hierarchy of enclosing volumes.

The RCODE package has been installed at JPL. It is used by the RTC subsystem to verify arm motions to be sent to the MCM. Other collision detection algorithms, such as that proposed by Canny [4], are also being investigated. Currently, the arm can only move directly to a specified goal location, and the collision detection package verifies the proposed path. In the future, a spatial occupancy simulator may be used by a routine to generate original collision-free paths, such as the one reported in [5]. Collision-free path planning is important because it is the link between artificial intelligence and robotics that allows the system to start to become truly autonomous.

4.3 The System Configuration Package

At startup time, the System Configuration package downloads executable and data files to all of the other computers, and establishes computer processes and communication links in the system, in order to bring the system up. A schematic diagram of the configuration package is shown in Figure 7. The configuration package is currently in the design stage, and will be delivered by the end of 1987.

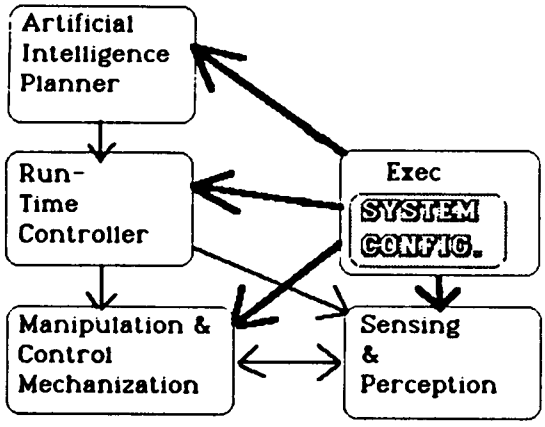


Figure 7: System Configuration Package

Configuration actions that the System Configuration package must perform include: downloading executable files, downloading data files, killing all unauthorized processes, starting a process by executing a file, initializing (starting up) a running process by downloading initialization parameters, downloading an initialization run-time data-base to a running process, establishing communication links between running processes, prompting the user and waiting for verification (for such things as turning on a hardware device), testing the status of a process, killing a specified running process, testing a hardware device controlled by a process, testing a communications link between two processes, and allowing an initialized, running process to "take off" and actually perform in the system.

Even given the actions needed to set the system up, configuring a robot system is not straightforward because of dependencies that exist in the order in which the configuration actions must be performed. For instance, in the TDS, the subsystems are arranged in a control hierarchy so that some computers send command messages to other computers. This requires those computers to be running, initialized, and ready to receive those commands, before those commands are sent. Before any system messages can be sent at all, the different communication links must be established between processes. Many of the processes require initialization, or special data-bases to be downloaded to them, after they are running but before they are ready to become part of the system.

In the initial version, the order dependencies will be handled by a programmer creating a command file of configuration actions that is sent to a command interpreter driving the configuration package. Subsequently, a simple backwards-chaining rule-based system will be created, to take a list of dependencies, automatically generate the proper order, and drive the configuration package.

4.4 The System-Wide Initialization Database and Editor

The Initialization Database will be used as part of the system configuration sequence to initialize the system with a given status, i.e. appropriate parts of the database are downloaded to running processes. The Initialization Databas is responsible for the entire system; the database must store all the data used by all of the different subsystems in the TDS. Therefore, the design of the database must be general enough to store all types of data currently used by the system, and to allow for expansion for future types of data that may be introduced. For this reason, SRI is designing a "flavor" based modeling scheme [6] that is able to represent both objects and network relationships between objects.

Each object in the database has a number of slots, each having a name and an indication of the type of information that may be stored there. The permissible types of information are not limited; in particular, the specified type can be a scalar, vector, character, or string, an array, a member of a user-defined set, a pointer to a link, or even a list of items. Objects are connected into networks with directed links; the link is also allowed to have information slots. Slot values for objects and links may be

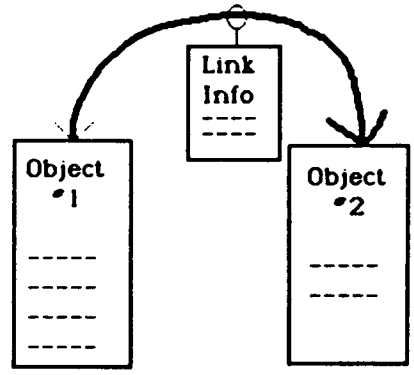


Figure 8: Objects and Network Connections

specified, or they can be defaulted to the normal value for that type. Since objects can be composed of slots of any type, and since arbitrary network structures can be built out of directed links, we anticipate that the design should be general enough for future expansion.

An example object might be the data-base entry for storing the information about a bolt. The bolt has such information slots as name, object type, absolute location, weight, visual model, graphics display model, and spatial occupancy model. It also has links to various networks such as those called relative location, obstructs, virtual enclosing object, articulation (attachment type), and is-an-assembly-of. A relative location link between the bolt and the door for instance might have the slots link-type, forward [4×4] transformation, relative-from-object, backward transformation, and relative-to-object.

In addition to the database, SRI is developing an editor to be used in entering information into the database. Besides the customary adding or deleting an object or modifying an object's slots, users will be able to define and modify object types. Both the editor and the database are expected to be delivered by the end of 1987.

5 Conclusion

This paper has presented a discussion of the current state of JPL's Telerobotic Demonstration System testbed, and the system integration work that SRI International is performing to help realize this testbed. The system was divided into autonomous mode subsystems, and additional subsystems (including Teleoperations); the workings of each subsystem by itself and how the subsystems integrate into a complete system were discussed. Finally, specific deliverables being contributed by SRI were explained. The goal of the TDS is to pull together the current state-of-the-art in teleoperations and different robotics areas. The different autonomous mode subsystems are expected to be demonstrated separately at the end of 1987; the entire system is expected to be demonstrated at the end of 1988.

References

[1] D.B. Gennery. Tracking known three-dimensional objects. In *Proc. AAAI-82*, pages 13–17, 18-20 August 1982.

[2] C.P. Fong, R.S. Dotson, and A.K. Bejczy. Distributed microcomputer control system for advanced teleoperation. In *Proc. 1986 Int. Conf. on Robotics and Automation*, IEEE, April 1986.

[3] J.K. Myers. A robotic simulator with collision detection: rcode. In *Proc. of First Annual Workshop on Robotics and Expert Systems-1985*, pages 205–213, Houston, Texas, June 1985.

[4] J. Canny. Collision detection for moving polyhedra. *IEEE Trans. on PAMI*, PAMI-8(2):200–209, March 1986.

[5] J.K. Myers and G.J. Agin. A supervisory collision-avoidance system for robot controllers. In *Robotics Research and Advanced Applications*, pages 225–232, American Society of Mechanical Engineers, Phoenix, Arizona, 1982. Republished in Robotics World, Vol. 1, No. 1, January 1983.

[6] *Symbolics Common Lisp: Language Concepts.* Symbolics, August 1986.

VISION TECHNOLOGY/ALGORITHMS FOR SPACE ROBOTICS APPLICATIONS

Kumar Krishen
Tracking and Communications Division
Code EE
NASA Johnson Space Center
Houston, TX 77058

Rui J.P. deFigueiredo
Dept. of Electrical and
Computer Engineering
Rice University
Houston, TX 77251-1892

Abstract

The thrust of automation and robotics for space applications has been proposed for increased productivity, improved reliability, increased flexibility, higher safety, and for the performance of automating time-consuming tasks, increasing productivity/performance of crew-accomplished tasks, and performing tasks beyond the capability of the crew. This paper provides a review of efforts currently in progress at the NASA/Johnson Space Center and at Rice University in the area of robotic vision. Both systems and algorithms are discussed. The evolution of future vision/sensing is projected to include the fusion of multisensors ranging from microwave to optical with multimode capability to include position, attitude, recognition, and motion parameters. The key features of the overall system design will be small size and weight, fast signal processing, robust algorithms, and accurate parameter determination. These aspects of vision/sensing will also be discussed in this paper.

I. Introduction

The Space Station, a major goal of our spoace program over the next decade, has been planned as a multipurpose facility in which missions of long duration can be conducted and supported. These missions will include science and applications, observation, technology development and demonstration, commercial laboratories and production facilities, operational activities such as servicing/maintenance, repair of satellites, support of unmanned platforms, assembly of large space systems, and as a transportation node for transfer to other orbits and planetary missions. An important technology area foreseen to increase productivity and enhance astronaut safety is *automation and robotics* (A&R). The use of A&R for the Space Station can be viewed in two major areas: teleoperated/robotic systems for servicing, maintenance, repairs, and assembly; and computerized systems to reduce the manpower requirements of planning, monitoring, diagnosis, control, and fault recovery of systems/subsystems. In addition to increase in the productivity through autonomy, A&R will result in increased operational capability, and flexibility. Robotic operations for the Space Station will involve maintenance/repair of the entire structure including various subsystems, orbiter/satellite servicing, astronaut assistance, equipment transfer, docking and berthing, inspection, remote monitoring, rocket staging, telescience, and assembly of the Station and large structures. To aid the astronauts in various tasks and replace him/her from some activities, robots must perform beyond the current state of the art by responding to a high degree of environmental uncertainity and operational flexibility. In order to accommodate various performance goals in robots, design concepts have been proposed by many organizations [1,2,3,4,5,6]. One approach advanced by the NASA/Johnson Space Center (NASA/JSC) involves using the Shuttle Remote Manipulator System (RMS) for Space Station Assembly (Figure 1). The next step in the use of A&R would be the Space Station Mobile RMS (now known as Mobile Service Center (MSC)). With the MSC tasks such as the final Station assembly, Station/satellite maintenance and repair, and routine inspections could be accomplished. The Orbital Maneuvering Vehicle (OMV) could similarly be used for the retrieval and repair services. As a final step, robots could be made autonomous and free-flying for inspection, retrieval, and repair tasks. A simplified schematic which shows the functional elements of an automated system is presented in Figure 2. One of the key elements of the system is *sensing and perception.* Its primary function is to provide information regarding the position of the object in its environment relative to the system's effector. This function involves isolation, description, identification, location, and data transmission. In a broader context a class of object properties which include geometric, mechanical, material, optical, acoustic, electric, magnetic, radioactive, chemical, and weight may be needed. The type, volume, and precision/accuracy of the data needed will depend on the nature of the task to be accomplished.

As the robotics era dawns in space, vision will provide the key sensory data needed for multifaceted intelligent operations. In general the 3D scene/object description along with location, orientation, and motion parameters will be needed. Sensor complements may include both active and passive microwave and optical with multifunction capability. The fusion of the information from these sensors, to provide accurate parameters for robots, provides by far the greatest challenge in vision. Furthermore, the compression, storage, and transmission of the information associated with multisensor capability require novel algorithms and hardware for efficient operation.

In this paper, the vision data requirements are discussed from the standpoint of various applications. A review of the advanced systems technology for space applications is provided. The progress in the area of algorithm development fo parameter estimation is summarized. Future concepts in bo sensor and algorithm development are elaborated.

II. Vision Requirements

The vision requirements for space robotics are characterized by environmental factors and tasks that the robot has to perform. The natural space environment consists of intense light and dark periods. At a nominal Space Station altitude of 270 nmi, the sunlight intensity will fluctuate between about sixty minutes of extreme brightness (13,000 footcandles) and thirty minutes of nearly darkness [7]. Furthermore, due to the absence of atmosphere, light is not diffused/scattered. Consequently, the unenhanced images have large contrast with intensity changes of the order of 10. The intense specular reflections combined with camera performance can cause bloom and Fraunhofer/Airy rings resulting in scene obscurity. Further complexity results from other objects, such as stars, moon, sun, Earth, and other satellites in the field of view(FOV). Object reflectivity can also pose problems for the vision systems. Most space systems are painted white or finished with smooth, specular materials to provide highly reflective materials. The ubiquity of white surfaces intensifies the problem of relying on photometric data for object identification/discrimination. A secondary source of concern affecting vision is the absence of gravity. For free flying and tethered objects there would be an increased number of positions and orientations in which the objects may improve due to the lack of disturbances caused by aerodynamic and gravitational forces.

Many tasks have been proposed for robotic operations in the Space Station era. The most significant tasks include assembly of space structures, maintenance and repair, inspection, and aid/retrieving of astronauts during Extra-Vehicular Activity (EVA). Assembly can include mating structures, bolting, locking, and forming joints in the structure itself. For the Space Station assembly initially the Shuttle Remote Manipulator System (RMS) controlled by astronauts in the cabin or EVA can be used. This can be followed in the later stages of assembly by the Space Station Mobile Service Center (MSC). As time proceeds, the assembly processes could involve Orbital Maneuvering Vehicle (OMV) or a free flying robot with robotic arms and various sensors for vision (Figure 3).

In the case of Space Station the maintenance and repair tasks can include structural damage, failure of systems and components, cleaning and storage of space debris, and environmental effects on the spacecraft. Inspection is the first significant part of the maintenance and repair activity. The inspection can include low velocity encounters with solar array, thermal radiator, hulls, windows, TV cameras, heaters, fluid containers, aging composite materials, printers, recorders, and door mechanisms. The unpredictable nature of maintenance and repair tasks creates a problem in the development of the capability/design of space robots. The vision capabilities must be adaptive/versatile to accommodate these uncertainties. A detailed comparison of data using computer stored scenes/identification parameters with data from distorted/damaged systems, structures, or components yields the failure or absence of the parts. Another crucial application of the space robot is the retrieving or aiding of EVA or Man Maneuvering Unit (MMU) seated astronaut. This task can be thought of as a special one belonging to a class of retriever and repair of spacecraft/orbiting objects. The robots/manipulators (such as RMS, MSC) can be operated by humans using teleoperation. The commands and other vital data are transmitted using communications systems. In more advanced concepts, the humans act as supervisors, setting schedules, tasks, and evaluating the performance of the robots. When this interaction is neglegible, the robot can act autonomously.

For a direct control teleoperator systems, the primary function of robotic vision is to provide information about the position of the object relative to the system's effector. In a direct controlled teleoperator system, some subfunctions may be allocated to the humans. In particular, the object identification can be delegated to humans. In a goal-directed teleoperator system, reliability and flexibility has to be incorporated so as to allow it to search, identify, and locate parts based on existing data base. In the absence of the vision system the required object identification and position data has to be entered manually. In general, for the autonomous robotic systems, three levels of information are needed. These levels pertain to the scene/world in which the objects are located, the objects themselves, and the specific parts of the objects. In most tasks an envelope of paramenters can be preprogrammed into the system. For example, in docking and berthing applications the robotic vision/sensing may be needed within a cone of thirty degrees to a distance of fifty meters. Beyond this spatial zone, a radar system may be used for tracking/monitoring the object motion.

The levels of information depend on the application involved. As an example in the satellite servicing area, the vision/sensing system may have to provide the necessary information to guide a robot/astrobot to a particular area, say an antenna feed. The satellite could be rotating and translating simultaneously. Furthermore, the antenna could be gimbaling with a certain motion. In this scenerio, the data would not only include a 3D dynamic description of the target/object but also its position and rotational parameters with respect to the satellite. To accomplish this, algorithms are needed for the parameter estimation. The vision/sensing instrumentation in this case would not only involve fixed field of view video systems, but laser/millimeter wave radars which could be slaved to the antenna feed motion. Doppler signal processing at microwave or optical frequencies can be used to sense moving parts within a scene. The implication on the vision/sensing systems is clear, several sensors are needed to complete the basic information needed for an autonomous robot. Clearly, for space applications the size, speed, and weight parameters are of paramount importance. Autonomous robot performance depends crucially on the vision capabilities. In certain operations, humans can be surpassed by robots due to memory and vision. Robotic vision can allow more precise measurements and faster response during time-critical situations. Other limitations of humans include fatigue, a limited spectrum, and inaccurate color and grey scale resolution. Robotic vision provides an opportunity to utilize active and passive sensors in microwave, and optical/infrared bands. Furthermore, polarization and lookangles can be optimized to accurately measure scene parameters of interest. The vision extension to shadowed and occuluded regions is important in many applications. The illumination intensity variations along with shadows can be used to determine the relative motion between the camera and the object. Structured multispectral illumination can be used to derive the 3-D description

of the target. Mathematical models coupled with real-time imagery/data can be used to derive the motion and shape parameters. Associated with the sensory data is the need for computer architectures which provide high-speed processing, parallel computations/algorithms, associative memories, and intelligence. The transfer and reduction of the sensor data requires an efficient communications subsystem [8].

Moving objects in space need to be recognized and defined in terms of their position, orientation, and velocities for proximity operations. Soft docking is an important operation for many robotic activities. A detailed analysis of docking for robotic manipulations shows many benefits can be entailed with accurate tracking sensor data. The perturbations of the target/object position and attitude can be minimized by using accurate measurements of distance/range and velocity. The relative velocity and maximum shuttle RMS tip force for the docking of Shuttle to Space Station were analyzed by Mission Planning and Analysis Division of NASA/Johnson Space Center (JSC). The capture stopping distances and the relative velocities for various forces are shown in Figure 4. With the knowledge of very accurate relative velocity (0.1 ft/sec) the perturbation force can be minimized at a particular stopping distance. The same result holds for a robot docking with a satellite, etc. Based on these considerations a laser docking sensor is now under development at JSC to provide performance goals as stated in Table 1. This is an illustrative example of the type of vision data needed in addition to the imagery. The overall scope of the vision data needed for the proposed JSC EVA Retriever project is depicted in Figure 5. A multi-sensor vision system has been proposed for this application. To achieve independence of the sunlight and to enhance accuracy, a multiple structured illumination source with controllable intensity, wavelength, polarization, field of view, and angles of incidence can be incorporated to alleviate limitations in vision systems being proposed by many organizations [9,10,11]. In the development of the space vision systems cost effectiveness, speed, small size, lightweight, high reliability and flexibility, and ease of operation must be considered.

III. Vision System Concepts/Technology

The initial use of vision for the Space Station could be to provide feedback to the human operator of the robotic structure. In the initial vision systems, NASA anticipates the use of stereo televisions for label/feature based object recognition [12]. Color and 3-dimensional imagery will be important to both telepresence and robotic vision. NASA's television program from Apollo through the Space Shuttle programs has been one of high crew and ground participation and control. Several limitations of these earlier video systems have already surfaced. One notable one was during the Solar Max Satellite repair when the shadowed surface could not be approached, and the grappling of the Solar Max was severally restricted because of limitations with the manual camera light level controls. For robotic application, several features must be added to the presently available space television system. Specifically, predictive auto focusing, programmable predictive scene control with auto zoom, gamma, and iris, automated or voice controlled pan, tilt, pointing, and scene tracking capability. Illumination affects the scene definition and is, therefore, critical to the robot's performance. The lighting technique should involve a combination of artificial lights and natural sources (Figure 6). The parameters/performance for the artificial light should include programmable wavelength, polarization, intensity, and angle of incidence, as well as the number and positioning of these light sources. The natural incident light from sun, moon, Earth, and stars should be characterized in terms of both color and intensity on the surface of the scene/object which is being viewed. For Space Station, a light intensity of 100-foot candles at the working surface is considered desirable [7]. For limiting glare, polarized filters can be used for both lights and cameras. The modes of lighting can include strobe lighting to eliminate motion blurring, infrared/ultraviolet illumination to reduce glare, and structured lighting to achieve 3-dimensional robotic vision [7].

Articulation mechanisms for cameras and lighting should incorporate flexibility. These mechanisms would form a significant part of the closed-loop control for endeffector tracking, stereo camera adjustments, and autonomous operations. In the case of stereovision, the focal length, intercamera distance, and inter-camera angle must be automatically adjusted to provide optional stereo acuity. In the case of moving objects/scenes, the vision articulation should provide the feedback to determine the trajectory to a particular point on the object/scene.

Several technology innovations are envisioned for making television's multi-features highly reliable [13]. Solid-state cameras, based on charged coupled device (CCD) or charge injection device (CID) technology with variable spatial resolution, with panels in the order of 1024×1024, can feature automatic zoom and ability to detect objects as small as 2 millimeters at a distance of 1 meter. Furthermore, the density of the sensing elements can be patterned after the eye with high resolution in the middle and low resolution in the peripheral vision. Brightness filters, automatic iris, and automatic gain control can be incorporated to allow handling of high intensity sunlight. Intensities of sun can range up to 13,000-foot candles necessitating bloom protection, large dynamic range, and protection from burn-in [7]. Similarly, sensitive night/dark vision modules with nonblooming characteristics are needed. In particular, very high quality and resolution imagery is needed to meet robotic vision requirements. In certain applications, large format cameras capable of precise image mensuration at the pixel level may be required. In some applications, design features favorable to telerobotic vision applications may be incorporated to identify and determine aspect/attitude of various objects. Object shapes, color, and markings are some of the parameters useful for this purpose. Color vision offers an additional capability for the recognition of objects. The levels/shades of colors used in robotic vision can be substantially higher than can be distinguished by the human eye. Its immediate effect on the design of the video system is added complexity, data processing, and resultant cost. Furthermore, the rate of data transmission increases significantly. Infrared cameras may be extremely beneficial in the location of objects in the dark and occluded areas. Although inherently low resolution, infrared imagery can provide gross identification of the objects.

The accuracy of television-based measurements is adversely affected by the presence of the earth, sun, moon, and/or stars directly in the field of view (FOV) of the camera system. CID video implementations can allow reduction of image blooming and removal of the limited area in the FOV corresponding to the sun, moon, and/or stars. In the case of the extended background provided by Earth, selection of appropriate operating optical wavelength for the video system can reduce or eliminate this backgrouund radiation. A wavelength of 0.94m provides an attenuation of 21.6 dB due to water absorption in the earth's atmosphere [14]. Video systems have been developed at JSC in this band for future use in space robotic applications [13].

Automation in TV operations can be incorporated by processing the imagery to determine parameters which are needed in the feeback loop. One such parameter is a particular object in the scene to be tracked as it moves. Recognition of the object, along with position of its centroid as a function time, are needed to point the TV to this object. Accurate algorithms for such parameter estimation in presence of rotation/motion and multi-object environment are presently under development at many institutions. A complementary and independent automation feature in space TV operation can use voice control. Such an implementation has been developed at JSC [15,13]. This voice control system (VCS) allows hands-off control of TV functions including: (1) monitor selection, (2) camera selection, and (3) pan, tilt, focus, iris, zoom, and scene track. Future use of voice has been projected to include the EVA astronaut. In this application, the astronaut can ask for Heads Up Display (HUD) of vital data from the Shuttle computers. The data can include system parameters, orbital parameters/location, system status, and particular subsystem data. The technology innovations for future VCS include speaker independence/user-trained, very large vocabulary, and isolated and continuous speech recognition.

The need for video data to be able to interface with digital processors/computers has given impetus to digital TV technology. For the solid state implementations using VLSI/VHSIC, recognition/prepro- cessing algorithms can be implemented on the same electronics chips making the size of these video imagers small. As this technology is rapidly moving forward, the need for handling and transmission of high data rates is becoming obvious. For a video system at 5 MHz baseband, an 8-bit digitization would generate 80 MBPS data stream. For color TV implementations and multiple systems, this bit rate can multiply significantly. For real-time processing of this data, compression techniques have been proposed which can also be implemented by innovative chip designs. The compression algorithms should be automatic and transparent to users, not destroy or discard any relevant information, and compress/ decompress data at speeds significantly higher than the associated device data transfer speeds.

Fourier optical processing offers a method of high speed parallel processing of data needed to support automation and robotics applications (Figure 7). The inherently parallel nature of optical processing, coupled with the easy and natural optical Fourier transform and the programmable masks, can obviate numerical processing for many applications. The masks are used to modulate the optical Fourier transform of the input scene. An optical retransform then allows direct detection of, say, the mathematical correlation between the viewed scene/data and the reference image/data. This scheme allows correlation or convolution computations in a rapid manner; the speed essentially controlled by the recalling of computer-memorized data and transfer of the input scene to the programmable masks. Texas Instruments, under NASA/JSC sponsorship, has fabricated an early prototype of this processor. Many improvements in the performance of these processors are envisioned for operational use. Designs are needed for Deformable Mirror Device (DMD) high spatial resolution, accuracy, and reduction of nonlinear effects caused by diffraction/scattering. The phase-only nature of these processors results in loss of correlation due to rotation and translation of the object/scene. Work is in progress at JSC to incorporate rotation-invariant filtering (directional filtering) for compensating the phase-only correlation effects.

Another rotation-invariant methodology is the use of a transformation such as the logarithmic spiral grid for picture digitization to make it to correspond to the human eye [16,17]. This spatial mapping from a high resolution imager to the input modulator in the correlator results in insensitivity to scale and rotation of a viewed object. Changes in scale and rotation of the input image become displacements in the correlation plane (Figure 8). Continued development of various mappings will result in the design of VLSI cameras whose receptor patterns are best suited to drive a subsequent optical correlator.

Another processing technology can be based on neural networks. Neural networks are patterned after the human neurons in the brain and can be termed as the learning networks. Hardware implementations of neurocomputers has already made it to commercial markets. For example, the TRW Mark III has 8100 processing elements and 417,000 interconnections. As a coprocessor to the VAX computer, it speeds operations by a hundred times. Analog electronic neurons have been assembled and connected for the analysis and recognition of acoustical patterns, including speech [18]. To recognize speech/sounds at the phoneme or diphone level, the set of primitives belonging to the phoneme is decoded such that only a neuron or nonoverlapping group of neurons fire when the sound pattern is present at the input. The output from these neurons is fed into a decoder and computer which then displays the phonetic representation of the input speech. Similarly, neural network architectures/algorithms have been proposed to provide anthromorphic framework for analysis and synthesis of learning networks for correlation, identification, and tracking applications [19]. The current technology allows 100-million processing elements along with 100-million interconnects. By late 1990's one billion neuron networks, with 10 billion interconnects, can be projected as technology in this area advances rapidly. Increases of the number of neurons on a single chip are projected to ten thousand. These analog neuro-network processors can then be directly used for video/scene recognition and mensuration. Images of space scenes/objects in certain aspects can be memorized on the neural networks. Based on this data, new aspect object views of the incoming data can be recognized using interpolation and extrapolation techniques.

For 3-D vision, range information to each pixel can be added to the video imagery. Laser scanning devices capable of giving angles/position and ranges to each pixel within the field of view provide the depth/height profile of the object. Included in these measurements can be the laser reflectance of the pixel. Two technologies of the solid-state laser vision devices currently available are those using mechanical motion of mirrored surfaces, and those that involve an inertialess change in the optical properties of the transparent medium. The latter class includes diffraction of light from an acoustically generated periodic structure. Phased-arrary solid-state scanning devices are also currently under development. These devices have the promise of providing fast, accurate, and lightweight laser vision. Another application of the laser range data can be in the automatic zoom/focus of video systems. The laser vision measurements are dependent on the intensity of reflected radiation. If coherent radiation is used in order to generate an image of the object information from the amplitudes, as well as the phases of the scattered radiation, a 3-D reconstruction of the object can be made. Such devices are known as holographic devices. The source of coherent radiation can be a solid-state laser that can, in principle, provide a resolution of the order of about 1 m. Part of the radiated beam is deflected toward the detector (Figure 9), where it interferes with the backscattered light from the object. The hologram can then be generated using known reproduction processes, or a 3-dimensional description of the object such as a Fast Fourier Transform (FFT). Several applications of holographic scanners have been discussed by Sincerbox [20]. Some of these are directly applicable to space robotics systems.

Microwave systems have been used to detect relative speed of objects and their range in many applications. Their use in space robotics applications is being studied at NASA/JSC. In particular, millimeter wave radars provide attractive performance parameters in addition to their small size. The possiblity of broader beam than laser systems makes these sensors attractive for initial acquisition of moving objects. A radar at the frequency of 100 GHz has been developed at NASA/JSC [21]. This particular system is for use on a Man Maneuverable Unit (MMU) to provide relative range and velocity to the object. The radar is designed to operate over the range of speeds from 0.1 to 2.0 fps. This type of radar operating at several carrier frequencies can be used to measure backscattering coefficients for various polarization combinations. These coefficients are object structure dependent. There is also the possibility of penetration through thermal protection and obscuration caused by nonmetallic objects. This data can be utilized in an interactive manner with that of the video systems to provide scene definition/parameters in certain situations/scenerios.

Another promising microwave technique is the time-domain imaging, the synthesis of the scattered electromagnetic field distribution over an object plane. The transmitted pulse is an impulse source offering higher instantaneous signal-to-noise ratio, higher resolution, and option for echo time gating. Technology advances in the fast pulse generators and sampling devices allow the fabrication of time-domain imagers in picosecond region with measurements and recording of both phase and amplitude of the returned/transmitted signal. The scattered signal can be formulated as the convolution of the source with the transmitter, receiver, and scatterer responses [22]. From a set of time-domain responses obtained from different viewing directions, the two-dimensional field distribution is synthesized using a technique similar to the one in tomography [23]. The resultant image closely resembles the object geometry [22]. Thus, time-domain impulse imaging is another tool in extracting physical information about the object.

Space telerobotic and autonomous robotic operations have to be monitored and controlled remotely without the availability of hardline power and data services. The robots/end effectors must reach small, crowded, or restricted space. The communications system on the robot has to be able to transmit multiple channels of high-quality video and high rate data from other sensors. In most implementations, it should be able to receive high-rate data from the control and monitoring station. The coverage for the robot/end effector should be spherical without blockage/interference from the system. Furthermore, the time delays through processors, prime power, size, and weight should be minimized. In view of these desired performance goals, higher wavelengths would be attractive for communication systems. The bands could include millimeter wave and optical/infrared. Infrared/Laser communications offer unique advantages which are being explored at JSC. The design features include multi-access, packetized, high-rate, broad-beam links.

This section has dwelt on a broad set of concepts for system implementations for robotic vision. Active and passive sensors in microwave, optical and infrared bands, along with high-rate communications systems, are needed for various vision applications. Superconductivity devices/systems will have a significant impact on the vision systems design and performance. Examples of systems benefiting from this technology would be: (1) millimeter wave high efficiency distributed antennas with broadband and large beamwidth performance, (2) microwave power switches, networks, and distribution circuits resulting in substantial reduction in power loss, and increase in bandwidth and sensitivity, (3) development of optical and infrared detector cameras for low level/dark sensing, and (4) development of programmable signal processors, neuro-networks for speech and scene recognition, and communications monitoring/ control processors.

IV. Information Processing Algorithms

The processing of video data at various information levels, spanning from the data level to the intelligence level, is driven by *algorithms*. As mentioned earlier, the result of this processing is to provide a human operator or a robot with the parameters needed to control a mechanism. In other instances the processing leads to a high level description/interpretation of the observed scene for consumption by a human or robotic supervisor.

In a given application the set of vision algorithms may be grouped into three stages, depicted in Figure 10 and explained in detail in [24]. These three stages provide a meaningful rationale for a *CAD-based vision* under current development at Rice.

(1) The *first stage* is an Image Preprocessing Stage which sends the noisy pixel data into a set of *labeled features*. Typically, there are three types of features, namely, *corn*

C-6

vertices, *edges*, and *shaded regions* corresponding to surface patches on the imaged object. In a given application one may use any one of these types of features or a combination of some or all of them. For example, a common set of features is the *wireframe* W_f of an image f of an object O_m. The wireframe W_f consists of the set of edges and corners present in the image f. The wireframe of the image of a cube is shown in Figure 11a. >From now on, we will assume that the output of the IPS is a labeled wireframe. A typical sequence of algorithms constituting the IPS would be algorithms for Gaussian filtering, Sobel operation, thresholding, median filtering, and contour thinning. Other sequences of image processing algorithms can be selected, depending on the type of the image data used. The wireframe of a mock-up of the Space Shuttle of Figure 12 extracted using these operations is shown in Figure 13.

(2) The *second stage* is what we call a *Symbolic transformer* (ST) (see Figure 10) which maps the labeled wireframe W_f into an attributed graph $G(W_f)$. One way of converting W_f into $G(W_f)$ is to map each face F_i (the region enclosed by a mesh in the wireframe) into the node N_i of the attributed graph $G(W_f)$ and the edge defining the boundary between two adjoining faces, say F_i and F_j, into the link l_{ij} of the graph. Referring to Figure 11, the graph $G(W_f)$ corresponding to the wireframe W_f of the visible surface of the cube of Figure 11(a) is mapped into the subgraph within the dotted curve (consisting of the nodes N_1, N_2, and N_3 and the links joining these nodes to each other) of the graph of Figure 11(b). The whole graph of this figure corresponds to the wireframe of the entire surface (visible and occluded) of the cube. In such a symbolic representation, we assign an attribute (feature) vector to each node and each link in the graph. Thus, let an m-vector $I^i = col\,(I^i_1, ..., I^i_m)$ represent a set of attributes (features) associated with the face F_i which are *invariant* under 3D translations, scalings, and rotations. Examples of such I^i are the sets of numbers expressing the Gaussian curvature or mean curvature of F_i. We call the attributed graph $\tilde{G}(\eta)$ obtained from $G(\eta)$ by assigning to the nodes N_i, $i = 1, ..., n$, respectively, the attribute vectors I^i, the FIAG (Feature-Invariants/Attributed-Graph) [25] representation of the object η. A special case of this representation is the MIAG (Moment-Invariants/Attributed-Graph) representation of polyhedral objects proposed in [26], in which I^i constitutes a set of 2D moment invariants of F_i (these being invariant with regard to 3D translations, scalings, and rotations).

(3) The *third* and final *stage* (see Figure 10) is a set of High-Level-Processors (HLP's) that map the attributed graph into appropriate symbols or vectors giving the information that the vision system is required to provide to the human operator or robot. Thus, in Figure 10, HLP1 identifies the object being viewed as being the object O_m of the objects present in the computer library. In other words, HLP1 is embodied by an *identification algorithm*. HLP2 and HLP3 constitute implementations of algorithms for determinant of the position and orientation of the object respectively, etc.

Having outlined the general framework for the various algorithms constituting the vision system, we now focus attention on some of the algorithms making up the individual HLP's mentioned above.

(a) Object Identification/Recognition. The Moment-Invariants/Attributed-Graph (MIAG) algorithm [26] for recognition of 3D objects from a single picture has been successfully developed and tested [27]. The algorithm works for polyhedral objects, and its generalization for nonpolyhedral objects has been indicated [25]. Each face of a polyhedron can be considered to be a rigid planar patch (RPP). Motion of the object can be considered to be motion of its constituent RPP's. In the case of parallel projection, if an RPP undergoes rigid body motion in 3D, its image undergoes affine transformations. So the method which tries to identify an object in 3D motion should use features of images which remain invariant under affine transformations. General moment invariants introduced in [26] are such features. These are invariants of 2D (rigid planar patch) moments which remain invariant under 3D translations, rotations, and scalings. Identification of an object is achieved by matching the attributed graph of its image (see Fig. 11(a)) to a subgraph of one of the graphs corresponding to the models stored in the computer library. The algorithm matches a pair of nodes by comparing the Euclidian distance between their feature vectors. Thus, if $I = (I_1, I_2, I_3, I_4)$ is the feature vector of a node consisting of four moment invariants of its corresponding face, and $I' = (I'_1, I'_2, I'_3, I'_4)$ the feature vector of the node to which it is being matched, the distance between them is taken to be

$$d = \sqrt{\rho_1(I_1 - I'_1)^2 + \rho_2(I_2 - I'_2)^2 + \rho_3(I_3 - I'_3)^2 + \rho_4(I_4 - I'_4)^2} \quad (1)$$

where ρ_i's are appropriate weighting factors. The driver algorithm arbitrarily picks a node N_j in the image attributed graph; then it looks for a node O_j in the model graph with the same feature vector. If matched, these nodes are marked as a pair, and an adequate node in the image graph is chosen, and the nodes adjacent to O_j are scanned to see if it matches one of them. In practice, after a few node matchings, a unique identification is achieved.

(b) Motion Parameter Estimation. Using appropriate camera calibration, all the motion parameters (position, velocity, attitude, and attitude rate) except for a scaling factor, can be determined by means of a single high precision camera. For this purpose, there are basically two model-based methods available: One, based on the contraction of the moment tensors of a surface patch of the model and its image, determines the attitudes vector $\underline{\theta}$ (raw, pitch, and yaw) and attitude velocity $\dot{\underline{\theta}}$. (See [25] and [26] for details.) The other, based on the correspondence (assumed known) of eight points on the image f and eight points on the model η^i (assumed located and oriented in a standard position), yields all the motion parameters except for a scaling parmeter. This second method has been extensively discussed by Longuet-Higgins [28], Tsai and Huang [29], Haralick, et al. [30], and others, and recently extended to the case in which both the object and camera are moving by Fotedar, et al. [31].

(c) CAD-Based Vision. What we have described above constitutes a framework for CAD-based vision (see [24] for details). The current CAD-based systems are driven by 3D geometric modeling procedures originally developed for the representation and manipulation of objects in a design or computer graphics environment [32-34]. The system under development at Rice, based on the representations described

above, will be fully compatible with the requirements of a vision system.

There are three other areas of algorithmic development related to vision which are of special interest and the work on which is being actively pursued. These are described below.

(1) *Shape Extraction Based on Illumination.* As pointed out earlier, the fact that near vacuum prevails in space scenarios makes the scattering of light by a surface strongly dependent on the surface properties. Thus, by using appropriate mathematical models for the surface and for the 3D illumination conditions, it is possible to design algorithms that precisely determine the shape of the surface by the shadowing caused by illumination as well as any changes that have occurred on the surface conditions.

We note that a variety of methods have been developed for extracting shape based on camera data, each working under a different set of conditions and using different clues for reconstructing the surface. Thus, *Stereo Vision* uses the disparity between two images of the same object from two cameras to reconstruct the surface. Methods based on *Structured Lighting* project a known pattern of light on the surface and reconstruct the surface by looking at the distortion of this pattern. *Shape from Shading* is based on the premise that surfaces reflect different light intensities depending on the relative orientation of the surface to the light source and the observer. In principle, knowing the form of this dependence and the amount of light actually reflected back, the surface orientation can be calculated (see [35] and references therein).

Two methods have emerged for extracting shape from shading. One is called the proper *"Shape from Shading"* method while the other is termed *"Photometric Stereo."* Classical shape from shading techniques reconstruct the object from a single photograph of the object when the light source is placed in a known position. Photometric stereo involves reconstructing the object from multiple images of the object taken by moving the light source to different positions, while the position of the camera remains fixed. Some of the advantages that these methods offer are *high resolution* surface reconstruction as typically needed on assembly line operations. Handling of new parts, object testing for tolerance, estimating and repairing structural damage are some tasks which need a high resolution surface reconstruction front end. Another possiblity is the design of a front end visual system for feeding in surface models of real world objects to a CAD system.

Stereo vision and structured lighting methods have an inherent matching problem (used in generating the disparity map and the line matching) that is as yet unsolved. This problem is absent in shape from shading methods. Furthermore, in comparison with the structured lighting methods, shape from shading methods offer the advantage that the whole surface of interest is imaged. No shadows are willfully projected on it.

In shape from shading algorithms, the characteristic strip expansion methods [35] have several shortcomings, including sensitivity to measurement noise and a tendency of adjacent characteristic strips to cross over each other, due to accumulation of small numerical errors. Finally, the procedure is not amenable to implementation in parallel form. The variational method [35] that uses an object's occluding boundaries as cues to the recovery of its shape from shading alleviates these limitations. The blending of concepts from variational calculus with those from the best approximation theory can lead to spline-based solutions for the gradient functions determining the local surface shape orientation, as obtained in [36]. Research is also under way at Rice [37] to investigate certain aspects of Photometric Stereo such as completeness of illumination, optimal light placement, and robustness with respect to noise.

(2) *Shape Extraction from Sparse Range Data.* Our second set of algorithms for the extraction of shape of 3D objects are the ones based on sparse range data. A new methodology for surface reconstruction from such data was recently developed by Kehtarnavaz, et al. [38]. Such a reconstruction is formulated in terms of three separate subproblems: (i) 3D contour segmentation, (ii) segment matching, and (iii) surface patch formation. This framework is based on a *syntactic/semantic criterion* which incorporates the shapes of the contours in creating the surface. First, the contours are divided into sets of 3D curve segments in order to distinguish local shapes or substructures in the contours. Then, the curve segments are found on adjoining contours with similar shape characteristics. Finally, parametric surface patches are formed between the matched pairs of curve segments on adjoining contours by appropriately blending them. Typical reconstruction obtained by these results are illustrated in Figures 14 and 15.

(3) *Shape Extraction by Sensor Fusion.* Although radar scattering cross-sections alone cannot provide a complete description of a scattering surface, they are very useful when used to complement optical images, providing information in those regions of the object where a camera is blind due to phenomena like specular reflection.

The specular point on a curved surface is a point at which the angle of reflectance equals the angle of incidence. In traditional shading models, highlights occur at these points. In space, these highlights are so disproportionately intense that they tend to obscure the surrounding details of the surface. This phenomenon can be traced to two causes: the *Airy disk* , and *blooming*.

The *Airy disk* , or ring, is an optical term for the first diffraction fringe surrounding the image of a point source transmitted through an aperture. Because lenses constitute finite apertures, these rings are present to some degree in all imaging systems [39]. Usually, when dealing with incoherent light emanating from curved objects, the Airy disks of a distribution of point sources tend to cancel each other out and are not visible in the resulting image. However, intense specular reflections become point sources which are orders of magnitude more intense than the surrounding reflections. The resulting diffraction fringe is highly visible and can wipe out the shading information in adjacent areas of the image.

Blooming occurs at points of high intensity in a television image. If the distribution of intensities is relatively even, this phenomenon is not a problem. If specular points occur whose intensities contrast sharply with their surroundings, the effect is noticeable. The anomalously high grid voltage in the camera cathode ray tube causes the electron beam to spread. The result is specular points which are smeared over several pixels. Blooming can obscure small features surrounding

specular points in the image of a highly reflective surface.

Space images also suffer from *indistinct edges*. Because of the complete lack of illumination on the shadow side of space objects, their edges are invisible against the dark background. This can result in false edges for curved objects, or the absence of one or more edges on polyhedrons.

The sensor fusion algorithm under development at Rice [40] reconstructs 3D space objects (whose images may be degraded as described above) given observations taken by a microwave radar system from a solitary remote point. Since microwave or millimeter-wave radar systems are currently found on a variety of space vehicles, radar scattering information would seem to be a logical addition to a space robot's sensory data. The unknown portion of the scattering surface is parametrically approximated with splines so that the microwave scattering equations can be used to derive the unknown surface. In this way the radar cross-sections are used to reconstruct those portions of optical images which are destroyed by high intensity specular reflections (see Figure 1). Edges which are lost in shadows can be inserted in a similar fashion. The solution procedure is an *iterative non-linear least squares* algorithm [41], using the incomplete optical surface to provide a first approximation to the actual surface parameters. A surface model is generated at each step in the algorithm and approximations to the co- and cross-polarized scattering cross-sections are computed from this model. A Physical-Optics approximation to the Jacobian is then used to update the unknown surface parameters for the next iteration. When the best possible surface is obtained, the least-squares algorithm is terminated and the new surface, with the degraded portions filled in, may be passed back to the optical shape-from-intensity algorithm for further refinement. Thus, we are fusing the optical image sensors with polarized microwave radar cross-sections to arrive at a target object characterization which is more complete than either of those derived from the image or radar separately.

V. Proposed Future Developments

As was mentioned earlier, the interaction of natural light with the objects in space has to be accounted for in the vision algorithms. Furthermore, artificial light(s) arrangements have to be developed which can provide structured (known distribution) light across the object. The pronounced shadows and specular points due to the vacuum and smooth parts of the object, provide a large dynamic range of the reflected/scattered signal. The intensity changes can be in the 10 range. The addition of artificial illumination provides the opportunity to control intensity, wavelength, polarization, and orientation with attendant advantages of increased recognition. Additionally, color will be another discriminant involved in the recognition. Analytical studies in these areas should lead to the design of illumination systems for space applications.

The use of laser vision and microwave scattering instruments creates another area of future development. Fast scanning/holographic lasers provide a depth perception of objects, which is quite complex. This depth data can be utilized to iteratively provide a 3-D image of the object by weighting video-aquired image data appropriately. These weights will depend on the surface curvatures as they project in the incidence direction of the laser beam. Both empirical and analytical studies are needed. The microwave backscattering can provide another independent set of data. The shape of certain objects can be directly deduced from this data. In many inspection tasks in which a nonmetallic shielding has obscured the view, such sensing will be mandatory. In other situations, the microwave data can be iteratively used with that of TV to arrive at more definitive description of the object. At the expense of complexity, doppler processing of microwave and laser data can be used to discriminate moving parts of a distant object. The advantage of such a vision is that it is independent of sunlight and it provides a direct measure of range and relative velocity of various parts of the object.

Another area of endeavor should be time-domain imaging. A sharp pulse transmitted yields a unique description of the object. This time domain reflectometery is evolving rapidly. Another mode of the system can utilize reflectivity data in the near-field. These techniques have not been explored for the robotic vision applications.

Finally, further research and development is needed in the area of multisensor coordination and fusion. The recognition algorithms are to be extended to include interrelating data from several cameras, laser scanners/holographic systems, and microwave sensors. These algorithms should include motion, rotation, and object changes as functions of space and time. In addition to this, "environmental" data pertaining to the events/objects and their status, has to be included. These aspects, along with rational models, incorporate expert and artificial intelligence techniques in the scene analysis. The goal should be a multisensor, multimode vision system capable of autonomous operation and self-calibration.

VI. Conclusions

This paper is aimed at providing a review of some of the efforts in progress at NASA/JSC and Rice University. The design and development of a vision systems for space applications needs several considerations which make them different compared to those used in ground applications. The concerns for space unique vision systems have been elaborated. Several efforts which need to be undertaken have been discussed. Considerable work has to be accomplished in order to provide robust, lightweight, small size, and autonomous vision systems for specific space applications.

Acknowledgement

The authors wish to thank Mr. R.S. Sawyer of NASA/Johnson Space Center for his encouragement and continued support of the research reported in this paper.

REFERENCES

1. Space Station Automation Study, Automation Requirements Derived from Space Manufacturing Concepts, Volume I and Volume II, General Electric, November 27, 1984.
2. Automation Study for Space Station Subsystems and Mission Ground Support – Final Report, Hughes Aircraft Company, November 1984.

3. Space Station Automation Study, Final Report, Volume I and Volume II, Martin Marietta, November 1984.
4. Space Station Automation and Robotics Study, Final Report, Boeing Aerospace Company, November 1984.
5. Satellite Services System Analysis Study, Executive Summary, Final Briefing, Grumman Aerospace Corporation, July 22, 1981.
6. NASA Space Station Automation: AI Based Technology Review, Executive Summary, SRI International, March 1985.
7. Bronez, Mark A., Vision Concepts for Space-Based, Tech Report SSS 86-0165, Rockwell International, Space Station Systems Division, Downey, California, January, 1987.
8. Will, Ralph W., and Sliwa, Nancy O., Design for a Goal-Oriented Telerobotic System, Presented at the AIAA Guidance, Navigation, and Control Conference, Williamsburg, Virginia, August, 1986.
9. Cohen, Fernand S., 3-D Object Recognition From a Single Image, SPIE Proceedings Volume 521 Intelligent Robots and Computer Vision, 1984.
10. Jurica, K.E., Kohn, W., and Lai, D., A Variable Configuration Controller for a Multipurpose Articulated End Effector, AIAA/NASA Symposium on Automation, Robotics and Advanced Computing for the National Space Program, Washington, D.C., September, 1985.
11. Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy, Progress Report 3, NASA Advanced Technology Advisory Committee, Washington, D.C., September, 1986.
12. Holcomb, L.B., Larsen, R., and Montemerlo, M., "The NASA Automation and Robotics Research Program," AIAA/NASA Symposium on Automation, Robotics, and Advanced Computing for the National Space Program, Washington, D.C, September 4-6, 1985.
13. Krishen, K., deFigueiredo, R.J.P., and Graham, O., Robotic Vision/Sensing for Space Applications, Proceedings of the 1987 IEEE Int'l Conf. on Robotics and Automation, Raleigh, N.C., March, 1987.
14. Lipoma, P.C., and Walton, D.P., "A Two-Dimensional Near-Infrared Tracking System," SPIE's Cambridge Symposium on Optical and Electro-Optical Engineering, Cambridge, Mass., September 15-20, 1985.
15. Jordan, William T., "Voice Controlled Closed Circuit Television for the Space Shuttle Orbiter," The Official Proceedings of Speech Tech. '85, New York, April 22-24, 1985.
16. Weiman, C.F.R. and Chaikin, G.M., Logarithmic Spiral Grids for Image Processing and Display, Computer Graphics and Image Processing, Volume II, No. 3, November, 1979.
17. Messner, R.A., and Szu, H.H., Simultaneous Image Processing and Feature Extraction for Two-dimensional Non-uniform Sensors, Proceedings of SPIE Conference on Intelligent Robots: Third International Conference on Robot Vision and Sensory Controls, Cambridge, Mass., November, 1983.
18. Mueller, P., and Lazzaro, J., A Machine for Neural Computation of Acoustical Patterns with Application to Real Time Speech Recognition, American Institute of Physics, Conference Proceedings 157, Neural Networks for Computing, Snowbird, Utah (edited by John Denkar), 1986.
19. Farhat, N.H., Miyahara, S., and Lee, K.S., Optical Analog of Two-Dimensional Neural Networks and Their Application in Recognition of Radar Targets, American Institute of Physics, Conference Proceedings 157, Neural Networks for Computing, Snowbird, Utah (edited by John Denkar), 1986.
20. Marshall, Gerald F. (Ed.), "Laser Beam Scanning," Marcel Dekker, Inc., New York, 1985.
21. Lichtenberg, C.L., "Man Maneuverable Unit Millimeter-Wave (100GHz) Developmental Radar – Design and Test Results Report," NASA/Johnson Space Center, Houston, Texas, 1986.
22. Yeung, W.K., and Evans, S., Time-Domain Microwave Target Imaging, IEE Proceedings, Volume 132, Pt. H, No. 6, October, 1985.
23. Lewitt, Robert M., Reconstruction Algorithm: Transform Methods, Proceedings of the IEEE, Volume 71, No. 3, March, 1983.
24. deFigueiredo, R.J.P., "A Framework for Automation of 3D Machine Vision," Texas Instruments AI Industrial Automation Call for Papers, 1987 AAAI Meeting, Seattle, WA, July 1987; extended version to appear in the *TI Journal*.
25. Markandey, V., Tagare, H., and deFigueiredo, R.J.P., "A Technique for 3D Robot Vision for Space Applications," Proc. of Space Telerobotics Workshop, JPL, Pasadena, CA, Jan. 20-22, 1987.
26. Bamieh, B., and deFigueiredo, R.J.P., "A General Moment-Invariants/Attributed-Graph Method for 3D Object Recognition from a Single Image," IEEE Journal of Robotics and Automation, vol. RA-2, No. 1, pp. 31-41, March 1986.
27. deFigueiredo, R.J.P., Bamieh, B.A., Fotedar, S., Hack, E., Trahan, K., and Wu, C., "Demonstration of a Methodology for Machine Recognition and Attitude Determination of a 3D Object from a Single TV Picture Frame," Tech. Report EE8520, Dept. of Electrical and Computer Engineering, Rice University, December 1985.
28. Longuet-Higgins, H.C., "A Computer Algorithm for Reconstructing a Scene from Two Projections," Nature, vol. 293, pp. 133-135, Sept. 1981.
29. Tsai, R.Y., and Huang, T.S., "Uniqueness and Estimation of Three-Dimensional Motion Parameters of Rigid Objects with Curved Surfaces," IEEE Trans. Pattern Anal. Machine Intell., vol. PAMI-6, no. 1, January 1984, pp. 13-27.
30. Zhuang, X., Huang, T.S., and Haralick, R.M., "From Two-View Motion Equations to Three-Dimensional Motion Parameters and Surface Structure: A Direct and Stable Algorithm," Proc. IEEE Int. Conf. on Robotics and Automation, San Francisco, CA, April 7-10, 1986, pp. 621-626.
31. Fotedar, S., and deFigueiredo, R.J.P., "Determination of Motion Parameters of a Moving Object from Moving Camera Data," Rice University, Tech. Report EE8708, April 1987.
32. Okino, N., and Kubo, H., "Geometric Modeling in CAD/CAM," Infor. Proc. Society of Japan, vol. 21, no. 7, 1980, pp. 725-33.
33. Mortenson, M.E., Geometric Modeling, John Wiley & Sons, 1985.
34. Encarnacao, J., and Schlechtendahl, E.G., Computer Aided Design, Springer-Verlag, 1983.
35. Horn, B.K.P., Robot Vision, McGraw-Hill, NY, 1986.
36. deFigueiredo, R.J.P., and Markandey, V., "Recovering Shape of Space Objects from the Shadowing Caused by Illumination," Tech. Report EE8608, Rice University, Sept. 1986.
37. Tagare, H.D., Private Communication.
38. Kehtarnavaz, N., and deFigueiredo, R.J.P., "A Novel Surface Reconstruction Framework from 3D Contours," Proc. of SPIE's Cambridge Symposium on Advances in Intelligent Robotic Systems, Cambridge, MA, October 1986; detailed version to appear in Computer Vision, Graphics, and Image Processing.
39. Nussbaum, A., and Phillips, R.A., Contemporary Optics for Scientists and Engineers, Prentice-Hall, New Jersey, 1976, p. 222.
40. Shaw, S.W., Krishen, K., and deFigueiredo, R.J.P., "Microwave and Video Sensor Fusion for Shape Extraction of Space Objects," Tech. Report EE8706, Rice University.
41. Dennis, J.E., and Schnabel, R.B., Numerical Methods for Unconstrained Optimization and Nonlinear Equations, Prentice-Hall, New Jersey, 1983, p. 229.

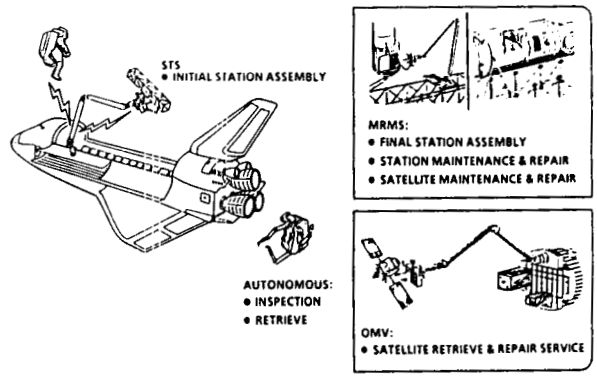


FIGURE 1. CONCEPTUAL SPACE STATION ROBOTICS/AUTOMATION

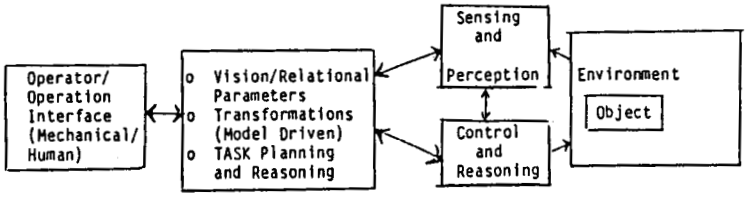


FIGURE 2.

FUNCTIONAL ELEMENTS OF TELEOPERATOR/AUTONOMOUS SYSTEM

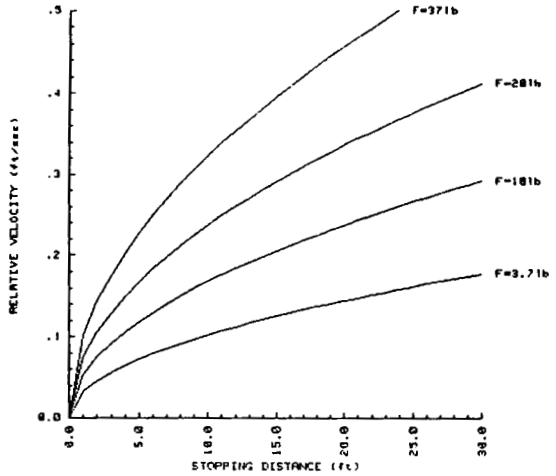


FIGURE 4.

CAPTURE STOPPING DISTANCES AND VELOCITIES FOR VARIOUS FORCES

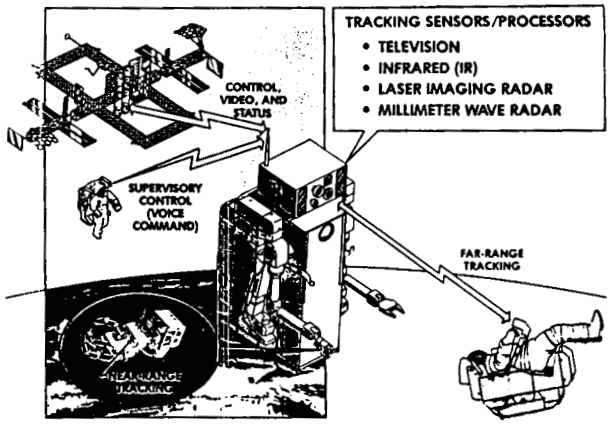
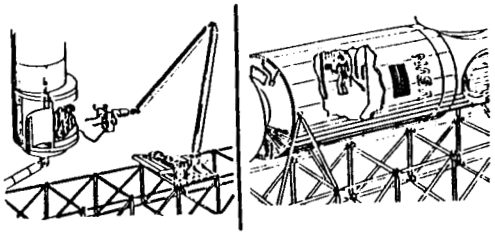


FIGURE 5. NASA/JSC EVA RETRIEVER VISION SYSTEM CONCEPT

SERVICE ON SPACE STATION CONCEPT



FREE FLYER SERVICE CONCEPT

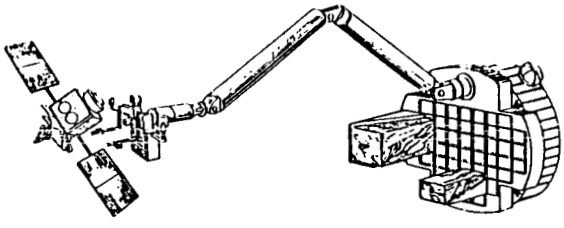


FIGURE 3. ROBOTICS SERVICING CONCEPTS

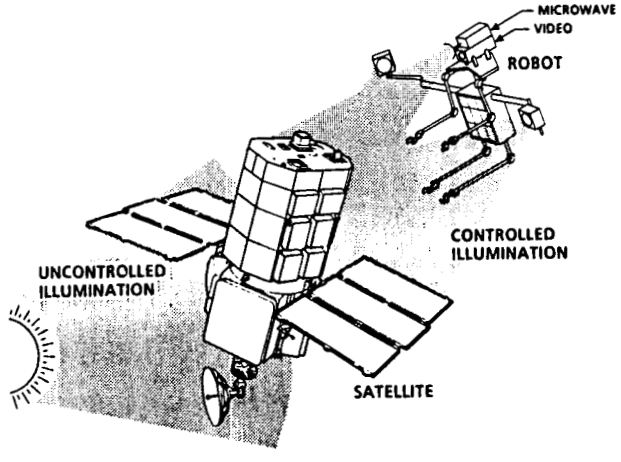


FIGURE 6.

SPACE ILLUMINATION FOR SHAPE DEFINITION/IDENTIFICATION

PARAMETER	LIMITS	ACCURACY (1σ)
RANGE (R)	0-1 KM (3280 FT)	.01 R; 2.5 MM ≤ 10 M
RANGE RATE	±3 M/S (± 10 FT/S)	.0001 R/S; 3MM/S ≤ 30M
POINTING	±π/2 RAD (±90°)	
BEARING ANGLE	±.2 RAD (±10°)	3 MRAD (.2°)
BEARING ANGLE RATE	±20 MRAD/S (±1°/S)	.03 MRAD/S (.002°/S)
ATTITUDE (P,Y)	±.5 RAD (±28°)	7 MRAD (.3°) } AT R ≤ 100 FT
ATTITUDE (R)	±π RAD (±180°)	7 MRAD (.3°) } AT R ≤ 100 FT
ATTITUDE RATE	±20 MRAD/S (±1°/S)	.03 MRAD/S (.002°/S) } AT R ≤ 100 FT
R, Ṙ OUTPUT DATA RATE	1 Hz	
ANGLE OUTPUT DATA RATE	3.125 Hz	

TABLE 1. LASER DOCKING SYSTEM PERFORMANCE GOALS

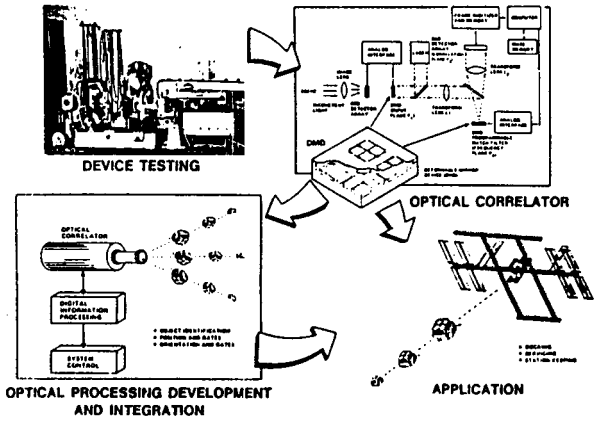


FIGURE 7.

OPTICAL PROCESSING FOR CONTROL APPLICATIONS

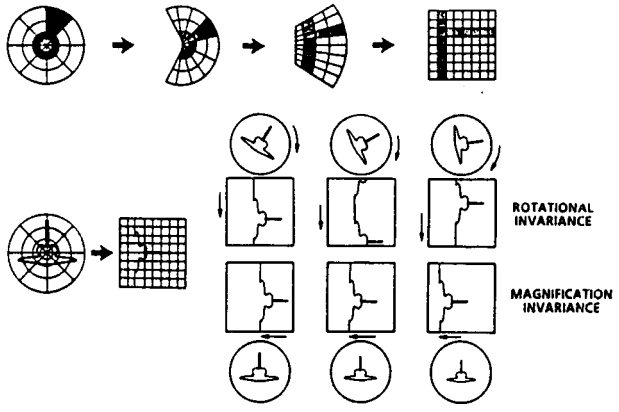


FIGURE 8.

COORDINATE TRANSFORMATION/MAPPING FOR ROBOTIC VISION

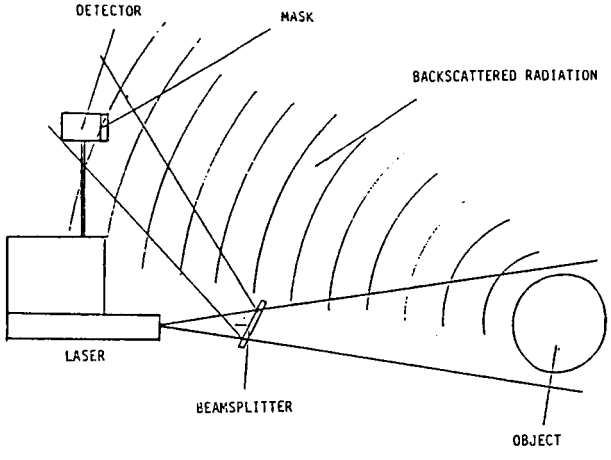
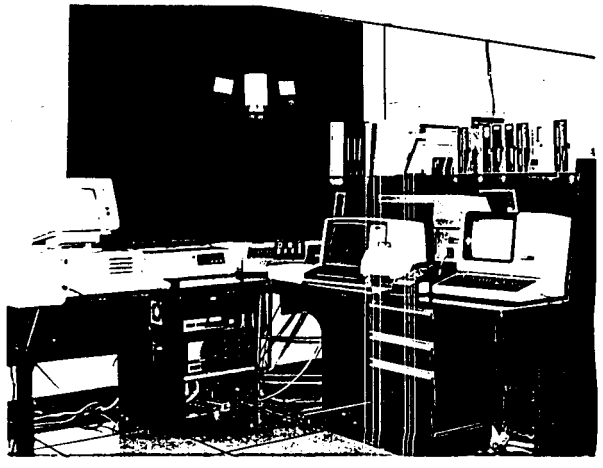
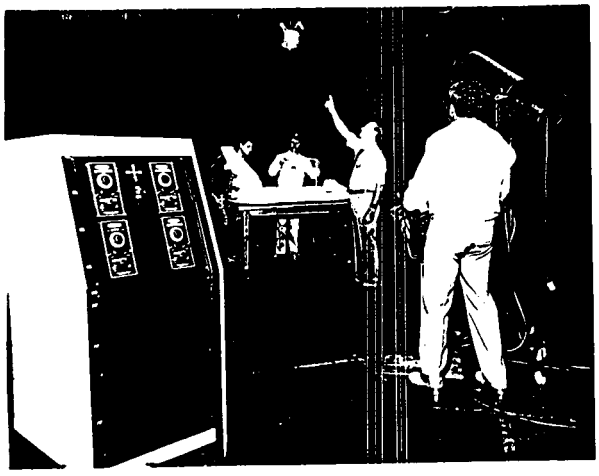


FIGURE 9.

CONCEPTUAL SCHEME FOR 3-D IMAGE PROCESSING BASED ON HOLOGRAPHY



TESTBED FOR SHAPE FROM SHADING ALGORITHM VERIFICATIO



SMART TELEVISION TESTBED

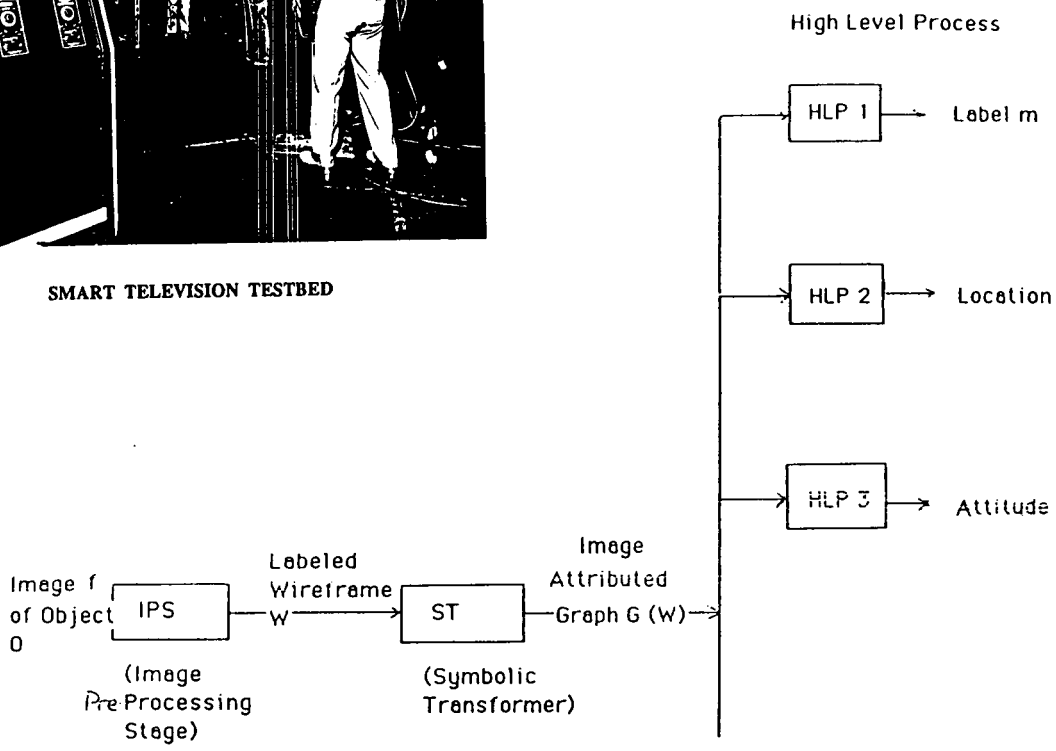
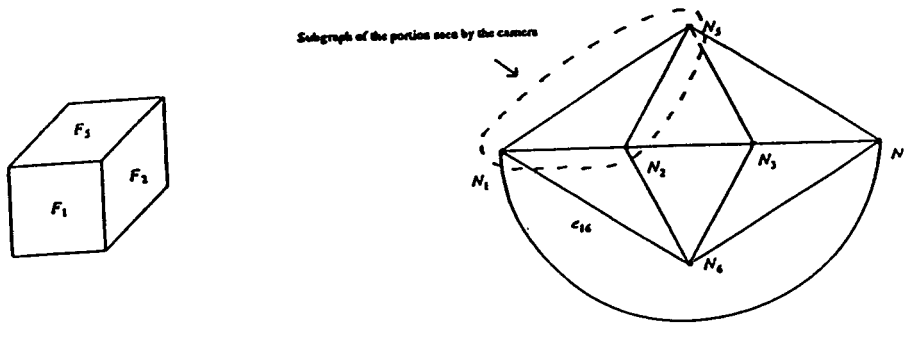


FIGURE 10. THE THREE STAGES OF THE VISION SYSTEM DESCRIBED IN THE TEXT



(a) Object (b) Attributed Graph

FIGURE 11. THE WIREFRAME OF A CUBE AND ITS ATTRIBUTED GRAPH REPRESENTATIVE

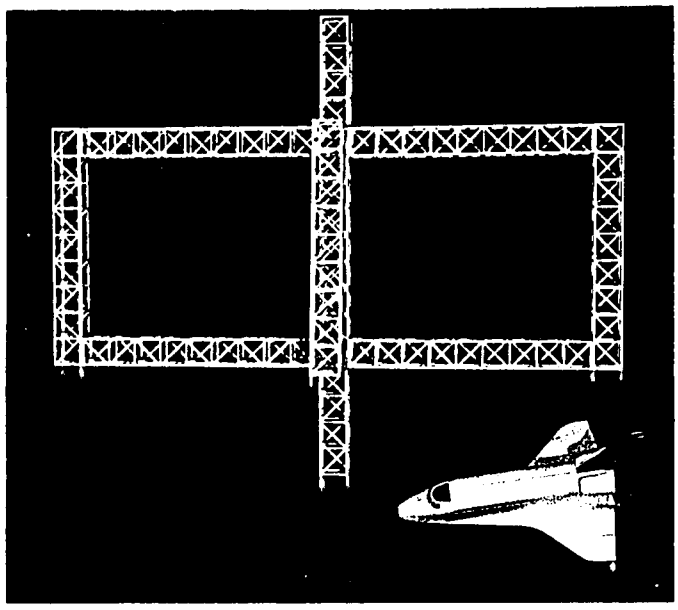
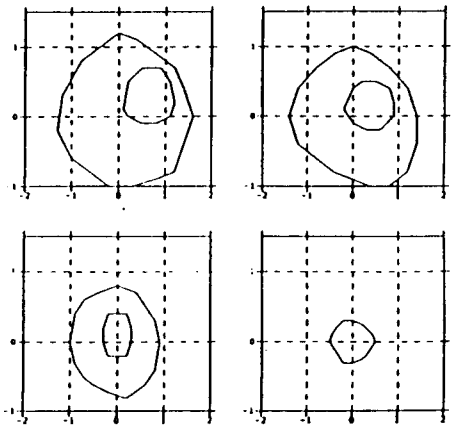


FIGURE 12.

LABORATORY MODELS OF THE SPACE SHUTTLE AND PART OF THE SPACE STATION



EDGES OF A TYPICAL SET OF LEFT VENTRICULAR PET

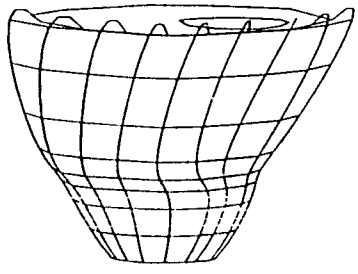


FIGURE 14.

(POSITRON-EMISSION-TOMOGRAPHY) SLICE AND THEIR RECONSTRUCTION BASED ON CARDINAL SPLINE BLENDING FUNCTIONS

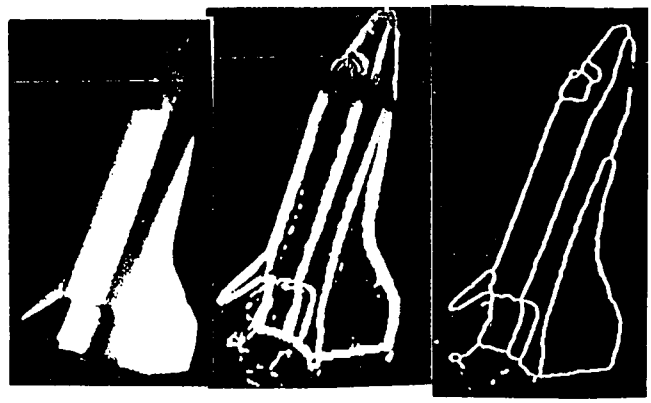


FIGURE 13.

PRE-PROCESSING OF A PICTURE OF THE SPACE SHUTTLE MODEL

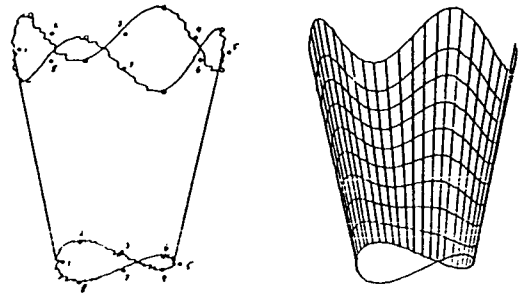


FIGURE 15. BOUNDARIES OF A BROKEN GLASS (NOISE ADDED) AND ITS SURFACE RECONSTRUCTION

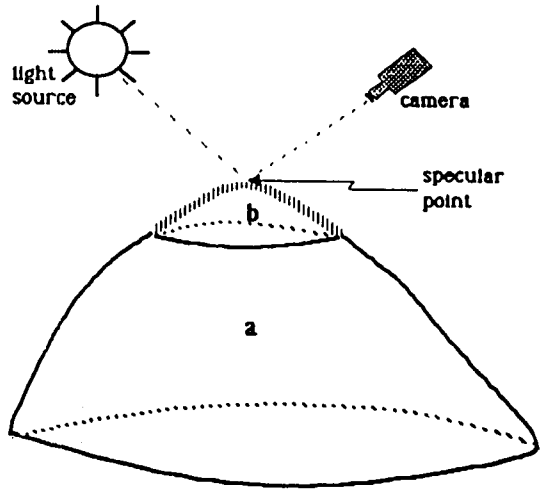


FIGURE 16. SURFACE RECONSTRUCTION BASED ON SENSOR FUSION (SURFACE A IS TO BE RECONSTRUCTED FROM IMAGE DATA WHILE SURFACE B FROM RADAR DATA)

Machine Vision by Optical Correlation, Programmable Spatial Light Modulators, and Real-Time Image Warping

(Paper not provided by publication date)

N88-17268

AN OPTIMAL RESOLVED RATE LAW FOR KINEMATICALLY REDUNDANT MANIPULATORS

McDonnell Douglas Astronautics Co. - Engineering Services
16550 Space Center Blvd., Houston, Tx. 77062

B. J. Bourgeois

ABSTRACT

The resolved rate law for a manipulator provides the instantaneous joint rates required to satisfy a given instantaneous hand motion. When the joint space has more degrees of freedom than the task space the manipulator is kinematically redundant and the kinematic rate equations are underdetermined. These equations can be locally optimized, but the resulting pseudo-inverse solution has been found to cause large joint rates in some cases. In this paper a weighting matrix in the locally optimized (pseudo-inverse) solution is dynamically adjusted to control the joint motion as desired. Joint reach limit avoidance is demonstrated in a kinematically redundant planar arm model. The treatment is applicable to redundant manipulators with any number of revolute joints and to non-planar manipulators.

INTRODUCTION

The resolved rate law for a manipulator converts the instantaneous hand rates into instantaneous joint rates [1]. This allows the joints to be simultaneously commanded to move the hand with a desired instantaneous translational and rotational velocity. The space that the hand moves in is called the task space [2], and is usually composed of 6 degrees of freedom. The space mapped out by the joint angles is called the joint space. The mathematical relationship between the task space and the joint space defines the resolved rate law for the manipulator.

The kinematic equations express the hand state in terms of the manipulator joint angles. The matrix that results from differentiating the kinematic equations is the Jacobian matrix for the manipulator. The kinematic equations for most manipulators are very nonlinear and generally cannot be inverted to solve for the joint angles in terms of the hand parameters [2].

The Shuttle Remote Manipulator System (SRMS) has six joints and the end-effector operates in a six degree of freedom space (three spatial and three angular). This is a convenient design because the number of degrees of freedom in the joint space and in the task space are the same; the Jacobian matrix is square and can be easily inverted, except in specific configurations where the Jacobian matrix is singular. When these singularities are avoided, the inverted Jacobian is a valid resolved rate law for the SRMS because it transforms a desired end-effector translational and rotational velocity into joint rate commands.

The simple resolved rate law described above is used in the SRMS flight software to drive the manual and the automatic modes. For these modes there is a requirement to move the hand coordinate system (or some coordinate system rigidly associated with the hand system) from one state to another with translation along a relatively straight path and rotation about a constant vector. Much effort has been spent finding such paths that are free from encounters with joint reach limits. When a manipulator has more joints than the number of degrees of freedom in the task space it is said to be kinematically redundant [3]. The Jacobian matrix for a kinematically redundant manipulator is not square and cannot be directly inverted to arrive at an easy resolved rate law. There is not enough information to solve for the joint rates needed to move the hand. In general, there are an infinite number of ways to move the joints in unison to provide the desired hand motion for the kinematically redundant manipulator [4],[5].

It is essential to arrive at a resolved rate law in order to control or simulate a manipulator. Several methods have been introduced to arrive at adequate resolved

rate laws for the kinematically redundant manipulator. One approach is to add specific constraints on the manipulator so that the kinematic equations can be solved. A more general approach is to minimize or maximize an objective function subject to the kinematic constraint equations. These methods have been investigated in several papers to study iterative solutions to the kinematically redundant constraint equations [1],[3-7].

In this paper a modified pseudo-inverse technique is used as a control law with joint reach avoidance for the redundant manipulator. This law can be derived by optimizing the sum of the weighted squares of the joint rates. The behavior of the control law is demonstrated and investigated using a kinematically redundant planar arm simulation. Several algorithms are introduced and evaluated for dynamically adjusting the weighting matrix during the trajectory for the purpose of avoiding joint reach limits.

PROBLEM FORMULATION

The resolved rate law for a manipulator is derived from the kinematic equations. For a manipulator with n joints and a hand operating in a task space of m dimensions, the m kinematic equations are of the form:

$$x = \{x_k\} = \{f_k(\theta_1,\theta_2,\ldots\theta_n)\}\ k=1,m \quad (1)$$

where x is the vector containing the task space coordinates and θ is the vector of joint angles. If each joint is moved by a small amount, $\Delta\theta$, then the movement of the hand in the task coordinates, Δx, is found in the differential of the kinematic equations:

$$\Delta x = [J]\ \Delta\theta \quad (2)$$

where J is the Jacobian matrix [8], composed of the partial derivatives of the functions f with respect to each of the joint angles. Similarly, the kinematic rate equations are found by differentiating the kinematic equations with respect to time.

$$v = dx/dt = [J]\ w \quad (3)$$

where v is the hand velocity vector expressed in the task coordinates and w is the vector of joint rates. The resolved rate law is found by solving the kinematic rate equation 3 for the joint rates (w) in terms of the hand velocity (v). In the case where the task space and the joint space have the same number of dimensions (m=n) the Jacobian matrix is square and the resolved rate law is easily found.

$$w = [J]^{-1}\ v \quad (4)$$

For a kinematically redundant manipulator (n>m) the Jacobian matrix is not square and the system of equations is underdetermined. For a 7 jointed manipulator operating in a task space of 6 dimensions there are 6 kinematic equations and 7 joint variables. The Jacobian matrix is a 6 by 7 matrix. To solve the set of equations, introduce an objective function to be minimized subject to the constraint equations:

$$v_{6x1} = J_{6x7}\ w_{7x1} \quad (5)$$

A weighted function of joint angles, proposed by Whitney [1] is:

$$Z = 1/2\ (a_{11}\ w_1^2 + \ldots\ a_{nn}\ w_n^2) \quad (6)$$

or in matrix notation:

$$Z = 1/2\ w^T\ A\ w \quad (7)$$

Using the method of Lagrangian multipliers the solution is:

$$w = A^{-1}\ J^T\ (\ J\ A^{-1}\ J^T\ )^{-1}\ v \quad (8)$$

When the weighting matrix is not used, or set to identity, the solution is:

$$w = [\ J^T\ (\ J\ J^T\ )^{-1}\ ]\ v \quad (9)$$

The expression in brackets in equation 9 is the Moore-Penrose pseudo-inverse of the Jacobian for the underdetermined system of equations 5 [4],[9].

The constraint on the weighting matrix A is such that the function Z must be non-negative for all values of w [11]. This condition will be satisfied by considering only diagonal weighting matrices with positive values. For the case of the seven jointed manipulator described above this resolved rate law is

$$\underset{7x1}{w} = \underset{7x7}{A^{-1}}\ \underset{7x6}{J^T}\ (\underset{6x7}{J}\ \underset{7x7}{A^{-1}}\ \underset{7x6}{J^T}\ )^{-1}\ \underset{6x1}{v} \quad (10)$$

where the dimensions of each matrix and vector have been indicated for clarity. The weighting matrix A can be used to control the motion of the joints by dynamically changing the values of the diagonal components during the trajectory.

IMPLEMENTATION

The control law expressed by equation 8 was implemented into a simulation of a Kinematically Redundant Planar Manipulator (KRPM). The KRPM model has a task space composed of 3 degrees of freedom: X, Z and P (pitch) directions for the hand; and all joints are pitch joints. The number of pitch joints (n) can be specified from 3 to 10. For this study, n was set to 4 so that there is only 1 redundant joint. This was done to form a direct analogy with the 7 jointed manipulator operating in a task space of 6 degrees of freedom. The 4 jointed KRPM is shown in Figure 1.

The KRPM model was simulated in FORTRAN on an HP9000 desktop 32 bit super micro computer. The Jacobian is computed numerically using a recursive vector form [1] to allow the simulation of various arms types. The attributes of the manipulator are described in a data file. Various manipulators may be represented by changing the data file.

Implementation of the Control Law

The optimal RRL (equation 8) was implemented into the KRPM arm model by first adapting the dimensions to those of the planar arm. The task space contains 3 dimensions, and the number of joints (n) may be 4 or greater. The RRL for the 4 jointed KRPM is defined as follows with dimensions shown.

$$[RRL]_{4x4} = A^{-1}_{4x4} J^{T}_{4x3} (J_{3x4} A^{-1}_{4x4} J^{T}_{4x3}) \quad (11)$$

The simulation iterates through the RRL to drive the hand to a desired state. The flow of the calculation is as follows. The arm is positioned at a valid set of joint angles (initial θ) and through the forward kinematics the resulting hand state (initial x) is computed. Then an input is made to indicate the desired final hand state for the arm (final x). The difference between the two hand states (Δx) is found.

$$\Delta x = x_{final} - x_{initial} \quad (12)$$

The number of steps to take during the trajectory (s) can be selected. The vector Δx is divided into s steps. The RRL is computed using equation 11 and then the desired joint angle step $\Delta\theta$ is computed.

$$\Delta\theta = [RRL]\, \Delta x/s \quad (13)$$

The joint angles are then updated by adding the changes in joint angles. This procedure is repeated for steps 2 through s, maintaining the same step lengtn in distance (X and Z) and in rotation but with an adjustment in the vector direction (Δx). After the last step is taken (step number s), a Newton-Raphson (NR) iteration is automatically invoked to trim up the final hand state to within a tolerance of the desired hand state. This method does not consider the joint rates of the manipulator. This simulation progresses by taking steps.

Step Size

A study was performed to determine the step size needed to provide hand motion along a straight line. Good results are measured by inspecting the path that the end-effector describes. The ideal trajectory should be a straight line. Several trajectories were tested while varying the number of steps between 1 and 80. A ten step iteration results in a reasonably straight hand trajectory, but eighty steps were required for very good results.

BEHAVIOR OF THE OPTIMAL RESOLVED RATE LAW

The ability of the rate law to handle the redundancy of the arm was demonstrated by driving the arm to the same hand state from various starting configurations. This also serves as an inverse solution to the kinematics, by providing several possible joint sets that satisfy the requested hand state. In Figure 2 the end-effector was commanded to the state X=3, Z=0, and Pitch= 0 (3,0,0) from four different initial joint configurations. In each case the end-effector ends up at the final state of (3,0,0), but with different final joint angles. This simple test demonstrates the ability of the control law to drive the KRPM to different final joint states for a given end-effector state. The final joint angles are dependent on the initial joint angles.

The above maneuvers were performed with the weighting matrix A in equation 11 equal to identity. This is the equivalent of the pseudo-inverse solution.

The Effect of the Weighting Matrix

The effect of the weighting matrix on the motion was demonstrated by running the same trajectory with various values of one of the components of the A matrix and observing the effects on the motion of the corresponding joint.

In Figure 3 the arm was commanded to the end-effector state of (2,0,0) at B from the joint angle state (90,-90,0,0) at A. When the weighting matrix is not used (pseudo-inverse), the final value of

the second joint is undesirable (Figure 3a). When a value of 2.0 is used for A(2,2) and 1.0 for all other diagonal components of A, the final position of joint 2 is noticeably better (Figure 3b). Joint 2 moved less from start to finish than in Figure 3a. The joint moves progressively less from start to finish as A(2,2) is increased. The remaining pitch joints have moved more to make up for the loss of mobility in joint 2, thus resulting in a more desirable overall final arm configuration.

In Figure 3a the final condition of the arm is not desirable because joint 2 could be very near a reach limit, thus restricting any future movement after arrival at the desired end-effector state. In Figure 3d, the final situation is much more desirable, because joint 2 has more freedom to move around in the neighborhood of the final end-effector state.

REACH AVOIDANCE ALGORITHMS

The behavior of the pseudo-inverse of equation 9 has been reported to be peculiar in some cases [4]. The peculiarity has been associated with joint reach limit violations during certain tasks such as a closed path or cyclic motion. If reach avoidance logic is incorporated into the RRL, these problems may be resolved. One method of incorporating reach avoidance into the RRL is to include the upper and lower joint limits as a constraint in the optimization algorithm [6], which is a complicated treatment. A simpler approach is taken here which makes use of the weighting matrix A in the RRL of equation 11.

In the previous section the effect of the weighting matrix on the arm motion was illustrated. It was shown that the redundant arm can be controlled to arrive at different final joint angles, some more desirable than others, and yet satisfy the same hand state. With these two findings, it is evident that the arm can be driven to arrive at various final joint angles as desired by controlling the weighting matrix during the trajectory.

The components of the weighting matrix (diagonal) must be computed repeatedly according to some driving requirements. Examples of driving requirements are obstacle avoidance, joint reach limit avoidance, or some mechanical or electrical criteria. For this study, the goal is reach limit avoidance.

Three algorithms were implemented for evaluation. The first algorithm is the simplest: when any joint is within a tolerance of a reach limit then the component of the weighting matrix for that joint is set to a large value (ABIG), otherwise the component is set to unity.

The second algorithm is similar to the first, but has the following requirement. If a joint is moving away from its reach limit then the weighting matrix component for that joint is set back to unity. This encourages the joint to move away from the tolerance zone.

The third algorithm does not use the tolerance test. The value of the weighting matrix component for each joint is scaled from 1 at its mid-range value to ABIG at either of its joint reach limits. Also, as in algorithm number 2, if the joint is moving toward its midrange value then the value of the weighting matrix component is set to unity. This algorithm is designed to encourage each joint to stay near the midrange value.

Each of the above three algorithms were implemented into the KRPM model described previously and tested until validated. The algorithms were then used to study a single joint encountering a reach limit and two joints encountering reach limits simultaneously.

To test the ability to avoid a single joint reach limit, a case was taken where the start and end of a trajectory are known to be valid end-effector states, but a reach limit is encountered without reach avoidance. In the trajectory that was selected the third joint begins at -90 degrees, reaches -106 degrees, and ends at -87 degrees. Suppose that the limit for this joint is at -100 degrees. Figure 4 shows the trace of the third joint with no reach avoidance and for each of the three reach limit avoidance algorithms using a value of 100 for ABIG. Each of the algorithms successfully avoided the imposed reach limit. In method 1 the joint angle does not move back out of the tolerance zone of 10 degrees. This is generally not desirable because the loss of motion in this joint removes the redundancy of the arm. In methods 2 and 3, the joint moved back out of the reach zone of 10 degrees from the reach limit. Methods 2 and 3 allow the joint to maintain motion.

In Figure 5a the arm was commanded from the joint state of (90,0,-135,90) to the hand state of (-.1,-2,90) causing two joints, joint 3 and 4, to approach reach limits. With reach avoidance both joint positions are improved in the final configuration (Figure 5b).

In the case shown in Figure 6a joint 3 exceeds a -160 degree limit and then goes past -180. With reach avoidance (Figure

6b) joint 2 swings out dramatically to allow joint 3 to avoid its reach limit.

SUMMARY

The pseudo-inverse Jacobian with a weighting matrix has been implemented with reach avoidance capability into a simulation of a kinematically redundant planar manipulator. This optimal resolved rate law has been demonstrated in the planar model and reach avoidance has been mostly successful with this model by dynamically adjusting the components of the weighting matrix during maneuvers. Three reach avoidance algorithms were tested. The locally optimized resolved rate law has been improved by incorporating joint reach avoidance.

REFERENCES

1. Whitney, D. E., "The Mathematics of Coordinated Control of Prosthetic Arms and Manipulators," J. of Dynamic Systems, Measurement and Control," Trans. ASME, Vol. 92, Dec. 1972, pp. 303-309.
2. Craig, J. J., Introduction to Robotics, Addison-Wesley, 1986.
3. Benhabib, B., Goldenberg, A.A., and Fenton, R. G., "A Solution to the Inverse Kinematics of Redundant Manipulators," J. of Robotic Systems, Vol. 2, No. 4, 1985, pp. 373-385.
4. Klein, C.A., and Huang, C. "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators," IEEE Trans. Systems, Man, and Cybernetics, Vol SMC-13, No. 3, March/Apr 1983.
5. Oh, S., Orin, D., Bach, M., "An Inverse Kinematic Solution for Kinematically Redundant Robot Manipulators," J. of Robotic Systems, Vol. 1, No. 3, 1984, pp. 235-249.
6. Goldenberg, A. A., Benhabib, B., Fenton, R.G., "A Complete Generalized Solution to the Inverse Kinematics of Robots", IEEE Journal of Robotics and Automation, Vol RA-1, No. 1, March, 1985, pp. 14-20.
7. Benati, M., Morasso, P., Tagliasco, V., "The Inverse Kinematic Problem for Anthropomorphic Manipulator Arms," J. of Dynamic Systems, Measurement, and Control, Vol. 104, March, 1982, pp. 110-113.
8. Paul, R. P., Robot Manipulators: Mathematics, Programming and Control, MIT Press., Cambridge, MA, 1981.
9. Forsythe, Malcom, and Moler, Computer Methods for Mathematical Computation, Prentice-Hall, NJ, 1977.
10. Whitney, D.E. "Optimum Stepsize Control for Newton-Raphson Solution of Nonlinear Vector Equations," IEEE Transactions on Automation Control, Vol. AC-14, No. 5, Oct., 1969, pp.572-74.
11. Zahradnik, Raymond L., Theory and Techniques of Optimization For Practicing Engineers, Barnes & Nobles, NY, 1971.

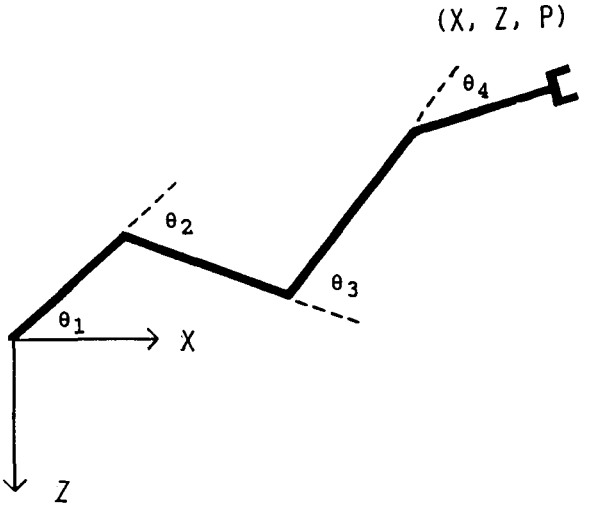


Figure 1 - A kinematically redundant planar manipulator with four joints and a task space of X, Z, and Pitch (P).

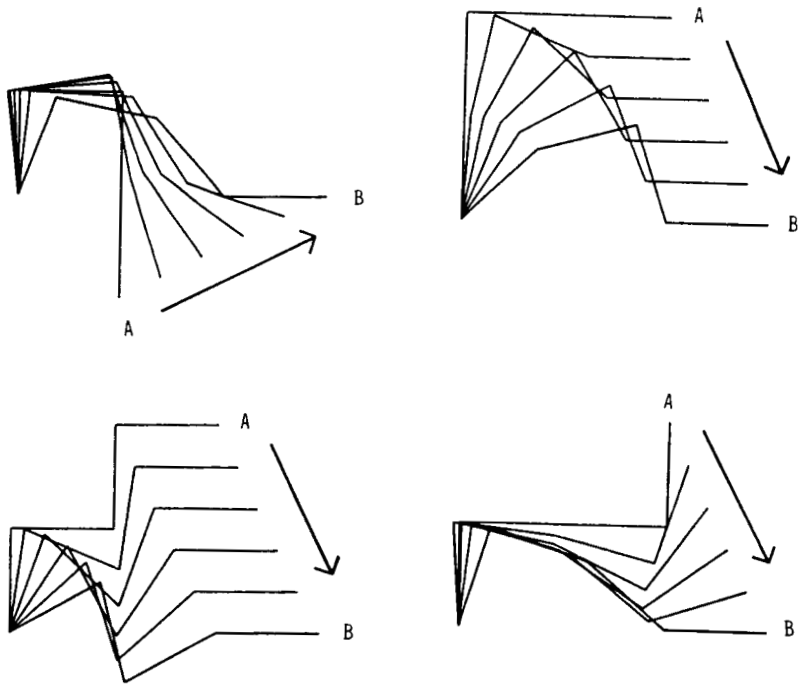


Figure 2 - The KRPM is commanded to the same hand state of (3,0,0) at B from four different initial joint angle states at A to demonstrate the behavior of the RRL.

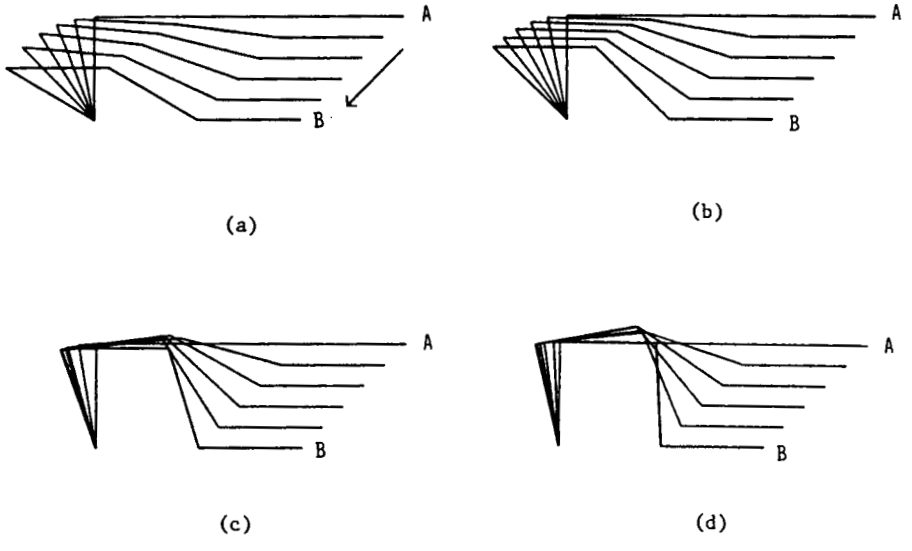


Figure 3 - Effects of the weighting matrix on the trajectory are shown for values of A(2,2) set at 1 (a), 2 (b), 10 (c), and 100 (d).

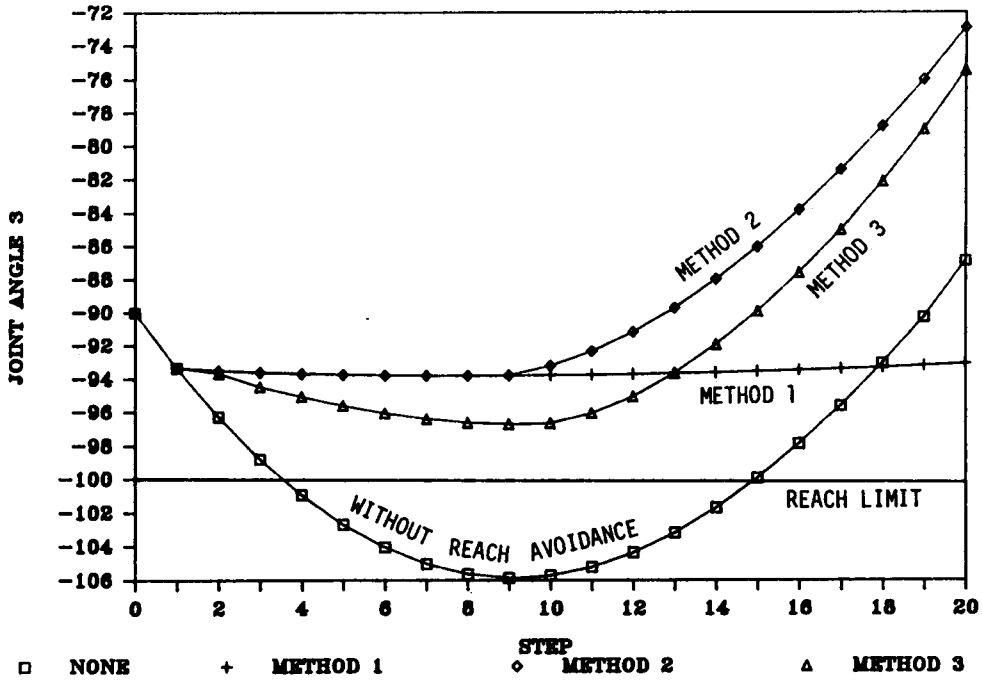


Figure 4 - The effects of reach avoidance on joint 3 are shown during a command from a joint state of (90,0,-90,0) to a hand state of (3,0,0). The reach limit is successfully avoided for each of the 3 reach avoidance algorithms.

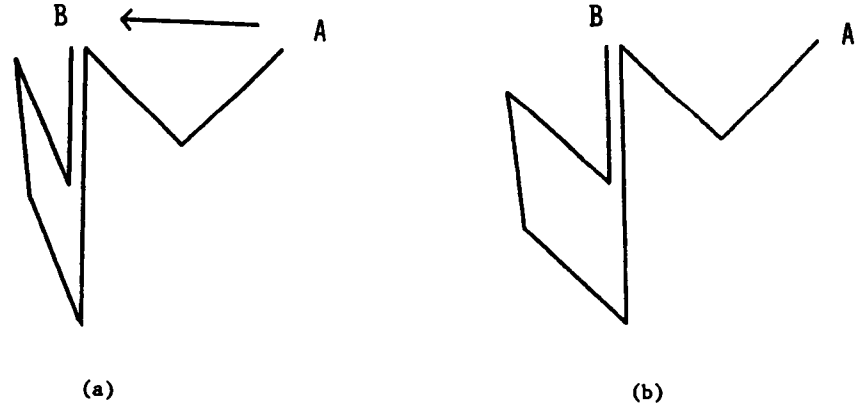
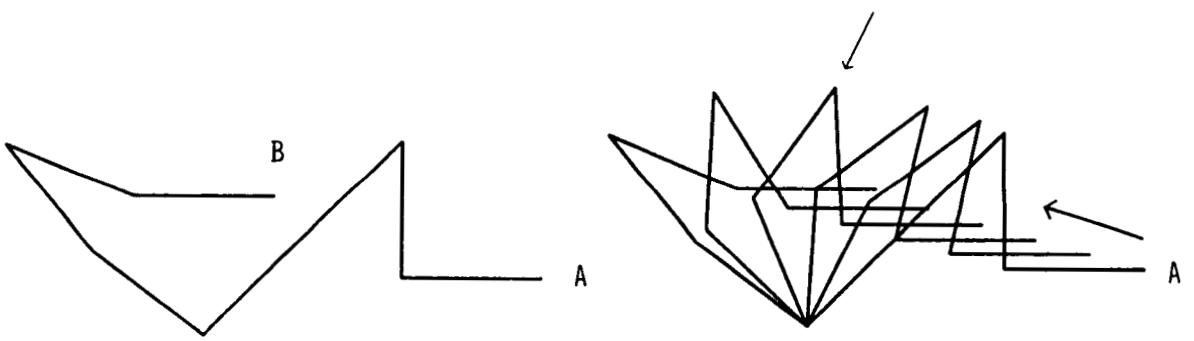


Figure 5 - Reach limit avoidance demonstration for the case of two joints violating reach limits. In (a) joints 3 and 4 violate reach limits of -160 and 160. With reach avoidance (b) both reach limits are avoided simultaneously.

(a)



(b)

Figure 6 - In (a) joint 3 exceeds -180 degrees. This problem is avoided in (b) when reach avoidance is used. The intermediate arm positions are shown to the right.

TELEROBOTIC RESEARCH AT NASA LANGLEY RESEARCH CENTER

Nancy E. Sliwa
Automation Technology Branch
NASA Langley Research Center
Hampton, Virginia

ABSTRACT

The Automation Technology Branch of NASA Langley Research Center has been researching automation and robotic techniques for space operations since 1979. This branch has developed and currently maintains the Intelligent Systems Research Lab. The lab houses a collection of computers and robotic peripherals that have been organized into a simulation testbed for telerobotic research. This testbed is being used to investigate and develop techniques for telerobotic on-orbit operations, such as assembly of large space structures and servicing and repair of spacecraft.

The branch has on-going work in many areas. Manipulator research includes dual-arm coordination studies, space manipulator dynamics, end-effector controller development, automatic space structure assembly, and the development of a dual-arm master-slave telerobotic manipulator system. Sensor research includes gravity-compensated force control, real-time monovision techniques, and laser ranging. Artificial intelligence techniques are being explored for supervisory task control, collision avoidance, and connectionist system architectures. A high-fidelity dynamic simulation of robotic systems, ROBSIM, is being supported and extended. Cooperative efforts with Oak Ridge National Laboratory have verified the ability of teleoperators to perform complex structural assembly tasks, and have resulted in the definition of a new dual-arm master-slave telerobotic manipulator.

This paper presents an overview of the facilities and research thrusts of the Automation Technology Branch, and includes a bibliography of research results and technical contacts for each research area. All technical contacts named are in-house personnel of the Automation Technology Branch.

INTRODUCTION

The Automation Technology Branch of NASA Langley Research Center has been researching automation and robotic techniques for space operations since 1979 (ref. 1). Early investigation established both the role of the branch in emphasizing the system integration aspects of telerobotics (ref. 2), and the need to provide an evolutional development of teleoperators toward increasing levels of intelligent automation. In examining NASA mission requirements for space automation, the tasks of structural assembly and satellite servicing were chosen as research foci (ref. 3).

TELEOPERATOR AND ROBOTIC TESTBED

The Automation Technology Branch maintains the Intelligent Systems Research Lab (ISRL), which houses a collection of telerobotic subsystems organized into a modular virtual architecture (ref. 4). This architecture, designated the Teleoperator And Robotic Testbed (TART), provides an environment where teleoperator and robotic technologies may be studied at the level of abstraction that best meets the researchers' needs. TART hides many of the low-level implementation details from the user and is defined by well-documented and tightly-controlled data structures having specific access mechanisms.

TART is a layered product in which each successive layer provides additional capability to the system. Currently, five layers are implemented: (1) user, (2) system, (3) scheduling, (4) communications, and (5) servo/sensor. The lowest four layers of TART accomplish most of the detailed programming and error checking required by user applications. The technical contacts for ISRL and TART are F. Wallace Harrison and C. Henry Lenox.

TELEROBOTIC SYSTEM SIMULATION

The TART architecture has been used to develop the TeleRobotic System Simulation (TRSS), a real-time, man-in-the-loop simulation of telerobotic operations. Early TRSS studies used computer-generated graphics to investigate the effects of time delays and teleoperator control modes on a precise alignment task (ref. 5). Later studies developed efficient algorithms to handle singularities in serial link manipulators (ref. 6). These techniques were demonstrated

on the ISRL manipulators (ref. 7). Most recently, TRSS has been used to investigate telerobotic structural assembly tasks for application to Space Station contruction.

Current TRSS control strategies are based on resolved motion rate control operating at 25 Hertz. Individual joint rates are integrated to provide joint positions, which are transmitted to servo-level pocesses resident in the manipulator controller. Operator control inputs can be referenced to arbitrary reference frames and combined with sensor-derived control signals. The technical contacts for TRSS are Donald Soloway and Bobby Glover.

Manipulator Control

A standardized set of homogeneous transform structures has been devised and implemented in TRSS to provide a generic control reference structure (ref. 8). By thus providing for the selection of arbitrary control reference frames, multiple manipulators can be simultaneously controlled individually or orchestrated to coordinate on a single task. TRSS currently incorporates two manipulators, which can cooperate in performing tasks such as positioning a long strut for connection to a node for structural assembly. The contacts for this work are Donald Soloway and L. Keith Barker.

Research continues with the Massachusetts Institute of Technology in dynamic control of manipulators in space. A virtual manipulator concept has been developed which facilitates the planning and control of the motion of manipulators mounted on spacecraft, thus minimizing the degrading consequences of manipulator/vehicle dynamic interactions (ref. 9). Additionally, the branch is supporting research in the control of flexible manipulators at the Georgia Institute of Technology. The contacts for these efforts are Jack Pennington and Donald Soloway.

An in-house effort is currently providing separate programmable controllers for each joint of the PUMA manipulators. This will allow research into alternative servocontrol strategies, including model-referenced adaptive control (ref. 10). The contact for this work is Donald Soloway.

Interest is growing in the possible use of trained neural networks to replace the need for kinematic manipulator control. Investigation is in progress of neural net capabilities and architectures. The contacts for these efforts are Donald Soloway and Nancy Sliwa.

Operator Interface

In addition to manual control, an interactive menu-driven interface is used in TRSS to provide a higher level of control abstraction to the operator, in effect automating some low-level task operations (ref. 11). This interface is organized such that more complex task operations can be described as a script of elementary operations and invoked as a single command primitive.

As this menu interface has evolved, it has become increasingly complex. Accordingly, an effort is now underway using expert system technology to automatically generate menu scripts based on high-level task requirements. This provides the operator with increased supervisory control. The contacts for this work are Nancy Sliwa and Eric Cooper.

Sensors

Sensor-derived control signals are essential to the automation of tasks in an uncertain telerobotic environment. TRSS uses both force/torque sensing and machine vision processing to assist in controlling the manipulators. A wrist-mounted, six degree-of-freedom force/torque sensor provides force and torque data that are processed into rates. These rates are then summed into the manipulator control rates to provide compliance, obstacle avoidance, and gravity compensation for loads carried by the manipulators. Additional force/torque sensors in the end-effector have also been used for fine motion compliance in dexterous tasks. The technical contacts for this work are Donald Soloway and Marion Wise.

Vision processing in TRSS currently emphasizes the determination of position and orientation of a marked part for acquisition by the manipulators (ref. 12, 13). This is done with quadrangle projection techniques, using a minimum of four identifiable target points and the principle of perspective transformation. This allows fast, robust automatic object acquisition by the telerobot.

Elastic matching techniques are also being researched in ISRL for automatic object recognition. A linear programming method called Goal Programming has been adapted to the elastic template matching approach to pattern recognition (ref. 14). This approach has been successfully applied to 3-space location of an isolated object, shape determination of isolated planar figures, image compression/restoration, and shape decomposition.

ISRL sensing capabilities are currently being expanded to include laser ranging. With support from the National Bureau of Standards and the U.S. Army, the branch has worked with Digital Signal Corporation to produce a high-speed, extremely precise range imager for the ISRL. The concept for this imager is based on the FM radar principle which makes use of the coherence and tunability of injection laser diodes (ref. 15). Work is currently in progress for applying this technology to produce an end-effector-mounted point-ranging sensor. Contacts for vision and laser research are Plesent Goode and Karin Cornils.

End-Effectors and Tools

TRSS has been using an end-effector fabricated at Langley from designs by the University of Rhode Island (ref. 16). This end-effector, a parallel jaw gripper, has proximity and cross-fire sensors for the detection of workpieces, limit and overload sensors, and manually-exchangeable fingers (ref. 17). The Automation Technology Branch has researched several variations of this end-effector, including finger-mounted force/torque sensors, automatically exchangeable ratcheting tools, and several task-specific gripper styles.

A microprocessor controller has been developed for this end-effector, with a sophisticated monitor to examine and change gains, speeds, and sensor values, and to move the grippers. This controller has been interfaced into the TART architecture to provide automated gripping and gripper-based sensing in TRSS (ref. 18). The technical contact for this work is Marion Wise.

Collision Avoidance

The branch has supported several efforts in collision avoidance and trajectory planning, including hextree environment modelling (ref. 19), freeway trajectories (ref. 20), gaussian configuration spaces (ref. 21), and dual-arm collision avoidance using velocity constraints (ref. 22). Research is currently in progress in-house establishing collision avoidance criteria for telerobotic environments, and developing a real-time collision avoidance monitor using abstract geometry and lazy evaluation techniques to avoid unnecessary calculations. The technical contact for this work is Nancy Sliwa.

System Architecture

In addition to the continuing enhancement of the TART virtual architecture, in-house research is continuing in the use of behavioral networks as a control architecture for telerobotic systems. Behavioral nets, a variant of connectionism, could provide a unified approach to combining high-level intelligent task control with low-level sensor and actuator control (ref. 23). This structure is also being investigated for use in interactive dynamic planning and scheduling systems. The contact for this work is Nancy Sliwa.

ROBOT SIMULATION

The Automation Technology Branch supported the development of ROBSIM (ROBot SIMulation), a high-fidelity, dynamic, off-line design and analysis tool for robotic systems and environments (ref. 24). This product, originally developed by Martin-Marietta, is being extended and enhanced by both in-house personnel and Grumman contractors. A real-time 3-D graphics display, improved user input interface, and interfaces to collision detection algorithms are upgrades which are currently in progress. This product has been distributed to more than 20 universities and research groups. The contacts for this work are F. Wallace Harrison and William Doggett.

TELEOPERATOR PERFORMANCE VERIFICATION

Teleoperated systems have so far seen only limited commitment for use in space due to the uncertainty of such systems' capabilities to perform complex realistic space tasks and of the time required to accomplish such tasks. The Automation Technology Branch, in cooperation with Oak Ridge National Laboratory (ORNL), has attempted to demonstrate the feasibility of teleoperated space operations and has established a database of task completion times (ref. 25).

Assembly Concept for Construction of Erectable Space Structures (ACCESS I) was a structural assembly flight experiment intended to study and verify the ability of astronauts to assemble in space a repetitive truss structure typical of that proposed for Space Station. Using the master-slave dual-arm manipulator (M-2) and highly skilled operators at ORNL, the ACCESS I experiment was duplicated in a controlled environment. This experiment proved that teleoperators have sufficient dexterity and control to perform such tasks in a timely manner without damage to the task components or to the manipulator system. Potential hardware modifications have been identified, and a data base of performance metrics has been established. The contacts for this work are Walter Hankins and Randolph Mixon.

SPACE STRUCTURE ASSEMBLY SIMULATION AND LAB

In cooperation with the Langley Structures Directorate, the Automation Technology Branch is investigating the automated assembly of large tetrahedral truss structures, such as would be used for large space antennae. Branch researchers have developed a real-time, 3-D perspective graphics simulation to be used in configuration analysis and development of assembly techniques. Since this analysis requires manual input, necessary components of the simulation include a command interpreter, a truss naming convention, a robotic system knowledge base, and an automated assembly sequence based on simple common substructures. Simple rules have been developed to allow an untrained operator to understand the assembly sequence and to successfully intervene in the event of system problems. An expert system approach is being investigated to minimize truss structure moves, potential interference, and robot arm base moves. The contacts for this work are Ralph Will and Sixto Vasquez.

The Automation Technology Branch and the Langley Structures Directorate have initiated the development of a laboratory facility to verify and demonstrate this approach to telerobotic construction of large space structures. The objective of this research is to design and test

structural elements, fasteners, end effectors, and tooling for telerobotic assembly of space structures, and to develop design criteria for manipulators, sensors, computers, and human/machine interface for systems for large space assembly. Future research with this facility will include enhanced sensing capability, curved structure assembly, repair of damaged structure components, and the automation of additional contruction tasks, such as cabling and panel installation. Contacts for this work are Marion Wise and Jack Pennington.

SPACE MANIPULATOR DEVELOPMENT

The Automation Technology Branch is working with Oak Ridge National Laboratory (ORNL) in developing a new concept in telerobotic manipulators (ref. 26). The Laboratory Telerobotic Manipulator (LTM) design combines the best capabilities of teleoperated manipulators and robotic manipulators. The objectives of the LTM program are:

1. Provide prototypical laboratory hardware for NASA ground-based research in shared teleoperator/autonomous control, teleoperator control methods, and space task demonstrations.

2. Provide a high quality, dexterous, dual-arm, force-reflecting teleoperator to maximize the commonality to astronaut EVA task performance.

3. Provide robotic features for laboratory research and an evolutionary path toward system autonomy.

4. Provide configuration and performance consistent with a space flight system.

Unique features of the LTM include (1) replicated joint concept for reduced design and fabrication cost, (2) dual-arm system for two-handed tasks and tooling, (3) differential traction drive mechanism for low backlash and high efficiency, (4)redundant kinematics for singularity and obstacle avoidance, (5) an interface to the GSFC/JPL Smart End-Effector, and (6) a hierarchical, distributed digital control system, with graphics-oriented operator interface.

LTM is configured as a dual-arm master-slave system. Each seven-degree-of-freedom arm has pitch and yaw motions in the shoulder, elbow, and wrist, plus roll motion at the wrist. Two basic two-axis modules provide speed and torque options, and are used in both the master and slave arms.

The LTM control architecture supports both robotic and teleoperated operations with real-time human control. The system will be entirely digital, using 32-bit microprocessors, standardized bus structures, and software implemented in high-level languages such as C, Pascal, Forth, and Fortran.

Detail design and fabrication of the first prototype of this space telerobotic system has begun, with intial operational planned for mid-1988. The Langley contacts for this work are Alfred Meintel and Jack Pennington.

INTERCENTER COOPERATIVE EFFORTS

Members of the Automation Technology Branch serve as consultants for and participants in several NASA automation and robotic efforts. These efforts include the Mars Rover project, the Flight Telerobotic Servicer project, the Telerobotic Demonstration Program, and the Systems Autonomy Technology Program. All branch publications are available upon request.

CONCLUSION

The Automation Technology Branch has developed an excellent telerobotic system research facility and maintains a group of in-house researchers who are competent not only in telerobot component technology, but also in the integration of such technology into specific application systems. While pursuing independent research to extend the state of the art in telerobotics for space applications, the branch is also cooperating with other research groups to solve specific application problems, advancing the use of telerobotics to fulfill NASA mission objectives.

REFERENCES

1. Meintel, Alfred and Larsen, Ron, "NASA Research in Teleoperation and Robotics," Society of Photo-Optical Instrumentation Engineers Conference, San Diego, California, August 1982.

2. Harrison, F. Wallace and Orlando, Nancy, "System-Level Approach to Automation," Fourth Annual UAH/UAB Robotics Conference, Huntsville, Alabama, April 1984.

3. Meintel, Alfred and Schappell, Roger, "Remote Orbital Servicing System Concept," Vol. 2, Satellite Services Workshop, NASA Johnson Space Center, Houston, Texas, June 1982.

4. Harrison, F. Wallace and Pennington, Jack, "System Architecture for Telerobotic Servicing and Assembly Tasks," 1986 SPIE Cambridge Symposium on Optical and Optoelectric Engineering, Cambridge, Massachusetts, October 1986.

5. Pennington, Jack, "A Rate Controlled Teleoperator Task with Simulated Time Delays," TM-85653, NASA Langley Research Center, September 1983.

6. Barker, L. Keith, Houck, Jacob and Carzoo, Susan, "Kinematic Rate Control of Simulated Robot Hand at or Near Wrist Singularity," TP-2240, NASA Langley Research Center, May 1984.

7. Hankins, Walter and Mixon, Randy, "Baseline Experiments in Teleoperator Control," TP-2547, NASA Langley Research Center, July 1986.

8. Barker, L. Keith and Soloway, Donald, "Coordination of Multiple Robot Arms," JPL Workshop on Space Telerobotics, Pasadena, California, January 1987.

9. Dubowsky, Steven, and Vafa, Zia, "A Virtual Manipulator Model for Space Robotic Systems," JPL Workshop on Space Telerobotics, Pasadena, California, January 1987.

10. Kornbluh, Roy, "An Experimental Evaluation of Robotic Manipulator Dynamic Performance Under Model Referenced Adaptive Control," Master's Thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, September 1984.

11. Cooper, Eric and Sliwa, Nancy, "ISRL TRSS Operator Menu Interface," ATB internal documentation, NASA Langley Research Center, 1987.

12. Goode, Plesent and Cornils, Karin, "Monovision Techniques for Telerobots," JPL Workshop on Space Telerobotics, Pasadena, California, January 1987.

13. Cornils, Karin and Goode, Plesent, "Location of Planar Targets in Three Space From Monocular Images," 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, Greenbelt, Maryland, May 1987.

14. Goode, Plesent, and Cornils, Karin, "A Multifunction Recognition Operator for Telerobotic Vision," AIAA Guidance, Navigation, and Control Conference, Williamsburg, Virginia, August 1986.

15. Goodwin, F., "Coherent Laser Radar 3-D Vision Sensor," Sensors '85, Detroit, Michigan, November 1985.

16. Tella, R., Birk, J. and Kelly, R., "General Purpose Hands for Bin-Picking Robots," Man and Cybernetics, Vol. SMC-12, No. 6, November/December 1982.

17. Wise, Marion, "ISRL Parallel Jaw Gripper End-Effector," ATB internal documentation, NASA Langley Research Center, 1984.

18. Bynum, Dr. Wm., "Command/Response Protocols and Concurrent Software - Final Report," NAG-1-670, NASA Langley Research Center, June 1987.

19. Potter, Dr. Jerry, "Robot Environment Expert System," CR-3915, NASA Langley Research Center, August 1985.

20. Brooks, Dr. Rodney, "Find-Path for a PUMA-Class Robot," Third Annual National Conference on Artificial Intelligence, Washington, D.C., August 1983.

21. Krause, Donald, "Path Planning with Obstacle Avoidance," NAG-1-632, NASA Langley Research Center, January 1987.

22. Wallace, Richard, "Three Findpath Problems," Fourth Annual National Conference on Artificial Intelligence, Austin, Texas, August 1984.

23. Sliwa, Nancy and Soloway, Donald, "A Lattice Controller for Telerobotic Systems," American Controls Conference, Minneapolis, Minnesota, June 1987.

24. Lowrie, James et al., "Evaluation of Automated Decisionmaking Methodologies and Development of an Integrated Robotic System Simulation," CR-165975, NASA Langley Research Center, September 1982.

25. Hankins, Walter, Mixon, Randolph, Jones, Howard, and Burgess, Thomas, "Space Truss Assembly Using Teleoperated Manipulators," 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, Greenbelt, Maryland, May 1987.

26. Herndon, J., Hamel, W. and Kuban, D., "Traction-Drive Telerobot for Space Manipulation," 1987 IEEE International Conference on Robotics and Automation, Raleigh, North Carolina, April 1987.

Manipulator Arm Design for the Extravehicular Teleoperator Assist Robot (ETAR):* Applications on the Space Station

Margaret M. Clarke
Space Station Systems Division
Rockwell International Corporation
12214 Lakewood Avenue
Downey, California 90241
(213) 922-0785

Charles J. Divona
and
William M. Thompson
NSA Electro-Mechanical Systems Division
2701 Saturn Street
Brea, California 92621
(714) 996-4670

ABSTRACT

The preliminary conceptual design of a new teleoperator robot manipulator system for Space Station maintenance missions has been completed. The system consists of a unique pair of arms that is part of a master-slave, force-reflecting servomanipulator. This design allows greater dexterity and greater volume coverage than that available in current designs and concepts.

The teleoperator manipulator is specifically designed for space application and is a valuable extension of the current state-of-the-art earthbound manipulators marketed today.

This paper describes the manipulator and its potential application on the Space Station.

INTRODUCTION

The potential use of a teleoperator robot system in and around the Space Station is being considered as an aid to astronauts in performing extravehicular activity (EVA). This paper describes the preliminary conceptual design work for a telerobot for Space Station maintenance and the anticipated application of the device for maintenance and tending of the Space Station user payload complement. Earlier work by the same team, completed in 1985, established mission objectives, equipment and interface requirements, and the environmental constraints of a free-flying telerobot. This early work has been reported extensively (References 1-5). In the current work we analyzed requirements for typical EVA maintenance tasks and then developed detailed manipulator concept solutions to those requirements. The resulting application concept employs the extravehicular teleoperator assist robot (ETAR) as a dedicated EVA tool used for tending the Space Station user payloads. The ETAR will assist the Space Station crew during EVA, either directly assisting an EVA astronaut or working alone in the EVA environment. The ETAR would be controlled by a second astronaut within the shirt-sleeve confines of the station as intravehicular activity (IVA). The IVA astronaut views the EVA operation through a window, from the cupola, or on a closed-circuit TV monitor. The ETAR design emphasizes maximum dexterity, minimum weight, high reliability, and optimum control characteristics. ETAR manipulator requirements and solutions were developed through the first five subtasks listed below. Lastly, potential applications were developed to use the ETAR to maintain the Space Station EVA payload complement.

1. Analysis of maintenance task requirements
2. Manipulator mechanical requirements
3. Preliminary mechanical design**
4. Operational requirements
5. Control concepts
6. Applications to user payload tending

ANALYSIS OF MAINTENANCE TASK REQUIREMENTS

We identified 21 typical tasks selected from an assortment of maintenance missions and representing a wide range of transfer routes, dimensions, mass, and handling requirements.

To help define requirements further, we selected three test cases from the sample for further study. They involved replacing the following equipment: solar array on the Advanced X-Ray Astrophysics Facility (AXAF), faint object spectrograph (FOS) on the Hubble Space Telescope (HST), and multimission modular spacecraft (MMS) modules in common use on present and future satellites. The reasons for choosing these three cases are summarized in Table 1.

Table 1. Test Case Characteristics and Handling Requirements

Test Case	Characteristics	Handling Requirement
Solar array	Fragile, large, flexible	Precise force control
FOS	Special covers and latches	Dexterity
MMS	Frequent use	Must be quickly and easily performed

MANIPULATOR MECHANICAL REQUIREMENTS

The preceding information established the basic configuration, degrees of freedom, and size requirements for the ETAR manipulator slave arms. The criteria used in establishing these requirements provide a capability to accomplish the above tasks and to be of general assistance to the EVA crew.

*Work performed under Rockwell Contract SC000730-01 with Essex Corporation to support NAS8-36629

**Preliminary designs were evaluated with respect to existing teleoperated manipulator configurations. The results of this evaluation are reported in Reference 9.

Configuration: Number of Joints

The ETAR arm will be controlled predominately in a teleoperated mode; that is, the crew person will control the remote arm from within the spacecraft using familiar hand and arm movements while viewing the EVA operation from the cupola, a window, or a TV monitor. We will use the arm kinematics that approximate a man-like configuration and arm motions for the operator to relate most efficiently to, and control the motions of, the ETAR arm. To implement this, an anthropomorphic configuration was assumed with the arm containing a shoulder, elbow, and wrist joint.

Degrees of Freedom (DOF's) of Each Joint

The shoulder has two DOF's: pitch and yaw. This configuration is analogous to the human shoulder, which allows the arm to swing out to the side (yaw) and forward (pitch). Most of the contemporary manipulators in the nuclear industry incorporate pitch only. These include the advanced servomanipulator (ASM) developed at Oak Ridge National Laboratory (ORNL), the M-2 (Central Research Laboratories), and the RM-10 (Remote Technology Corporation). Incorporation of a yaw DOF greatly extends the total coverage capability of the arm. This capability is of major importance because it minimizes the redocking requirements of the space vehicle that carries the arm.

The elbow has both pitch and yaw. These DOF's are standard on most anthropomorphic configurations, allowing the wrist and hand to be brought in close to the body.

A 3-DOF wrist joint is incorporated in the ETAR arm. The motions are pitch, yaw, and roll. The pitch-yaw-roll wrist is a dexterous, back-drivable configuration that can be designed to be singularity free. Our prime design candidate for the wrist is the tendon-linkage type in which all wrist drives (including the wrist roll) are located in the forearm. This arrangement is far superior to any currently proposed concepts advocating designs that place the wrist roll drive forward of the pitch-yaw axis.

For this study, only an open-close, tong-type end effector will be considered. Other special hand configurations will be considered later.

Figure 1 shows the ETAR arm with the DOF's. The arm has 7 DOF's plus the end effector motion. This is one more than the minimum needed to position an object anywhere within the operating envelope in any angular orientation. The extra DOF permits

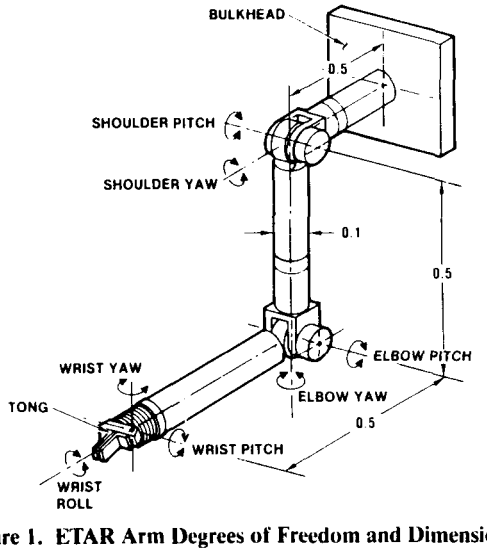


Figure 1. ETAR Arm Degrees of Freedom and Dimensions

not only a greater operating envelope, but also the capability of reaching points in space with two different arm configurations. This allows the arm to reach objects that would otherwise be impossible.

Bases for Overall Size

The 21 orbital maintenance representative test cases were reviewed to establish the overall size envelope of the ETAR. The largest dimension of each object to be transferred was established and the mean and median determined excluding items that were extremely large or for which we lacked sufficient information. The results are the following:

- Mean (17 items) = 3.14 m
- Median (17 items) = 3 m

In establishing the ETAR arm length, it was assumed that a two-arm configuration would be employed on the assist robot, the arms would be separated by a distance of 0.75 m, and the total wrist-pivot to wrist-pivot dimension would correspond to the 3-m dimension determined above. This assumption is based on the outstretched dimension of the arms. The 0.75-m separation is a good compromise for establishing a reasonable zone of mutual operation for both arms. From the above, the dimension of a single arm, from the wrist-pivot to the shoulder-pivot, is the following:

$$D_{w-s} \frac{3 - 0.75}{2} = 1.125 \text{ m} \quad (1)$$

To maximize commonality of parts and modules, the wrist-to-elbow and elbow-to-shoulder dimensions are assumed equal to half of D_{w-s} or

$$D_{w-e} = D_{e-s} = D_{w-s} = 0.5625 \text{ m} \quad (2)$$

To accommodate the shoulder yaw motor/drive, a distance of 0.5 m between the robot vehicle bulkhead and the shoulder axis was assumed.

Figure 1 also shows the ETAR dimensions. The upper and lower arm diameter (0.1 m) is based on the estimated size of the motors mounted inside the arms.

The volumetric coverage of the ETAR is significantly greater than existing manipulators, such as the M-2, ASM, and RM-10. This is a direct result of the shoulder yaw DOF. Except for the shoulder yaw, the other major DOF's are symmetrical with respect to the bulkhead. In this configuration, the bulkhead can be oriented at any position in space and allows the arms to interface effectively with the work task. The wrist pitch and yaw allow the location of the hand or end effector to be anywhere within a hemisphere perpendicular to the lower arm. We assume that the tong opening will allow the gripping of objects as large as 9 cm.

PRELIMINARY MECHANICAL DESIGN

The preliminary mechanical design configuration (Figure 1) incorporates a similar pitch-yaw joint in the shoulder and elbow and a 3-DOF pitch-yaw-roll wrist. The shoulder and elbow yaw drives are located above the respective joints within the cylindrical arm structures. The pitch drives are concentric with the pitch axis. All three drives for the wrist are in the lower arm.

The wrist mechanism is based on a design described by Rosheim (Reference 7). The mechanism features 180 deg of pitch and yaw and continuous bidirectional roll. It is singularity free,

back-drivable, and mechanically efficient. Two push-pull rods drive the pitch and yaw motions. These are actuated via linear ball screws by motor-resolvers located in the forearm. The roll drive motor directly rotates the tool plate at the end of the unit.

OPERATIONAL REQUIREMENTS

Because the purpose of the ETAR manipulator arm is to assist or conduct many of the operations performed by an EVA astronaut, we used a human arm as a reference base and then stated ETAR arm operational characteristics compatible with astronaut capabilities to perform identified tasks. Resulting ETAR arm operational characteristics are discussed below.

Force

A reasonable operational capability for an astronaut is to exert a steady force of 90 n (20 lbf) with the hand in any direction. This is, therefore, the design force capability of the ETAR arm: any DOF can exert this force when acting alone. With several DOF's acting, the force capability is the vector addition of forces. Each DOF also has a larger peak capability of 135 n (30 lbf) for short times. Power supply limitations and motor heating establish the peak force value and its time duration.

Speed

The minimum speed capability was based on operating experience with master-slave manipulator systems. A speed capability of 1 m/sec (40 in./sec) does not noticeably impede the motions of the operators. They are not forced to fight the system to increase speed if it has force feedback or distracted by lack of synchronization with no force feedback.

Force Sensitivity

The ability to detect small forces or small changes in force is a further requirement. Again, manipulator operational experience indicates that a sensitivity of 2 percent or less of maximum force is desirable. This requirement is related to the one for low friction. The operator-sensed friction from bearings, gears, motor brushes (if any), etc. must be significantly less than 2 percent.

Natural Control

The above characteristics will provide the system with natural control. The ultimate objective is to achieve telepresence—the operator has the sensation of being at the work place and not working through an intermediate device. Anything that detracts from that illusion, such as force and speed limitations, reduces operational effectiveness and efficiency. In addition, the choice of DOF's and configuration of the master also affect natural control. The latter also influences the degree of dexterity the system must have. The configuration and DOF's of the arm and master must not require awkward maneuvers to perform the required operations.

Reliability

Finally, all of the above features and characteristics must be achieved with a minimum of mechanical complexity and with highly reliable, space worthy components.

CONTROL CONCEPTS

ETAR Arm Drive Systems

We have chosen direct drive motors in the arms for arm actuation. This eliminates steel cables, tapes running over pulleys, or torque tubes and gear drives found in current servomanipulators. The ETAR concept takes advantage of the microgravity environments by placing the motors in the arms. This eliminates much of the mechanical complexity, friction, and maintainability and reliability problems associated with earthbound designs.

The motors in this design are low-speed, high-torque, brushless servomotors. The low speed and high torque require little or no gearing. Some commercially available brushless motors operate in a speed range of less than 1 r/sec and produce high output torque. They almost meet the requirements of speed and torque for some of the wrist motions and, therefore, could be applied with only a small advancement in the state of the art. The shoulder and elbow motors, however, will require further development, incorporation of gearing, or both.

The mechanical power output of each motor and, hence, the electrical power input is approximately the same. This is because each motor must produce the same maximum linear force of 90 N (20 lbf) and the same maximum linear speed of 1 m/sec (about 40 in./sec). For example, although the torque arm lengths are longer for a shoulder motor than for a wrist motor, the rotational speed for the shoulder is proportionally lower to achieve the same tangential velocity.

All of the motors include an integral, high-accuracy, brushless resolver as a position transducer. Tachometer generators are not included since state-of-the-art control electronics can easily derive velocity signals from the position signals.

Control System

Each of the candidate arms can be controlled in either a teleoperator or preprogrammed robotic-type mode. In the teleoperator mode, the crew person remains active in the control loop, receiving information from the remote task site through viewing and force reflection. The crew person then controls the remote manipulator arms from the IVA control station. In the robotic mode, the manipulator arms are programmed to perform tasks autonomously. The crew person is not active in the control loop, but may assume a vigilant role.

Basic System-Teleoperator Mode

In a teleoperator mode, a special type of control system must be used. The position-position, force reflecting servomechanism (FRS) without force transducers is the classical system used in all operating servomanipulators. One of these systems will be used for each DOF of the ETAR servomanipulator.

The FRS consists of two positional servomechanisms connected bilaterally so that the input of one is the output of the other. The combination works to produce position and velocity correspondence between the two systems. Force reflection is produced by a positional or velocity error between the two systems. Both systems try to reduce the errors—one by pushing against a load or obstruction, the other by pushing against the operator. If both systems have the same components, it is a one-to-one system and the forces are equal.

Robotic Configuration

The servomechanisms described above can be driven not only in teleoperator mode by operator-generated signals, but also in robotic mode by a preprogrammed signal. The preprogramme signal can be stored in a memory medium locally or remotely. T source of the programmed signal could be a teaching operation which an operator runs through a sequence of motions and p tion command signals are recorded for playback. Another so is a computer-calculated and -generated sequence. A combin of both can also be used.

Master Controller

We recommend a replica master controller for the ETAR manipulator arms. We also considered, but do not recommend, 6-DOF hand controllers.

Replica Master

Our basic concept for the master controller is a replica of the slave and consists of two skeletal-type replica master arms on a common mounting structure. It resembles the slave's configuration, though it may be scaled down in size. It provides the crew person with a natural control system. Experience with earthbound manipulator systems has indicated that skeletal slave-replica master controllers are easy to operate, safe, and simple enough to be readily maintainable.

6-DOF Hand Controllers

Although we did review the state of the art of 6-DOF hand controller, we are not recommending it as a controller for ETAR. For example, Corker and Bejczy (Reference 8) at Jet Propulsion Laboratory developed a 6-DOF force-reflecting, universal hand controller (FRHC). It has an operating volume of about 1 ft and is a compact device that saves space in the master station. The main reason it was not chosen as the prime concept is that it cannot accommodate the seventh DOF in a natural way. In addition, the motions of each DOF do not coincide with those of the slave arm. Therefore, the DOF's are not independent, and coordinate transformations must be performed in the control system to resolve the slave motions. The additional computation and potential cross-coupling between DOF's would result in a more complex control system and potential stability problems.

ETAR Complete Control System

As depicted in Figure 2, a complete ETAR system consists of two operating arms, TV systems, and auxiliary systems. The master station provides teleoperator manual controls, appropriate automatic control and ETAR transporter (carriage) controls. Electric power to the system is provided via bus bars, with signal control via infrared or laser transmission.

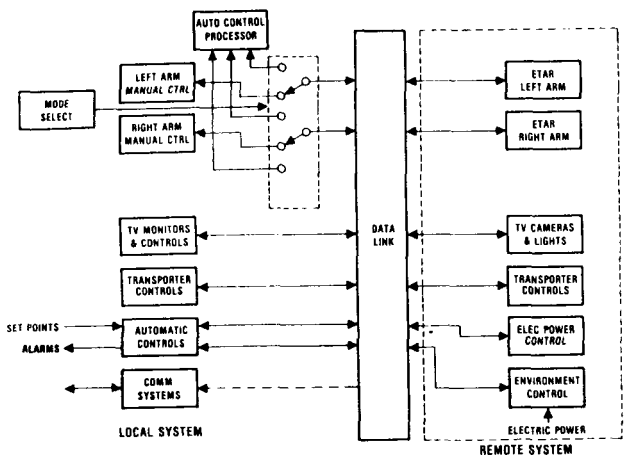


Figure 2. ETAR Complete Control System

APPLICATIONS TO PAYLOAD TENDING

The ETAR manipulator has two major applications in user payload tending: planned payloads and advanced missions.

PLANNED PAYLOADS

A Rockwell statistical analysis of the NASA/SSP mission requirements data base suggests that ETAR would be useful in servicing the payloads identified in Table II. Tasks include consumables replacement, specimen change out, and cleaning sensors and reflectors. Tasks will be scheduled routinely and performed by assisting the EVA astronaut or by the ETAR alone, producing significant EVA time savings.

Table II. ETAR Tasks for Planned Payloads

Mission Code	Payload Name	ETAR Task
SAAX 0001	Cosmic ray nuclei experiment	Change out pressurized gas bottle
SAAX 030	Space Station Hitchiker 1	Change out equipment can
SAAX 207C	High-resolution telescope and spectrograph	Replace film cassette
SAAX 207E	Solar ultraviolet spectral irradiance monitor	Replace inert gas bottle
TDMX 2011	Spacecraft materials and coatings	Change out specimen tray
TDMX 2441	Microelectronics data systems experiment	Change out black boxes
SAAX 4002	Polcats (Canada)	Clean sensors
COMM 4001	Solar cells (Canada)	Replace solar panels

ADVANCED MISSIONS

We have identified several ETAR applications for advanced, yet to be determined, user missions. Two concepts are the payload farm and batch processing facility.

Payload Farm

The payload farm concept requires the ETAR to be mounted to a carriage transporter that travels on a fixed track attached to the Space Station truss (Figure 3). Two or more rows of payload hosting bars, each capable of hosting a number of payloads, are in close proximity to the track. The bars provide support, convenient location, and any required power or utilities to the rows of payloads. The end of the track leads to an airlock through which the ETAR and carriage can enter and exit an IVA area for service.

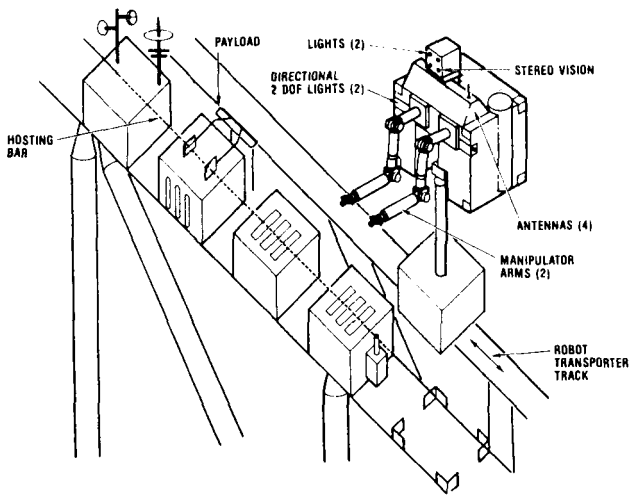


Figure 3. Concept for ETAR Payload Tending

Payloads are separate and discrete, perhaps each belonging to a different commercial user. However, they adhere to rigorous standards regarding size, shape, mass, and connectors and interface with the ETAR end effector. The ETAR can, therefore, be programmed to repeat the same service tasks on each payload (for example, payload change out). The interior of the payload can then remain proprietary to its owner—an attractive feature for commercial users.

Batch Processing Facilities

The Space Station may also house batch processing facilities in which operations are analogous to earthbound chemical plants where humans perform similar operations on batteries of production tanks (pharmaceutical production, wine making). ETAR places raw materials in the user production facilities, performs any required service, and later retrieves the finished products. This concept realizes significant EVA crew time savings and enhances the role of the Space Station as a commercial facility for multiple batches.

CONCLUSIONS

From the information developed in this study, an effective teleoperated manipulator system for application on and around the Space Station can be developed with a minimum extension to the state-of-the-art technology. The ETAR can perform many of the routine tasks now being performed by EVA astronauts and assist the EVA astronaut in performing extensive tending tasks.

In developing the preliminary concept design of the ETAR manipulator system, existing teleoperated systems were reviewed and found to lack many attributes and capabilities we feel are necessary for a space-based system. In most cases, this lack of space compatibility is understandable due to the fact that the systems were conceived for earthbound application. The ETAR arm, however, should prove to be an invaluable Space Station asset because it will have been specifically designed to assist in Space Station operations.

SUMMARY

The work presented above represents the preliminary conceptual design of a teleoperated manipulator system specifically designed for use on a spacecraft such as the Space Station. A number of anticipated Space Station tasks were evaluated to establish the requirements of the system. The system is designed to be controlled predominantly by an IVA astronaut in the Space Station shirt-sleeve environment. The slave arms of the system are located outside the station and will be of valuable assistance in payload tending tasks, enhancing the commercial role of the Space Station.

REFERENCES

1. Clarke, M.M., M.A. Bronez, W.M. Thompson, and M.T. Gibbons, *Orbital Equipment Transfer Techniques, Final Report, Task 1,* Rockwell International, Space Station Systems Division, Downey, California, SSS 85-0095, September 1985.

2. Clarke, M.M. and M.A. Bronez, "Telerobotics for the Space Station," *Mechanical Engineering*, February 1986, pp. 66-72.

3. Bronez, M.A., M.M. Clarke, and A. Quinn, *Requirements Development for a Free Flying Robot — The ROBIN*, Proceedings of IEEE Conference on Robotics and Automation, San Francisco, California, April 1986.

4. Quinn, A., M.M. Clarke, W.M. Thompson, and N. Shields, *A Robot Device for Orbital Equipment Transfer*, Proceedings of SME Robots 10 Conference, Chicago, Illinois, April 1986, pp. 5-99 to 5-108.

5. Bronez, M.A., M.M. Clarke, and A. Quinn, *A Telerobot for Space Station Orbital Equipment Transfer*, Proceedings of Second World Conference on Robotics Research, Scottsdale, Arizona, August 1986.

6. Clarke, M.M., W.M. Thompson, and C. J. Divona, *Requirements and Conceptual Design of the Manipulator System for the Extravehicular Teleoperator Assist Robot (ETAR)*, Rockwell International, Space Station Systems Division, Downey, California, SSS 86-0139, November 1986.

7. Rosheim, M.E., *Four New Robot Wrist Actuators*, Proceedings of SME Robots 10 Conference, Chicago, Illinois, April 1986, pp. 8-1 to 8-45.

8. Corker, K. and A.K. Bejczy, *Recent Advances in Telepresence Technology Development*, Proceedings of Twenty-Second Space Congress, Cocoa Beach, Florida, April 1985.

9. Divona, C.J., W.M. Thompson, A. Quinn, and M.M. Clarke, *A Teleoperator for On-Orbit Manipulation*, Proceedings of International Topical Meeting on Remote Systems Pasco, Washington, April 1987.

DEVELOPMENT OF A FACILITY USING ROBOTICS FOR TESTING AUTOMATION OF INERTIAL INSTRUMENTS

Joy Y. Greig
Central Inertial Guidance Test Facility
6585th Test Group, Holloman AFB, NM 88330

Gary B. Lamont, Daniel J. Biezad,
Zdsislaw H. Lewantowicz
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright-Patterson AFB
Dayton, Ohio 45433

ABSTRACT

The development of a facility for inertial instrument testing using a robot arm involves a variety of studies. Foremost is a feasibility study of the application which involves accuracy analysis of the static and dynamic configurations. As part of this aspect, simulation of a robot arm in performing the tests is desired along with modeling evaluations. Also, economic analysis of various arm configurations should focus on appropriate commercial systems that have a high probability of providing an applicable testing environment.

In this study, the Integrated Robotics System Simulation (ROBSIM) was used to evaluate the performance of the PUMA 560 arm as applied to testing of inertial sensors. Results of this effort were used in the design and development of a feasibility test environment using a PUMA 560 arm. The implemented facility demonstrated the ability to perform conventional static inertial instrument tests (rotation and tumble). The facility included an efficient data acquisition capability along with a precision test servomechanism function resulting in various data presentations which are included in the paper. Analysis of inertial instrument testing accuracy, repeatability and noise characteristics are provided for the PUMA 560 as well as for other possible commercial arm configurations. Another integral aspect of the effort was an in-depth economic analysis and comparison of robot arm testing versus use of contemporary precision test equipment.

INTRODUCTION

Specialized test facilities, such as the Central Inertial Guidance Test Facility (CIGTF) at Holloman Air Force Base, New Mexico, are responsible for the testing of high quality inertial rate sensors and accelerometers. Due to the large investment in resources, it is important that all sensors be free from major defects when scheduled for precision testing. Initial sensor checkout tests, for example, should not tie up unique and specialized test equipment which may cost millions of dollars (2,4).

Although these expensive devices for testing inertial sensors have been very effective, due to their unique design they often lack the flexibility required to implement new test procedures. Moreover, there is little evidence of rapid innovation in designing and building new test fixtures with enhanced capabilities. These problems of cost, inflexibility, and lack of new capabilities impose significant constraints on component testing programs.

A potential approach to addressing these problems comes from the rapidly developing engineering science of robotics, where cost is decreasing due to the exponential rise in the number of units being produced (increasing from 20,000 units in 1976 to 250,000 in 1984), and where the digital capabilities being designed into robots have the potential to provide flexibility in systems tests and data acquisition (16). Finally, robotics is a highly innovative area fueled by vast research funding. It is probable that if the key difficulty of precision can be solved, the use of programmable robots for inertial testing should become a reality.

This paper discusses the feasibility of robotics applications to inertial component testing by addressing three major areas: technical feasibility, economic feasibility, and limitations. Facilities at the Air Force Institute of Technology (AFIT) provided the testing environment. Technical advice and the accelerometer for the study were provided by CIGTF.

In the following sections we discuss feasibility objectives and robot specifications, approach and design of the experiment, results of the experiment, economic analysis of a robotics test facility, and conclusions and recommendations resulting from the study.

OBJECTIVES/SPECIFICATIONS

The robot in itself is not a precision test device relative to inertial sensor accuracies. Both inertial sensor and robot accuracies were investigated in this study to determine the feasibility of using a robot as a testbed. Three tests on a PUMA 560 robot arm were accomplished to

illustrate this and to examine robot performance criteria for sensor/system laboratory testing.

Once technical feasibility is established, the next important question is, "Is the proposal economically feasible?" To determine cost-effectiveness, a life cycle costing analysis was performed for both the robotic and non-robotic testing units.

Limitations of robotic testbeds are a final consideration. Practical engineering limits, computer modeling limits, and measurement and instrumentation limits are addressed and related to the component test facility application.

APPROACH/DESIGN OF EXPERIMENT

The overall approach of the experiment was to design tests which would determine the feasibility of using a robot arm as a testbed for inertial sensors and thus the feasibility of developing a robotic inertial guidance test facility.

Simulation and Emulation

An effective approach to the development of robotics applications is to proceed first with simulation and then follow with emulation. Simulation is performed using a comprehensive robot simulation program (3,11,17). At AFIT a powerful computer program called ROBSIM produced by Martin Marietta Aerospace for NASA Langley was employed. A good simulator such as ROBSIM includes arm and environment synthesis (or definition), joint motion or joint torques and forces simulation, and analysis of the simulation.

Emulation of a test facility followed, using a PUMA 560 Robot Arm, data acquisition equipment, and a precision accelerometer and gyroscope.

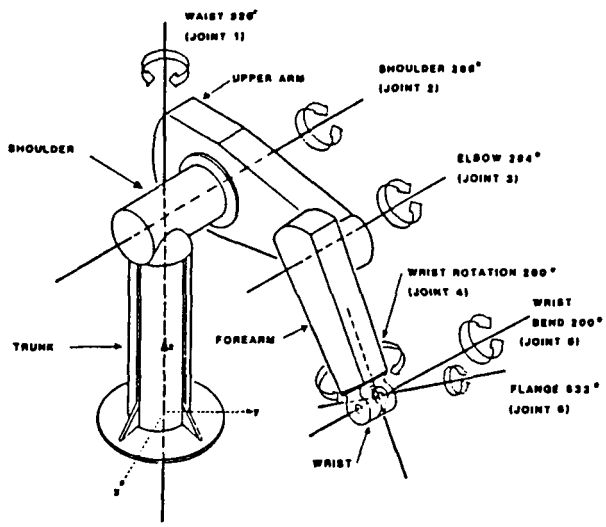


Figure 1. PUMA 560 Robot Arm (Reference 19)

Robot Flexure

In the inertial sensor testing application, ROBSIM was used to characterize arm flexure before performing the sensor tests. The flexure experiment was performed by securing a high-accuracy Systron-Donner 4841F accelerometer to an aluminum mount which was screwed on to the robot tool flange (Figure 1). The flange was rotated 90 degrees from the READY position (Figure 2) to position the input axis of the accelerometer vertical up. From this position the flange was first rotated counterclockwise in ten-degree increments to 90 degrees from vertical and then back to vertical in ten-degree increments. The accelerometer output was stabilized and recorded at each position. The experiment was then repeated in the same configuration but with the flange fixed and the shoulder rotated in ten-degree increments about the base y-axis starting from a vertical position. Shoulder and flange rotation alignment errors were calculated and compared. Larger shoulder rotation alignment errors would indicate flexibility of the robot arm. Performing this experiment on both ROBSIM and the PUMA provided a basis for comparison; the rigid-link model on which ROBSIM was predicated (7,8) aided in identifying actual robot positioning errors (7,8).

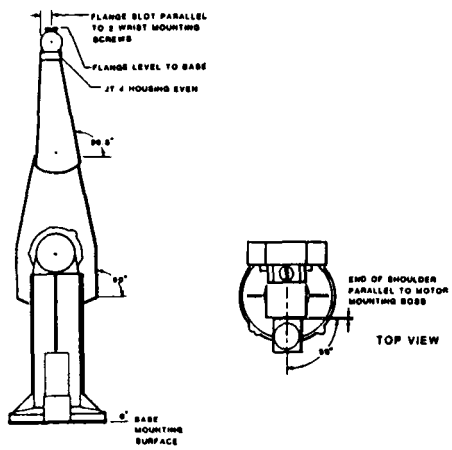


Figure 2. READY Position of PUMA 560 (Reference 20)

Robot Alignment

As with any other test stand, a robot must be calibrated and aligned. To demonstrate the alignment of the robot arm with local vertical, a vertical-seeking test was designed, using the output of the Systron-Donner accelerometer and the PUMA 560's operating system to accomplish the calibration. In an actual testing situation a high-precision accelerometer, a triad of accelerometers, a laser, or some other means could be used either to verify the robot's position or to position it (if its own positioning system were limited). In this demonstration, however, a single accelerometer was used to locate local vertical.

The direction of vertical could be determined by simply maximizing a single accelerometer reading and using a numerical algorithm to zero in on vertical. However, most practical applications are faced with limited numerical accuracy in reading an accelerometer. Because of the non-linear nature of accelerometer reading accuracies, it is more accurate to find two orthogonal horizontal vectors and compute their cross-product to locate vertical. The natural precision geometry of the PUMA 560 manipulator supplies the proper configuration to determine vertical, using wrist bend (Joint 5) in conjunction with a 90-degree rotation in the waist (Joint 1); see Figure 1. For an expanded discussion of the theory behind finding horizontal and the calculations involved in this test, see Reference 10.

Robot Precision

The degree of testing precision achievable with the PUMA 560 Robot Arm was investigated by performing an accelerometer four-point test using the arm as a testbed and the Systron-Donner 4841F as the test item.

The Systron-Donner 4841F accelerometer is a conventional single-axis, pendulous, fluid floated, torque rebalance accelerometer, with an analog output in volts direct current (VDC) proportional to the applied acceleration. For the series of four-point tests, the accelerometer on its aluminum mount was secured to the flange and aligned parallel to local gravity. The pendulous axis (PA) of the accelerometer was aligned parallel to the Y-axis of the tool flange and its input axis (IA) perpendicular to the Y-axis of the tool flange. The robot wrist joint was rotated 90 degrees, followed by a 90 degrees rotation of Joint 5, in order to position the accelerometer IA up and parallel to local vertical (just as in the flexure test). The four-point test was then performed.

The software was designed to rotate the accelerometer to the four positions (by rotating the flange) and allow sufficient time to read the accelerometer output voltage at those positions. This was accomplished by the VAL II operating system DRIVE command.

The accelerometer output was analyzed by calculating and determining the stability of the accelerometer scale factor, 1-g bias, null bias, and misalignment angle, using standard four-point test analysis (see Reference 10). This data was placed in a table and compared to tests of the same type of accelerometer on precision non-robotic test units (21:27).

Robot Adaptability

Robot adaptability was demonstrated by performing a gyroscope (gyro) step-tumble test. This test demonstrated the maneuverability of a robot arm and the ease of reconfiguring the robot for different tests. For the step-tumble test the robot must be positioned to align the gyro's output axis parallel to the earth's rotational axis pointing north and then pointing south. The output of the gyro in these orientations is used to calculate the gyro drift characteristics. (For a thorough discussion of the gyro error model and drift coefficient determinations, see References 10 and 22).

The gyro used for the experiment was a Humphrey Model RG51-0106-1, a conventional single-degree-of-freedom (SDOF) torque-rebalance rate gyroscope. The PUMA 560 Robot Arm was again used as a test platform. The gyro was mounted to a metal support base which was in turn attached to the robot flange. The step-tumble test required the following gyro orientations to separate the drift coefficients for the gyro:

(1) Gyro OA parallel to the earth's spin axis (EA) pointing north, IA pointing west at the start of the rotations (OA//+EA)

(2) OA parallel to EA pointing south, IA pointing west at the start of the rotations (OA//-EA).

To align the gyro with the EA it was first necessary to determine the relationship between the PUMA World Coordinate System (WCS) and the EA. To find the WCS relative to EA it was necessary to know the latitude of the robot and the direction of True North with respect to the robot. This information was readily available for the test site and was used to determine the proper robot joint angles to align the gyro OA with the EA.

Once the OA and IA were properly aligned, the gyro was stepped through 360 degrees of rotation by rotating the flange 360 degrees clockwise (cw) followed by 360 degrees counterclockwise (ccw), pausing at each 45-degree increment to record the gyro output. One cw and ccw rotation of the flange for each orientation constituted one set of data for each step-tumble test. Eight sets of data were collected with OA south and eight with OA north (a total of 128 points in each direction).

The software was written for the robot's VAL II operating system which was accessed through a Zenith 100 (Z-100) running communication software to act as a smart terminal. The programs, written

in the VAL II language, positioned the robot arm for each of the required gyro orientations and rotations.

The statistical package BMDP was used to perform the least squares fit of the output voltage to the gyro model. The drift coefficients calculated from the fit were then summarized in tabular form.

RESULTS OF EXPERIMENT

Results of the experiment demonstrated both the advantages and the current limitations of robotic testbeds and simulations.

Robot Control and Alignment

The results of the flexure test showed larger shoulder rotation alignment errors than flange alignment errors when the position was 30 degrees to 90 degrees from vertical. The accelerometer outputs demonstrated the inaccuracies of robot positioning and indicated that the flexibility of the robot arm should be a consideration when precise positioning and orientation is needed. A plot of the actuator torque versus time for the shoulder rotation as generated by ROBSIM showed that the torque is a function of the robot orientation and that the orientation errors are due in part to mechanical flexure.

Robot control is also limited by control method and unmodelled forces, and by the restrictions of robotic programming languages. The most widely used control method today applies a separate axial control loop for each joint designed with linear-control laws (12:80), often with fixed gain (12:72). The required gain is highly dependent on the moment of inertia at each joint of the robot arm which in turn varies with the arm position and robot payload. A variety of schemes, including adaptive control, have been proposed and implemented (12:51-81), but research is still being done to represent previously unmodelled forces (13) and implement adaptive control.

Robot programming languages, too, can be a control limitation in that they often do not include the facilities to implement complex mathematical formulas. One must bypass the robot operating system to implement experimental techniques and gain greater precision.

The theory and analysis involved in performing the alignment (vertical-seeking) test presumed no robot joint positioning errors. There are, however, small accumulated errors via quantization of robot movement and calculations by the robot arm controller (19). No attempt was made to include these errors in the vertical-seeking algorithm. The algorithm did, however, locate vertical more precisely than could be done by simply placing the arm in the READY position, or by using a single accelerometer output determination.

Robot Precision and Adaptability

The results of the four-point tests in the following table (Table I) show that positioning precision can be achieved. Although the performance characteristic values are larger than those derived from four-point tests of similar instruments (see Table 2.3 from 21:27-28), the standard deviations and peak-to-peak spread are comparable. The laboratory environment for this research was much less controlled than that of a test facility such as CIGTF; noise sources from the laboratory and perhaps from the robot arm itself, and lack of temperature control contributed to the magnitude of the coefficients. However, the stability of the outputs is an indication of the positioning repeatability of the robot arm.

Table II summarizes the drift coefficients (and their standard error) of the performance model gyro equation. Since the duration of the tests was approximately three hours and the gyro's output axis was aligned with the earth's rotational axis, error sources did not include earth rate. All drift coefficients except D(O) were significant. From previous rate-table tests D(F) was determined to be 1.5 volts. Except for D(F), there was no test data with which to compare the drift coefficients. However, the coefficients are reasonable and, as with the accelerometer four-point test, indicated the feasibility of using the robot arm for testing inertial sensors.

The main purpose of the gyro test was to demonstrate the robot arm's ease of reconfigurability and its maneuverability and therefore its usefulness as a multi-purpose testbed. This was clearly demonstrated by the gyro step-tumble test.

Establishing testing feasibility using the PUMA 560 then led to determining a general set of robot criteria for the inertial sensor application, including economic considerations.

ECONOMIC ANALYSIS OF APPROACH

All the criteria for selecting a robot for industrial applications are fully described in the robotics literature (6:214-301; 12:263-272; 15). In this study we were addressing only the criteria pertinent to inertial sensor/system testing. A summary of the criteria is as follows:

(1) Load requirement - 5 to 25 pounds
(2) Drive method - Electric motor driven
(3) Number of axes - 6
(4) Axis rotation - Wrist pitch, roll, or yaw of at least 360 degrees; at least 180 degrees of rotation in other joints
(5) Off-line programming capability
(6) Repeatability of 0.010 inches or less
(7) Variable acceleration/deceleration desirable
(8) Floor mount.

An expanded discussion of these selection criteria is found in Reference 10.

Table I
Accelerometer Performance Characteristics from Four-point Tests

	Scale Factor (volts/g)	1-g Bias (μg)	Null Bias (μg)	Misalign (arcsec)
ON ROBOT ARM:				
Mean	1.018805	1207	1720	8154
Standard Deviation (ppm)	29	60	66	9
Peak-to-peak Variation	115	241	255	30
ON VERTICAL TABLE (21:27):				
Mean	0.02493	184.5	148.4	-30.6
Standard Deviation (ppm)	40	45.8	36.4	7.5
Peak-to-peak Variation (ppm) *	471	471	244	58

* Over 39 days. No data available for a single day's testing.

Table II
Performance Model Equation Coefficients

Drift Coefficient	Calculated Value	Standard Error
D_F	1.49999	0.00188
D_I	0.00249	0.00031
D_S	0.07619	0.00031
D_O	0.00188	0.00295
D_{IS}	0.00117	0.00035
D_{II}	0.00107	0.00035
D_{SS}	0.00107	0.00035
D_{OI}	0.00389	0.00036
D_{OS}	0.00120	0.00036

A comprehensive listing of prospective robots containing their physical characteristics and estimated base prices was obtained (18) using a commercial computer package called "Robot Search Program" (Robot Analysis Associates, Inc.). This list was reduced to four robots by entering the data into a spreadsheet (Lotus 1-2-3) and using the spreadsheet's capabilities to highlight the manipulators with the maximum performance capabilities (5:435-448). The results are summarized in Table III, along with non-robotic test tables.

The non-robotic tables have the advantage of continuous rotation and accuracies in the arcseconds range. However, the load capabilities are comparable, including the 100-pound load. For example, in addition to the robots listed, the Cincinnati Milacron T3-776 meets the rotational and accuracy requirements while carrying a load of 150 pounds. The robotic testbeds, however, are more versatile and less expensive and have other potentials discussed in the conclusions section.

Life cycle costing (LCC) over a 5-year period was the tool used to determine economic feasibility (9:66-67; 1:20; 2). Research and development costs, investment costs, and operational costs were included for the analysis. Table IV summarizes the results for both the selected robots and the non-robotic tables.

From the economic analysis it is feasible that a prototype robotic test station, the T3-646 for instance, could replace one table, perhaps the vertical table, with a resultant decrease in LCC of $17,364. Of course the savings increase

Another important advantage and source of savings is the versatility of a robot arm. Over the long term both standard and experimental inertial instrument tests can be performed by simply re-programming the robot, rather than rebuilding or developing a new test table. In the short term, as was the case for the gyro tests, the robot can be quickly reconfigured at any point in the test with no manual readjustments involved.

CONCLUSIONS AND RECOMMENDATIONS

In an attempt to control robots more precisely and to interface with computers (and computer simulations) other than the robot's particular controller, research is in progress to control robots from computers such as the VAX 11-780 (AFIT, NASA Langley) or interface with such computers for control and data acquisition (for example, Cincinnati Milacron's Robot Offline Programming System, or ROPS).

From the facility development study presented here, one can conclude that robots large and small could begin to be used as checkout testbeds for inertial sensors, possible in such applications as immediate flightline checkout of sensors or inertial measurement units (IMU's) suspected of being inoperable rather than sending them away to a depot for checkout.

Robots can be multi-purpose testbeds for performing standard tests on inertial sensors, and the potential for devising unique inertial sensor/system tests exists. Robots with variable acceleration/deceleration and a large rotational range suggest dynamic test possibilities that have not yet been explored. Perhaps subjecting the sensor/system to a helical motion, or to a rapid swinging motion of the robot followed by a sudden deceleration would excite sensor/system error

Table III
Performance Characteristics and Base Prices of Robotic and Non-Robotic Testing Units

Name	Mount	Max Rot (Wrist)	Other Rot	Joint	Max Load (lbs)	Accuracy (ins)	At (ips)	Variable Accel/Decel	Base Price
A'matix AID-900	Floor	440	p315	Wrist	66	0.008	30	Y	50000
Yaskawa	F/O/W	360	YR330	Wrist	26	0.008	80	Y	69600
Cinn Mil T3-646	Floor	900	PY238	Wrist	50	0.010	25	N/A	70000
PUMA 560	Floor	532	P200	Wrist	5.5	0.004	20	N	80000

Name	Mount	Max Rot (Wrist)	Other Rot	Joint	Max Load (lbs)	Accuracy (arcsec per axis)	At (ips)	Variable Accel/Decel	Base Price
Vertical Table	Floor	Contin.			50	< 1		N	150000
2-axis Contraves	Floor	Contin.			75	1		N	500000
3-axis Contraves	Floor	Contin.			100	# 3		N	## 3000000

Difference in accuracy due to different type of bearings, not number of axes
Estimated cost of new 3-axis table

Table IV
Total Life Cycle Costs

Device	LCC
Automatix AID-900	$ 146,279
Cincinnati Milacron T3-646	186,618
Yaskawa V-12	185,811
PUMA 560	206,787
Vertical Table	203,982
2-axis Contraves	522,239
3-axis Contraves	2,818,062

terms and thus enhance or replace centrifuge or other testing. Variations of system trajectories could be tracked with lasers and the system errors analyzed by comparison with the laser position data. With extensive computer simulation capabilities such as those of ROBSIM, engineering theory could devise new tests which would be efficiently and safely produced on the simulator, saving both time and money. The simulator-robot combination would encourage engineering creativity, an important commodity in the realm of research and development, where new tests and testing units are needed to keep pace with hardware developments (2).

This study raises further questions. Are robots feasible for system tests? Can the limitations be overcome? What should be done to extend the work presented here?

The solution for robot accuracy constraints may lie not in improving the robot's precision, but rather in providing precision reference measurements for use in sensor output analysis. Laser technology and other instrumentation advances have the potential to accomplish this. For example, providing precision through reference measurement is already in use in noisy, imprecise environments such as the test track at Holloman Air Force Base; and laser technology is currently being used for robot positioning accuracy (14). A cost analysis for laser or other precision measurement technology should be accomplished to extend the economic feasibility study.

The potential for testing precision sensors/systems should be further determined by noise characterization of the robot arm. In addition, the sensors used in this study, or similar sensors, should be tested under more controlled laboratory conditions and compared to test results from non-robotic units.

It is also recommended that test engineers and analysts take a new look at the possibilities for dynamic tests using robotic capabilities and begin devising those tests. The groundwork for a prototype effort has been presented in this study and is recommended for future implementation.

REFERENCES

1. Abundis, M., "Making Robots Count," ROBOTICS ENGINEERING, December 1986, pp. 17-20.

2. Alexander, R., Telephone interview, Central Inertial Guidance Test Facility (CIGTF), Holloman AFB, New Mexico, April 1987.

3. Black, A. J., A PROGRAMMER'S ASSISTANT FOR A SPECIAL-PURPOSE DATAFLOW LANGUAGE, MS Thesis, AFIT/GCS/ENG/85D-2, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, December 1985.

4. Bustle, A., Telephone interview, Central Inertial Guidance Test Facility (CIGTF), Holloman AFB, NM, January, 1987.

5. Cobb, D., DOUGLAS COBB'S 1-2-3 HANDBOOK, Bantam Books, New York, 1986.

6. Critchlow, A. J., INTRODUCTION TO ROBOTICS, Macmillan Publishing Company, New York, 1985.

7. "Evaluation of Automated Decisionmaking Methodologies and Development of an Integrated Robotic System Simulation," NASA CONTRACTOR REPORT 172401, Martin Marietta Aerospace, Denver CO, September 1984.

8. "Evaluation of Automated Decisionmaking Methodologies and Development of an Integrated Robotic System Simulation," NASA CONTRACTOR REPORT 178050, Martin Marietta Aerospace, Denver, CO, March 1986.

9. Fisher, G. H., COST CONSIDERATIONS IN SYSTEMS ANALYSIS, American Elsevier Publishing Co., Inc., New York, 1971.

10. Greig, J. Y., ROBOTICS APPLICATIONS FOR THE TESTING OF INERTIAL SENSORS, MS Thesis, AFIT/GE/ENG/87J-4, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH, June 1987.

11. Greig, J. Y., D. A. Karnick, D. J. Biezad, and G. B. Lamont, "Implementing and Testing a Robot Simulator," PROCEEDINGS OF THE NORTH CENTRAL AND INDIANA-ILLINOIS SECTIONS OF THE ASEE, Purdue University, West Lafayette, IN, 1986, pp. 404-411.

12. Koren, Y., ROBOTICS FOR ENGINEERS, McGraw-Hill Book Company, New York, 1984.

13. Leahy, M. B. Jr. and G. N. Saridis, "Compensation of Unmodeled PUMA Manipulator Dynamics," PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION, Raleigh, NC, March 30-April 3, 1987, pp. 151-156.

14. Montalbano, G. and R. Schlais, "Laser Guided Precision Positioning With A Gantry Robot," ROBOTICS ENGINEERING, Vol. 8, February 1986, pp. 23-25.

15. Nof, S. Y. editor, HANDBOOK OF INDUSTRIAL ROBOTICS, John Wiley & Sons, New York, 1985.

16. O'Toole, T. and M. Cetron, "The Coming Trend of Robots," Press interview, 1984.

17. Pai, A. L. and J. W. Pan, "A Computer Graphics Simulation System for Robot Manipulator Kinematics," PROCEEDINGS OF THE PHOENIX CONFERENCE ON COMPUTERS AND COMMUNICATIONS, March 1985, pp. 237-241.

18. Simmons, W., Rohr Industries, Inc., Riverside, CA, February 1987, personal correspondence.

19. Unimation, Inc., UNIMATE INDUSTRIAL ROBOT USER'S GUIDE TO VAL II, PART 1, VERSION 1.4B, Danbury, CT, 1983.

20. Unimation, Inc., UNIMATE INDUSTRIAL ROBOTS AND CARTESIAN COORDINATES, Danbury, CT, 1983.

21. Laboratory Tests of Systron-Donner 4841F Accelerometer Conducted by the Central Inertial Guidance Test Facility (CIGTF), REPORT NO. AD-TR-81-13, Holloman AFB, NM, CIGTF, 6585th Test Group, January 1981.

22. Wrigley, W., W. Hollister, and W. Denhard, GYROSCOPIC THEORY, DESIGN AND INSTRUMENTATION, The M.I.T. Press, Cambridge, MA, 1969.

Space Station Assembly: Working with Robotics

(Paper not written)

TELEROBOTIC TRUSS ASSEMBLY

Philip L. Sheridan
NASA/JSC, ES63
Houston, TX 77058

SUMMARY

The Structures and Mechanics Division (SMD) at Johnson Space Center telerobotically assembled the ACCESS truss. The SMD wanted to assemble hardware that was designed for and been assembled by EVA astronauts. Many problems were identified. Most belong in one of three main categories.

1. Truss Hardware
2. Manipulator
3. Vision

The tight alignment constraints of the ACCESS hardware made telerobotic assembly difficult. A wider alignment envelope and a compliant end effector would have reduced this problem.

The manipulator used had no linear motion capability, but many of the assembly operations required straight line motion. The manipulator was attached to a motion table in order to provide the X, Y, and Z translations needed. A programmable robot with linear translation capability would have eliminated the need for the motion table and streamlined the assembly.

Poor depth perception was a major problem. Shaded paint schemes and alignment lines were helpful in reducing this problem. The four cameras used worked well for only some operations. We were unable to identify individual camera locations that worked well for all of the assembly steps. More cameras or movable cameras would have simplified some operations.

The audio feedback system was useful. Often the first indication that a strut made contact with a node was an audio signal rather than a video one. Also, if a strut inadvertently hit something in the workcell, the operator was alerted.

Many of the lessons learned will be used to design robot friendly hardware and to define tasks suitable for a space telerobot.

INTRODUCTION

During the summer of 1987, the Structures and Mechanics Division (SMD) at NASA's Johnson Space Center conducted telerobotic truss assembly tests. These tests had four objectives.

1. Identify problems with the telerobotic assembly of hardware designed for EVA astronauts.
2. Evaluate an audio feedback system.
3. Demonstrate simplified remote manipulator system (RMS) dynamics.*
4. Establish an experience base for the development of robot friendly hardware/tasks.

The Assembly Concept for Construction of Erectable Space Structures (ACCESS) hardware was selected for these tests. The ACCESS truss has been tested in space and the Weightless Environment Training Facility (WETF) and the struts and nodes were small enough for the Deep Ocean Engineering (DOE) manipulator to handle.

The problems encountered and their solutions as well as our evaluation of the audio feedback system are discussed.

CONCLUSION

Assembling the ACCESS hardware showed how difficult it is for a telerobot to handle hardware designed for astronauts. In order for robots to efficiently help build and maintain Space Station, the hardware must be designed to be robot friendly. Shaded paint schemes and alignment lines partially made up for the loss of depth perception caused by the video system. Less rigid alignment constraints and compliant end effectors will reduce misalignment problems. Audio feedback increased operator awareness and should be included in future telerobotic experiments. The experience gained from this experiment

*This phase of the test was not completed in time for this publication.

will help in the design of robot friendly hardware and will help with the identification, testing and implementation of tasks suitable for a space telerobot.

DISCUSSION

For this test program there were seven major pieces of hardware. Figure 1 shows the layout of the hardware described below.

Hardware Description

1. 6-DOF Table

The six degree of freedom (6-DOF) table consists of a triangular shaped active table and six linear hydraulic actuators. The actuators provide the 6-DOF table with X, Y, and Z translation, Roll, Pitch, and Yaw. For this test, Roll, Pitch, and Yaw were not used.

2. DOE Arm

The Deep Ocean Engineering (DOE) arm is an electrohydraulic manipulator arm that was used to telerobotically assemble the ACCESS nodes and struts into a truss.

3. Linear Translation/Load Cell Table

The Linear Translation/Load Cell Table (LTLCT) was attached to the 6-DOF table through a set of load cells. The load cells will send load data to the computer for force and moment resolution during the dynamic simulation demonstration. The LTLCT supported the DOE arm and provided additional X-axis translation.

4. Mast

The mast was a mounting fixture to which the ACCESS nodes and struts were attached during the truss assembly.

5. Strong Back

The strong back is a fixed structure to which the mast and node/strut rack were mounted.

6. Node/Strut Rack

The node/strut rack stored the ACCESS nodes and struts before they were grasped by the DOE arm and assembled into a truss.

7. Video System

Television cameras were used by the teleoperator to monitor the truss assembly. They were mounted 1) to the left of the DOE arm, 2) on the ground below the mast, 3) behind the DOE arm on the 6-DOF table, and 4) on the DOE arm. Cameras 1, 2, and 3 had pan, tilt, and zoom capabilities. Camera 4 had a fixed view of the gripper.

Assembly Sequence

The top three nodes were manually mounted to the mast before beginning the demonstration just as they were for the WETF tests. The first step in the assembly process was to remove the fourth node from the node/strut rack and place it on the mast. This step was repeated until all six nodes were in place. Each of the 12 struts was connected using the following sequence. One strut collar was preset to the automatic position and the other collar was preset to the manual position. The strut was positioned between two nodes and linearly driven into place using the LTLCT. The automatic collar closed locking the strut onto one node. The DOE arm was then positioned near the manual collar where, using a friction pad, the collar was rotated to secure that end of the strut to the other node. The bottom three horizontal struts were connected, one at a time, to the nodes (refer to Figure 2). The top three horizontal struts were connected next, then the three vertical struts, and finally the three diagonal struts.

The initial assembly time was four hours for one bay. Before the second assembly several changes were made to the workcell shown in Figure 1. Camera 1 was moved 10 feet in the -X direction and four feet in -Y. This location provided a better view from which to see strut to node alignment. Camera 2 was repositioned four feet in the -X direction and one foot in -Y. Moving four feet in the -X direction made strut to node alignment easier to determine for the horizontal and diagonal struts. Moving one foot in the -Y direction made vertical strut insertions easier to see. The node/strut rack was rotated 90^{o}. This change decreased the wrist roll actions by nearly one half. A combination of these modifications and an increase in operator proficiency reduced the assembly time to two hours.

One of the major time consumers during the assembly was the 6-DOF table. Vibrations from the 6-DOF table's hydraulic pumps caused the DOE arm to vibrate at approximately 5 Hz. This vibration made the assembly process very difficult. Also, the 6-DOF table moves very slowly and approximately one half of the two hour assembly time was needed to move the DOE arm from the node/strut rack to a position near the mast where the insertion action could begin.

Low pass filters were put into the 6-DOF tables control system to eliminate the vibrations. The smooth operation made strut insertions easier and speeded up the movement of the table. This did not decrease the assembly time because the automatic collars would sometimes not close. Apparently the vibrations helped overcome some of the binding between the struts and the nodes which helped close the collars.

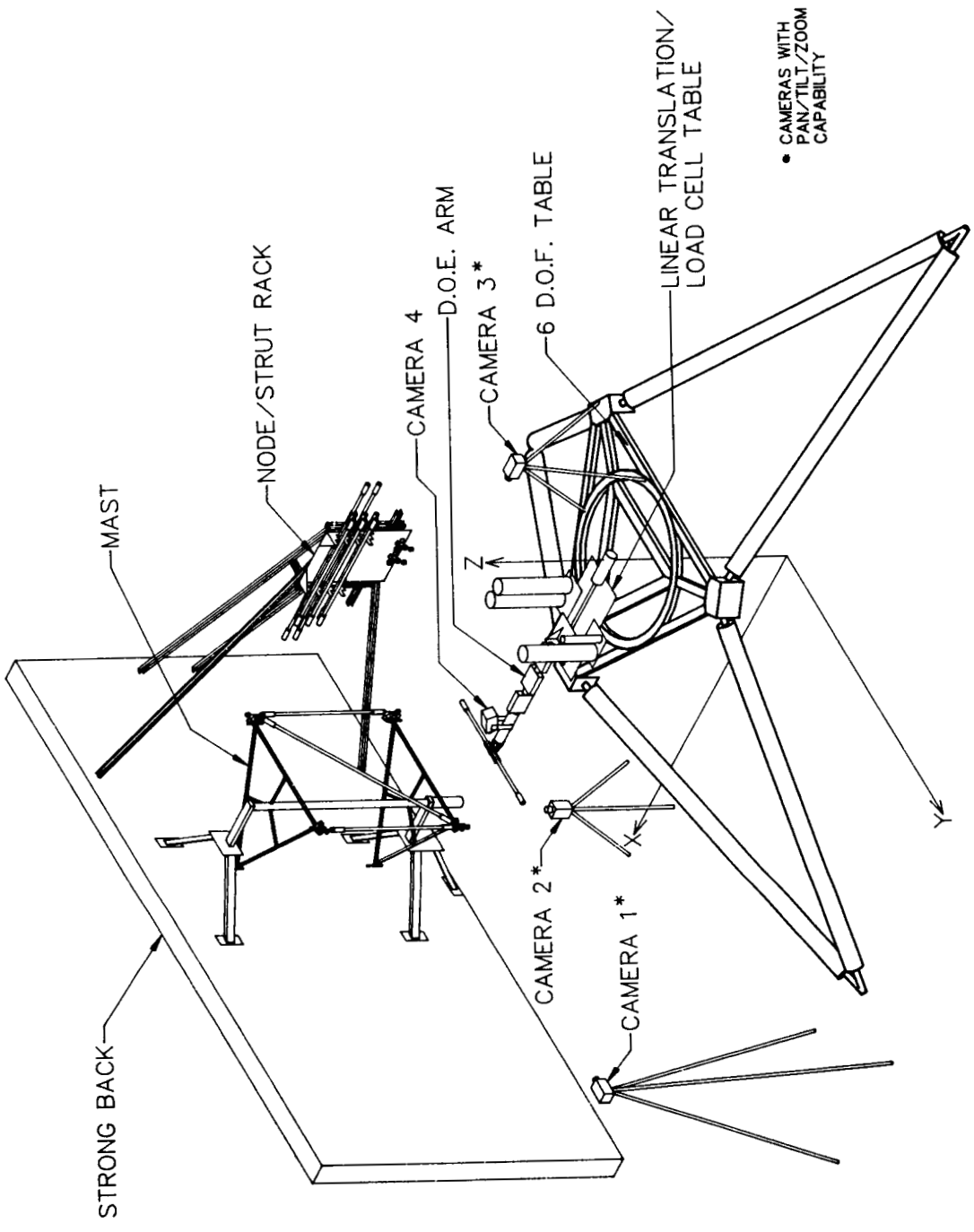


FIGURE 1. ATAD LAYOUT

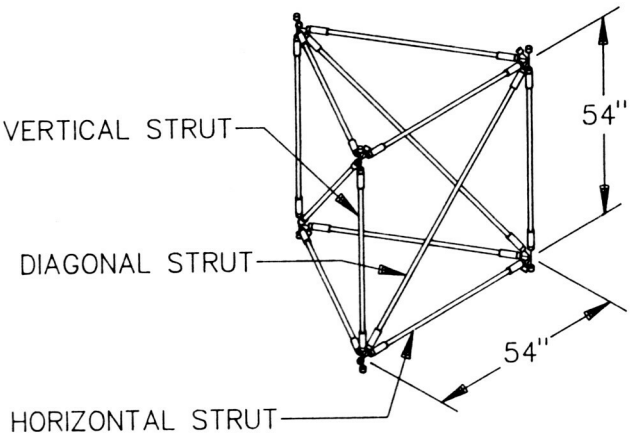


FIGURE 2. ACCESS TRUSS

Problems and Solutions

The problems we had belong in one of three main groups.

1. Truss Hardware

The ACCESS hardware has tight alignment constraints. Figure 3 shows a strut partially inserted into a node. If these two components are not perfectly aligned the strut cannot be locked into place. Larger alignment envelopes, guides, and a compliant end effector would have reduced misalignment problems.

2. Manipulator

The DOE manipulator has:

a. No linear motion.

b. No automation.

c. No joint position indicator.

Most of the operations during this test were completed using the linear motion provided by the 6-DOF table or the LTLCT. The movement of the 6-DOF table was very slow. A robot with linear motion would have eliminated

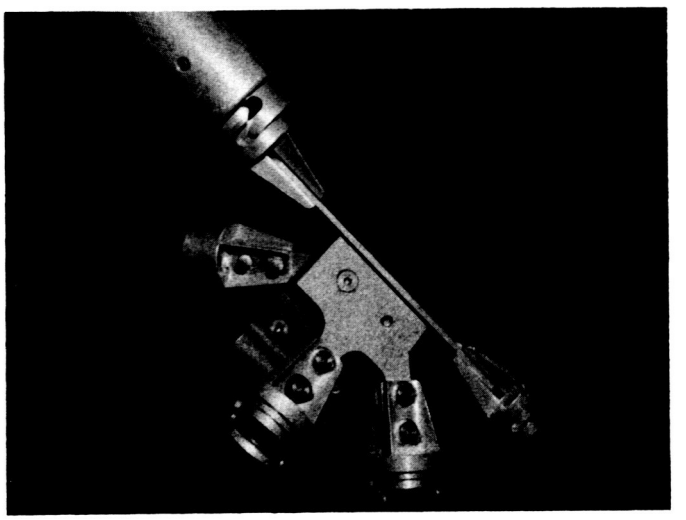


FIGURE 3. Strut Partially Inserted Into Node

the need for the 6-DOF table, and decreased the assembly time. A semiautomated operation would have been more efficient than a fully manual one. Preprogrammed manipulator positions would have been helpful, but because of variations in the ACCESS hardware and mounting fixture positions the actual strut/node connection was necessarily teleoperated. Many alignment problems were partially caused by an inability to put the DOE arm's joints into exact, known positions. Joint position indicators would have lessened the alignment problems.

3. Vision

Normal video equipment greatly reduces depth perception. Stereo vision may help with this problem. Shaded paint and alignment lines on the truss hardware partially compensated for the lack of depth perception.

Some of the camera positions used were very good for some operations and very poor for others. Adjustable (X, Y, Z) camera positions, or more cameras would have decreased the assembly time. Camera #4 did not have pan, tilt or zoom capabilities. These features would have been useful.

The ACCESS hardware and most of the fixtures in the workcell were bare aluminum. The reflected light caused glare which washed out some detail. Painting some of the fixtures flat black greatly reduced this problem. Anti-glare paint should be used for everything in the workcell.

Audio Feedback

The DOE arm has an accelerometer in its forearm that sends signals through a control unit to a headset worn by the operator. During this test the linear drive motor, collars closing, and node/strut contact were heard. The sounds heard through the headset were very similar to the actual sounds. The first two types of sounds made the operator feel closer to the workcell. The node/strut contact noise was very useful. It partially made up for the lack of depth perception. Often the contact was detected through audio feedback before it was detected visually.

ACKNOWLEDGEMENT

I would like to acknowledge Scott Askew, George Parma, and LeBarian Stokes. They made major contributions to this project and their advice was helpful in the preparation of this paper. I would also like to thank the Robotics lab and 6-DOF table support personnel.

CREW INTERFACE WITH A TELEROBOTIC CONTROL STATION

By

Eva Mok
Rockwell International Corporation
Space Station Systems Division
12214 Lakewood Blvd.
Department 590—Mail Code AE91
Downey, CA 90241
(213) 922-0912

ABSTRACT

A method for apportioning crew-telerobot tasks has been derived to facilitate the design of a crew-friendly telerobot control station. To identify the most appropriate state-of-the-art hardware for the control station, task apportionment must first be conducted to identify if an astronaut or a telerobot is best to execute the task and which displays and controls are required for monitoring and performance. Basic steps that comprise the task analysis process are (1) identify Space Station tasks, (2) define tasks, (3) define task performance criteria and perform task apportionment, (4) verify task apportionment, (5) generate control station requirements, (6) develop design concepts to meet requirements, and (7) test and verify design concepts.

INTRODUCTION

Designing an efficient man-machine interface is of high importance to Space Station telerobotic operations. It is estimated that the cost ratio of astronaut time spent on extravehicular activity (EVA) versus intravehicular activity (IVA) is 4 to 1. Therefore, means to increase the productivity of a crew member are important. The use of telerobotics presents a feasible way to reduce crew EVA hours. Tasks best performed by telerobots must first be identified to achieve an efficient man-machine interface. Requirements are then generated to guide the design of the control station, which provides the interface between the crew person and telerobot. This paper discusses a method and verification techniques for apportioning crew and robot tasks in the most effective way. Figure 1 is a flow diagram of our task apportionment and verification process.

Space-based crews will be working with highly automated and sophisticated telerobot systems. Interfaces between the crew and the system will have to be crew-friendly, whereby productivity and flexibility are increased, reliability is improved, and little or no recurrent training is required. The ideal design for a teleoperator control station provides displays and controls that are transparent to the operators to simulate their presence at the remote site. The operators can then pay full attention to the task without being distracted by remoteness. To create this type of environment, tasks must be apportioned between the crew and telerobot relative to their capabilities and limitations.

IDENTIFY SPACE STATION TASKS

The first step is to identify the types of tasks that will be performed on the Space Station. We are supplementing our data base on future Space Station tasks with expert opinion from astronaut consultation; data from Soviet missions; and our past experience in the Shuttle, Space Lab, and Sky Lab programs.

Define Task

This step defines a task in terms of how it is performed (Reference 1):

- Are tools required?
- Where is the task performed?
- Is it time critical?
- Does it require more than one operator?
- How complex is it?
- How frequently is it performed?
- Is the operator required to make frequent decisions?
- Is it concurrent with other tasks?

To identify the subtasks required to complete the task, we first develop scenarios of the step-by-step process by which the task is performed. A comprehensive literature review is conducted to aid in deriving the steps involved in performing the task. The literature reviewed comes from such resources as NASA requirements and procedures documents, as well as related literature from military and academic sources and from nuclear and other industries that use telerobots.

Define Performance Alternative

This step defines the limitations and capabilities of each performance alternative. The possible alternatives, or combinations of alternatives, for performing Space Station tasks include crew IVA, crew EVA, telerobotics, automated systems, and ground control. The following paragraphs give a more detailed discussion of these alternatives.

IVA crew performance is preferred with tasks that require supervisory control, learning, critical and quick decision making, and memorization (References 2, 3, and 4). Crew effectiveness is limited when tasks are tedious, have time constraints, and require extensive and immediate information processing. For instance, scheduled subsystem monitoring or subsystem checkouts do not make good use of the crew. Crew time is critical because of the small number of crew members available on the Space Station and the heavy work schedules they must meet every day. It is neither efficient nor pleasant to have the crew perform time consuming, repetitive, and unstimulating tasks. In such cases, it is better to have an expert system monitor and control subsystems and then interact conversationally with the crew. System status and anomalous situations should be reported to the crew through a conversational natural language interface, i.e., voice communication and graphic displays (Reference 4). Then the crew members can use their expertise to decide which action should be taken next.

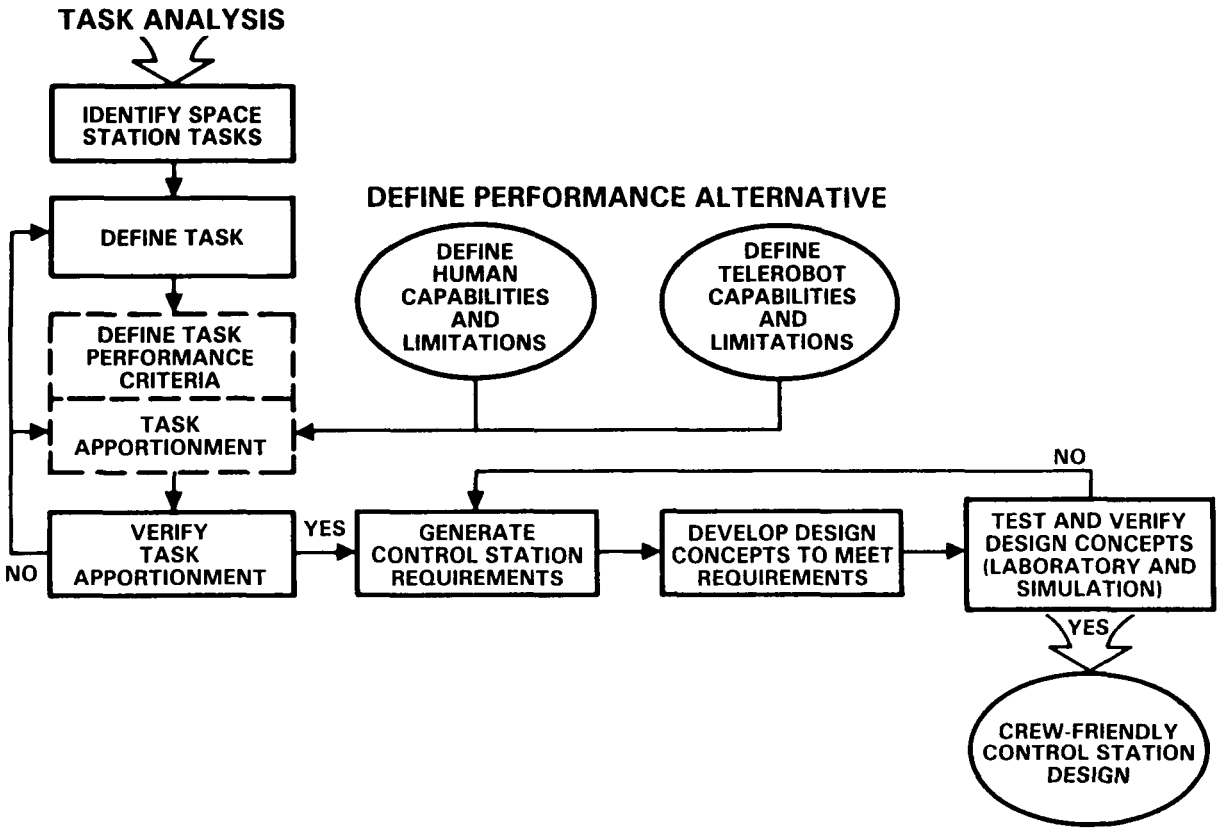


Figure 1. Method for Apportioning Space Station Tasks to Ensure Crew-Friendly Telerobotic Control Station Design

EVA crew performance capabilities are more limited than are those of the IVA crew. EVA is inherently more stressful than IVA because of the novelty and danger of the space environment and the limited duration of the space suit life support system. The physical difference between IVA and EVA is that the EVA crew wears the extravehicular mobility unit (EMU) or space suit. The constraints imposed by the EMU are time constraints, reach envelope, and dexterity limitations. Safety is one of our key drivers for reducing crew EVA. An EVA task that is a good candidate for telerobotic operations is the exchange of an orbital replacement unit (ORU) (Reference 3). Space Station apparatus is designed in a modular fashion. An ORU is the minimum-sized unit; if any part of an ORU fails, the entire unit is replaced. When the ORU can be located for easy access, and its removal and replacement involve simple standardized procedures, a robot can be used to perform the changeout. However, if an ORU is located where access is complicated, such as inside a housing requiring access through a panel opening, an EVA astronaut must perform the task.

Telerobot performance limitations and capabilities are related to end effector access, precision of movement, degrees of freedom, and effectiveness of IVA remote control. An outline of telerobot characteristics and requirements is shown in Table I. Robots and their associated computer systems tend to be more efficient than humans at continuous monitoring, repetitive tasks, storing and recalling large amounts of data in a short period of time, ignoring distraction, and resisting tedium or boredom. On the other hand, humans are best at using their intelligence at perceiving, understanding, continually refining what needs to be done on the basis of what has been learned, and solving unforeseen problems. Robots, however, can be given perceptual abilities outside the range of human capabilities, such as responding to radiant signals beyond the limits of human vision. They can work in the dangerous space environment and handle substances that pose unacceptable hazards for humans (References 3 and 4).

Table I. Summary of Space Station Telerobot Manipulator Characteristics and Requirements

Characteristics	Requirements
Degrees of freedom	• Operating arm = 7 • Stabilizing arm = 3
End effectors	Several types, interchangeable
Maximum tip force	225 N (50 lb)
Maximum tip speed	1 m/s (40 in./s)
Force sensitivity	2.5 N (0.5 lb)
Stiffness/compliance	• Adjustable by software control • Maximum 90 N/cm (50 lb/in.) • Minimum 2 N/cm (1 lb/in.)
Modes of control	• Full master-slave • Robotic control • Autonomous or supervisory control
Slave power required	300-W peak, 25-W standby
Arm length/reach	~ 1.5 m (60 in.)
Typical arm motion angles	± 45 deg
Typical wrist motion angles	Up to ± 180 deg, except wrist roll continuous
Master-slave signal transmission rate	200 kBd

Automation will play a major role in the success of the Space Station. The primary rationale for implementing automation on the Space Station is that it will increase crew productivity

(Reference 4). However, there are limitations to automating the Space Station. For instance, automated equipment cannot detect changes that lie just outside its programmed range, cannot make unusual decisions, and cannot correct mistakes. Automation will be implemented to relieve the crew from knowledge of detailed procedures for setting up and operating special equipment. Automation in the form of expert systems can provide higher order intelligence for assistance with planning, scheduling, monitoring, control, and fault management (References 3 and 4).

Ground control and support of the Space Station will always be essential, especially in the early stage of the program. As the space program matures, the goal is to minimize ground involvement with day-to-day operations. Initial ground control and support will consist of flight and system monitoring and assistance during the deployment, assembly, activation, check out, and verification of each new Space Station element (Reference 4).

DEFINE TASK PERFORMANCE CRITERIA AND PERFORM TASK APPORTIONMENT

The next step is to establish a set of weighted criteria describing the relative importance of task parameters. For example, reliability might be more important and, therefore, weighted higher than time to perform the task. Task analysis data, alternative definitions, and performance criteria are combined and entered into the Rockwell-developed analytical hierarchical process (AHP) for a hierarchical ranking of how each alternative satisfies the performance criteria. For instance, the robot may be the most reliable alternative, but also the slowest; the EVA crew might perform the task quickly, but at high risk.

Verify Task Apportionment

The identified tasks are simulated in our laboratories and test beds to verify apportionment decisions. Rockwell's Simulation and Systems Test Laboratory was outfitted to verify the optimal hardware to create an efficient crew-friendly control station. Rockwell's software has been used extensively for preliminary design of control stations, hardware placement verification, and crew integration. Table II is a list of control station prototype hardware used to perform station operations analyses in the laboratory. The scenarios developed for the task analysis steps are simulated and displayed at the simulator control station. Astronaut consultants, in-house and team member experts, and data from past programs augment this analytic process. Trade studies are performed to evaluate hardware cost effectiveness.

Generate Control Station Requirements

Data obtained from the task analyses will help derive man-machine interface requirements for the control station design so the crew can effectively control robotic and automated systems and monitor the tasks. Such requirements must refer to the following (Reference 2):

- Displays for performing a task
- Kinds of information for processing while the crew performs or supports the task
- Controls for executing the task
- Equipment (e.g., reliable, maintainable, and safe to operate)
- Control station surroundings (e.g., lighting, noise, traffic flow) that must enhance productivity
- Demands imposed by the control station on other systems

Figure 2 shows a requirements tree for the control station outfitting. For example, the requirements state that the station must provide controls from moving a telerobot around, operating the 6-degree-of-freedom (DOF) arms, end effectors, all displays, cameras, lights, other remote sensors, alarm acknowledgement, and all communication (References 5 and 6). The requirements generation is continued to a sufficient level of detail that individual pieces of equipment, with specific volumes and weights, can be stated as meeting the detailed requirements.

Table II. Prototype Hardware Is Used To Perform Control Station Operations Analyses

Hardware	Components
Work station local area network (LAN)	PC/AT processor with gateway to lab LAN
CRT displays	Color (digital-analog) and monochrome (B/W-green-amber)
Flat panel displays	Plasma (24 in.-17 in.-10 in.), TFEL (10 in.), liquid crystal display (10 in.)
Control devices	Keyboards, touch pads, touch screens, trackballs, joysticks, mouse, digitizer tablets, hand controls
Voice systems	Recognition/synthesis
Data storage	Tape, VCR, optical disc
Purchased software (S/W)	Operator system, compilers, graphics devices, utilities, S/W drivers, data bases
Video equipment	Color cameras, video switchers, converters, frame grabbers, stereo TV systems
Miscellaneous	Projection TV system (6-ft screen), portable computers (grid, panasonic), printers and plotters (color and black/white), image scanners

Develop Design Concepts

Two types of control station concepts have been selected from the above analysis. One concept houses the control station in a cylindrical module called a resource node (Reference 6). Because direct viewing is not possible from within this structure, the crew will have to rely on television viewing of telerobotic tasks. This is not necessarily a disadvantage, because direct viewing may be confusing when orientation of the telerobot is different from the operator's.

The second concept has the control station housed in the cupola, a glass dome-like structure with 360 deg direct viewing of external activities and telerobot activities (Reference 6). The crew uses displays and controls located inside the cupola to interface with the telerobot. In addition, the cupola allows the operator the option of relying not only on electronic displays (i.e., TV, graphics), but on his or her own advanced and versatile viewing and control system—the human eye/brain network. The cupola is preferred by most astronauts because it allows them to manage a task by direct view and to observe the solar system and Earth.

Test and Verify Design Concepts

As concepts are developed, they are subjected to the same test and verification procedures as those used in verifying task apportionment. Thus, by continually refining the design concepts, we move by successive approximation to the evolution of a final design.

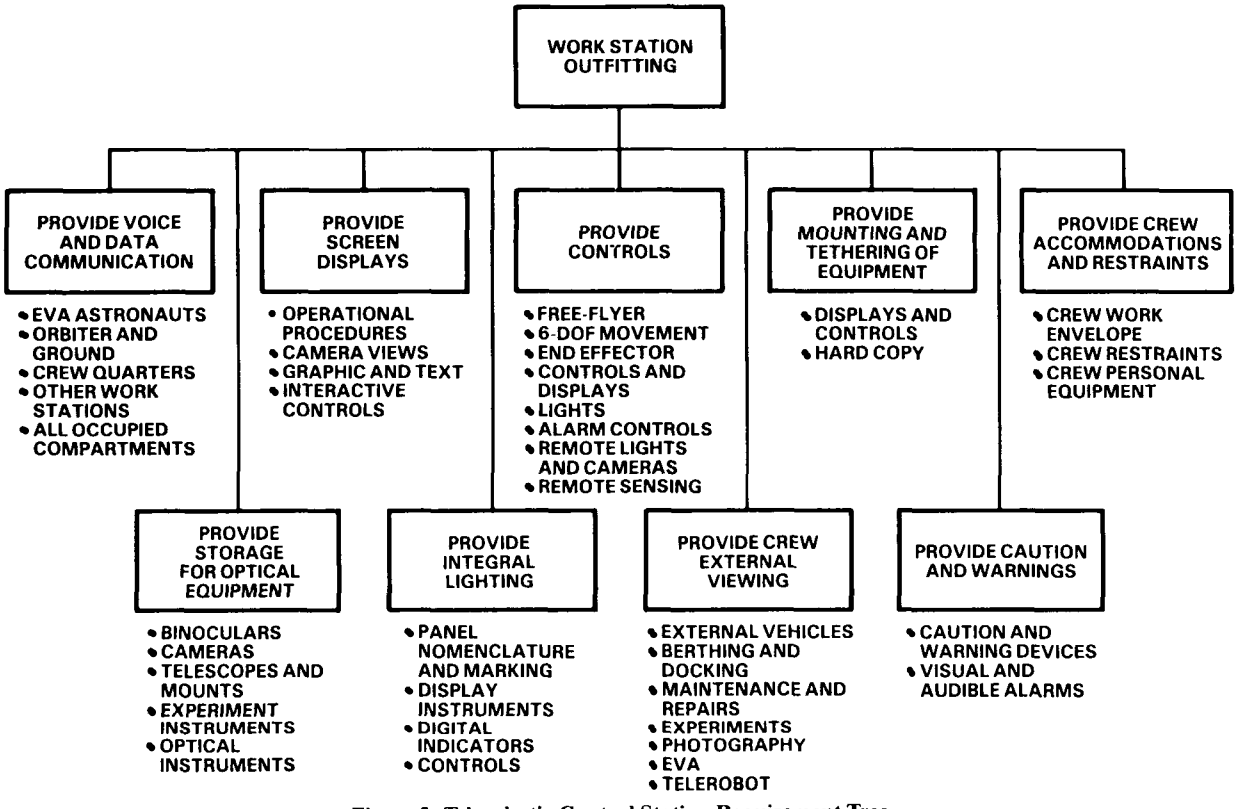


Figure 2. Telerobotic Control Station Requirement Tree

CONCLUSION

The Space Station is a complex and sophisticated structure filled with highly advanced and intricate devices. For humans to interact efficiently with these devices, the interfaces must be natural and direct. The development of crew-friendly control stations assists in accomplishing these goals. We believe the apportioning and verification procedures described above will allow us to design integrated and consolidated electronic controls and displays, permitting humans to effectively monitor and control events on board the Space Station.

ACKNOWLEDGEMENTS

The author would like to express her appreciation to the following individuals from the Space Station Systems Division of Rockwell International for their technical guidance and support in the development of this paper.

- Harvey A. Wichman, Ph.D.
  Crew Systems, Flight Crew Integration
- Kim Smith
  Human Factors, Flight Crew Integration
- Margaret M. Clarke, Ph.D.
  Manager, International Integration, External Integration

REFERENCES

1. Lewis, R.E., D. Gillian, and M. Rudisill, "The Use of Task Analysis in Designing Human-Computer Interface With the Space Station Information System," proceedings of AIAA 12th Annual Technical Symposium, Houston, Texas (May 14, 1987).
2. *Crew Interface With Space Operation*, IR&D Project No. 527, Rockwell International (June 1987).
3. *Automation and Robotics Plan*, Space Station Work Package 2 Definition and Preliminary Design Phase, DR-17, Rockwell International (June 13, 1986).
4. *Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy*, Vol. II-Technical Report, NASA, ATAC Report (March 1985).
5. Divona, C.J., W.M. Thompson, A. Quinn, and M.M. Clarke, "A Teleoperator for On-Orbit Manipulation," proceedings of International Topical Meeting on Remote Systems, Pasco, Washington (March 31, 1987).
6. Clarke, M.M., E.Y. Mok, W.B. Rosenfield, and A. Quinn, "An Orbiting Control Station For Free-Flyer Teleoperations: Preliminary Design Methodology," proceedings of SPIE Conference on Advances in Intelligent Robotics Systems, Boston, Massachusetts (October 1986).
7. Bronez, M.A., M.M. Clarke, and A. Quinn, "A Telerobot for Space Station Orbital Equipment Transfer," proceedings of Second World Conference on Robotics Research, Scottsdale, Arizona (August 1986).

TELEROBOT FOR SPACE STATION

LYLE M. JENKINS
JOHNSON SPACE CENTER

ABSTRACT

The Flight Telerobotic Servicer (FTS), a multiple arm dexterous manipulation system, will aid in the assembly, maintenance, and servicing of the Space Station. Fundamental ideas and basic conceptual designs for a Shuttle based telerobot system resulted from the Telerobotic Work System Definition Studies performed by Grumman and Martin-Marietta. Recent Space Station studies provide additional concepts that should aid in the accomplishment of mission requirements. Presently the FTS is in contractual source selection for a Phase B preliminary design. Concurrently, design requirements are being developed through a series of robotic assessment tasks being preformed at NASA and commercial installations. A number of the requirements for remote operation on the Space Station, necessary to supplement EVA (extra vehicular activity), will be met by the FTS. Finally, technology developed for telerobotics will advance the state-of-the-art of remote operating systems, enhance operator productivity, and prove instrumental in the evolution of an adaptive, intelligent autonomous robot.

INTRODUCTION

The Space Shuttle Remote Manipulator System has been the primary means for accomplishing remote manipulation operations such as handling payloads. The RMS is severely limited in its ability to perform dexterous tasks due to its size and control mode. It is a teleoperation system that is controlled in a resolved rate mode and is not suited to constrained motion tasks. Studies of requirements for servicing satellites has defined concepts for using smaller dexterous manipulators to perform more precise manipulative tasks. The Space Station has a number of applications for remote operating systems. The requirements include dexterous manipulation for its assembly, its maintenance, in servicing satellites and for servicing remote free-flying platforms.

SYSTEM DEVELOPMENT

The concept of a telerobot is being developed in the Flight Telerobotic Servicer for the performance of a number of dexterous manipulation functions that will be needed in the station assembly, in the station maintenance, and for servicing satellites. The concept is also being examined for remote servicing operations with the Orbital Maneuvering Vehicle and for free flying platforms. The Flight Telerobotic Servicer Program is expected to provide a unique capability for supplementing the work of the EVA.

The size of the Space Station will require a number of Shuttle flights for its construction, primarily with EVA. The tasks outside the pressurized modules on Space Station are presently planned for accomplishment by the crew in EVA (extravehicular activity). The FTS can be used in early operational tasks to increase EVA timeline margins. During the period when the Shuttle mission rules restrict the use of EVA due to the potential for space sickness, the FTS can be used to unstow and deploy assembly jigs and fixtures. Launch packages for struts and nodes can be prepared for start of structure assembly. Portable EVA restraint devices can be installed in the position for the initial EVA needs. During the assembly activities, the FTS can support operations by positioning cameras, by passing and holding tools and parts and by documenting, through TV images, the as-built configuration. Post EVA, the equipment status can be checked and the status of the closeout operation reviewed and documented. Incomplete steps in final preparations for return to earth can be performed with the FTS to preclude a contingency EVA. Assistance by FTS will increase operational margins and reduce astronaut exposure to hazards.

The FTS concept is based on two dexterous manipulator arms controlled from a remote station. The term "telerobotic" describes a system that is a combination of teleoperator and robotic control. The control station (work station) will be located in the cabin of the Space Shuttle on

initial missions. When the buildup of the station reaches the appropriate stage, the controlling operator will move to the interior of the Space Station at the standard work station. The FTS work station will be developed as a configuration of the standard work station.

The FTS was preceded by several conceptual studies including the Telerobotic Work System (TWS) shown in figure 1. Contracted studies by Grumman Aerospace Corporation and Martin Marietta Aerospace developed the TWS concepts illustrated in figures 2 and 3. These studies were based on the criteria of a design that would be able to perform tasks with a capability equivalent to the EVA astronaut. This EVA equivalency approach resulted in designs that are strongly anthropomorphic.

The reach and access envelope are summarized in figure 5 as functional requirements for manipulation. Additional requirements and design trades are summarized in reference [1] and [2]. A fundamental in the design approach is the use of modularity to allow effective evolution of the system as technology developed.

The current program plan for the development of the FTS consists of a technology element, a flight demonstration element, a flight system element and a ground system element. The technology element is the basis for the infusion of state of the art advancements into the initial and evolutionary FTS. A flight test of the FTS is planned to validate the telerobot design for zero-g operations. Testing in space will evaluate the task performance interface and provide a basis for the development of procedures and training. The operational FTS will be used during the initial assembly of Space Station. It will function as multipurpose tool for the maintenance of external subsystems on the station. It will have a role in the assembly, installation and servicing of payloads as well. The ground element of the program is intended to coordinate a network of telerobotic laboratories within NASA.

An evolutionary philosophy has been incorporated in the program plan for the Flight Telerobotic Servicer. The plan considers the technology base for development of the telerobot from manufacturing industry, the nuclear energy operations and undersea servicing and exploration. Although each of these areas has contributing features, the challenge of a space telerobot goes beyond the cumulative capability that has been developed in these industries. NASA's own program in telerobotic technology is a multi-center effort that is concentrated in the Office of Aeronautics and Space Technology telerobotic testbed at the Jet Propulsion Laboratory. The testbed serves as a systems laboratory for the resolution of systems issues as well as the demonstration of the functionality of various equipment combinations. The testbed has a shortcoming in the consideration of zero-g. Other capabilities for the effective simulation of the space environment exist in the flat floors and neutral buoyancy facilities at the Johnson Space Center and the Marshall Space Flight Center. Computer simulations are another way of evaluating space operations. Functional requirements are also the objective of a series of Robotic Assessment Tasks that are being conducted by the NASA centers responsible for Space Station work packages. Eventually the validity of the ground simulations will need to be correlated by flight test.

Flight tests may be categorized as research, validation of simulations, system development test and flight system verification. The human interaction with the displays and controls is difficult to simulate in system development. The operation of manipulator controllers in zero-g depends of the type of control, the actuation forces and the precision of position and movement. An evaluation of the complex interaction of the physical characteristics of the mechanical arms, controller parameters and the restraint of the operator must be provided to reduce risk in the design of the telerobot. A system with bilateral force reflection is generally acknowledged to require less training in the performance of remote manipulation tasks on earth. This has not been established for space operations. Although the benefits maybe confirmed, constraints imposed by the volume of the spacecraft may limit the application of bilateral force reflection.

SUMMARY

The FTS program is a key development in the capability to perform physical activities remotely in the hazardous environment of space. The program is planned to progress through a Phase B study to a Phase C-D design and development. With the continued application of technology, the productivity of the operator can be enhanced and the state of the art in remote operating systems can be advanced. The first space telerobot, the FTS, should prove to be an instrumental first step in the evolution to an adaptive, intelligent, autonomous robot.

REFERENCES

[1] Telerobotic Work System Definition Study. Grumman Aerospace Corporation, Contract NAS9-17229, Final Report SA-TWS-87-R002 dated April 1987.

[2] Telepresence Work Station System Definition Study. Martin Marietta Denver Aerospace Report MCR-86-528, May 1987.

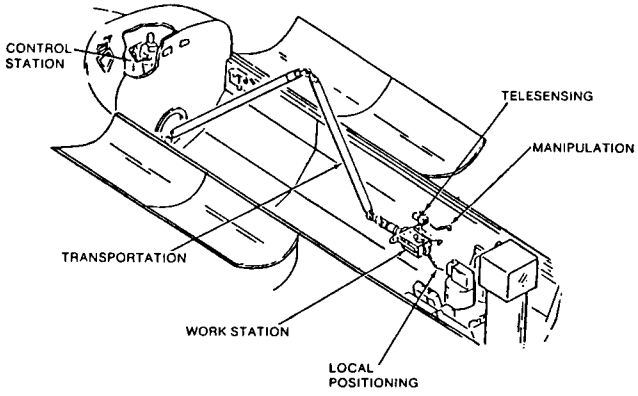


Figure 1. Definition of TWS Systems

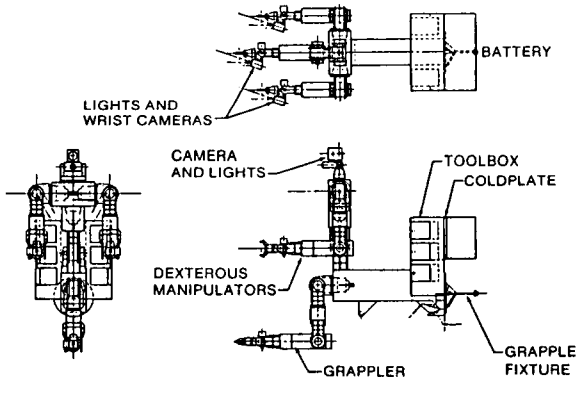


Figure 2. Grumman Telerobot Concept

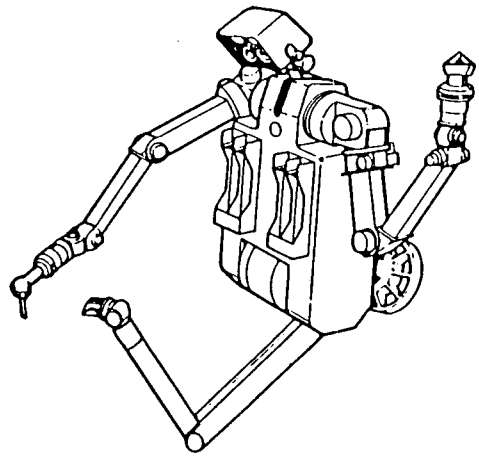


Figure 3. Martin Telerobot Concept

• DEXTERITY	
– ACCESS DEPTH	14"
– ACCESS OPENING	6' X 6'
– ORU DIMENSIONS (MAX)	36" X 36" X 72"
(MIN)	6" X 7" X 2"
– ORU MASS	3 LB TO 700 LB
• STRENGTH	
– INSERTION	20 LBS (PUSH/PULL)
– TORQUE (MAX)	2800 IN OZ
(WRIST)	576 IN OZ
(MIN)	10 IN OZ
• ACCURACY	
– TRANSLATION	.0625"
– ROTATION	±1°
• MOVEMENT	TWO HANDED OPERATION

R87-3945-111(T)

Figure 4. Summary of Mission Requirements for Manipulation

THE USE OF COMPUTER GRAPHIC SIMULATION IN THE DEVELOPMENT OF ROBOTIC SYSTEMS

Ken Fernandez

National Aeronautics and Space Administration
Marshall Space Flight Center
Information and Electronic Systems Laboratory
Huntsville, Alabama 35812

ABSTRACT

This paper describes the use of computer graphic simulation techniques to resolve critical design and operational issues for robotic systems. Use of this technology will result in greatly improved systems and reduced development costs. The major design issues in developing effective robotic systems are discussed and the use of ROBOSIM, a NASA developed simulation tool, to address these issues is presented. Three representative simulation case studies are reviewed: off-line programming of the robotic welding development cell for the Space Shuttle Main Engine (SSME); the integration of a sensor to control the robot used for removing the Thermal Protection System (TPS) from the Solid Rocket Booster (SRB); and the development of a teleoperator/robot mechanism for the Orbital Maneuvering Vehicle (OMV).

KEYWORDS

Robotics, Simulation, Computer-graphics, CAD, CAM, Off-line-programming, Robotic-welding, Robotic-spraying, Satellite-servicing.

INTRODUCTION

Robotic systems have become increasingly important to all facets of manufacturing: space is no exception. Perhaps the most publicized space robot is the Remote Manipulator System (RMS) which was built by Canada for the U.S. Space Shuttle. Prior to the RMS, robot manipulators were used on unmanned spacecraft to investigate soil properties on the moon and on Mars. Plans for the U.S. Space Station which will become operational in the early 1990's include the use of teleoperators and robots to perform routine station tasks e.g., inspection and maintenance. Earth-bound robots have also been used extensively to support the manufacturing of spacecraft components (Fernandez 1983,1985). Although the applications for space and earth seem radically different there remain many common issues in the procedures for design and testing of robot systems. Graphic simulation has proven to be extremely effective in the design of both types of system. In this paper we will examine: design issues for robots; ROBOSIM, a NASA developed computer graphic simulation tool; and three robotic systems that were developed using computer graphic simulation techniques.

Kinematic Design Issues

In designing a robot cell the selection of the robot's kinematic design is usually considered first. The number of robot joints, the type of joint (revolute or prismatic), and the physical configuration of each jointed segment are all elements of the robot's kinematic design. The position of the last reference frame (hand frame) is determined by the joint positions and the geometric relationships (kinematics). Minor changes in the kinematic design of a manipulator can greatly affect the volume through which the robot's hand may be moved. The design of the end-effector (tool) and the orientation of the part (workpiece) with respect to the robot (part positioning) also greatly affect the ability of a robot to perform a given task. For applications which will use an existing robot the designer must choose the appropriate robot, design the workcell layout and part fixturing. For systems which will use a custom-built robot, the task of designing the robot is added. A mistake in the design of a cell without the use of computer graphic simulation may not be detected until the hardware integration phase. This can result in costly schedule delays, procurement of incorrect components, and a greatly increased system cost.

PRECEDING PAGE BLANK NOT FILMED

Robot Motion Control

Robot control development is another area which can benefit from the use of computer graphic simulation techniques. Robot control algorithms may be viewed as existing at two levels: the kinematic control level; and the path planning level. Kinematic control algorithms are a function of the arm's kinematic design. These algorithms relate the position of the end-effector's reference frame to the joint position commands required to achieve the commanded position. These algorithms are a software implementation of the inverse kinematic equations. Prior to the use of graphic simulation, the control programs were debugged by observing the robot's motion subject to the commands of the experimental computer program. For robot systems with relatively low lifting capacity, a faulty program resulted in little more than embarrassment for the developer, however robot capacities have increased to the point where payloads are in the hundreds or thousands of pounds. Mistakes in programming can be serious. Another difficulty encountered in using the actual mechanism in the debugging process occurs for robots designed for use in zero-G which may not operate in a one-G environment. Again graphic simulation is the indicated procedure for this type of development.

Robot Path-Planning/Verification

Robot path-planning is the process of developing the sequential position, orientation and velocity commands that the robot's end-effector must execute in order to perform the desired function. Most current industrial robots are programmed using a teach pendant to manually command the robot to the desired points, this is the on-line manual programming method. Manual programming is highly in-efficient since the robot must be taken out of service, the path generated manually, replayed for verification and ultimately executed. On robots whose path programming is changed infrequently this is not significant, but for systems in which programming must be flexible manual programming is not satisfactory. Just as numerically controlled (NC) machine tools have become entirely programmed by off-line algorithms, the programming of robots will also eventually all be automated. Graphic simulation is a vital step that must be performed prior to the execution of an off-line generated robotic path program. Simulation will verify that: (1) the path specified is correct for the task; (2) the inverse kinematic equation may be solved at all points along the path program (controllability); and (3) the arm or other components will not collide accidentally with obstacles within the workcell.

Robot Dynamics

In industrial applications the primary dynamics issues are that the robot chosen for a task is capable of handling the required payload weights and transport velocities. Industrial robots are typically rated for lifting capacity only. An approximation of the robot's ability to perform a task dynamically can be made through dynamic simulation of the loaded robot. The maximum joint loads recorded during the dynamic simulation are compared to the loads that result if the manipulator were statically loaded per the manufacturer's specifications. If these joint loads are exceeded by the dynamic tests, then the robot may not be capable of performing the task. Since this is only an approximation, a safety margin should be used in making the final decision.

Although dynamic simulation is important for industrial robot systems, it is mandatory for systems used in space. Manipulator mechanisms and joint actuators are limited in weight due to launch considerations. Power supply limits reduce the size and rating of the mechanism's actuators. Dynamic studies will help to insure that the planned robotic tasks do not exceed the limits of the mechanism. The zero-G environment may be an advantage for handling larger payloads than would be possible on earth, but the dynamic interactions of the loaded manipulator and its mounting platform are significant for a space based robotic system. The possibility exists for parasitic oscillations to occur between the manipulator and the spacecraft's attitude control system. Simulation studies may reveal the existence of these or other undesirable effects.

ROBOSIM OVERVIEW

Simulation Procedure

ROBOSIM was developed over a three year period at the Marshall Space Flight Center (MSFC) to facilitate the design and development of robotic systems. Prior to ROBOSIM, robotic simulations were limited to the construction of scale models. Using ROBOSIM the kinematic design of the manipulator mechanism and other workcell components are modeled via a simulation language. The model consists of solid primitive shapes which approximate the robot's shape and mass properties. The joint configuration and type, either revolute, prismatic or fixed, are also specified. Once modeled, ROBOSIM computes

the standard linkage parameters (Hartenberg 1955), the inverse kinematics and the manipulator's dynamics. The designer may also specify the joint actuator transfer functions. Path motion is specified by position and velocity language constructs.

ROBOSIM Hardware Configuration

ROBOSIM is resident on a Digital Equipment Corporation (DEC) VAX11/780 processor. During simulation development the user may use a low cost terminal with TEK 4014 graphics compatibility. Although a simulation may be executed using a non-real-time terminal, the use of a real-time graphics display is preferred. Interfaces have been provided for several dynamic display systems including Evans & Sutherland PS330, GTI Poly 2000, Silicon Graphics IRIS with other interfaces planned. A limited Initial Graphics Exchange Standard (IGES) pre- and post-processor allows ROBOSIM to communicate graphics and tool motion commands with any CAD/CAM system adhering to the standard which was developed by the U.S. National Bureau of Standards.

The simulator's speed for non-dynamic studies is greater than real-time. This speed is decreased for very large models with multiple robots or robots with many degrees-of-freedom. Studies that required the modeling of dynamic effects also load the simulation processor. An Applied Dynamics AD10 parallel processor is used to improve the simulator's response in these situations.

ROBOSIM Software System Structure

ROBOSIM's software structure may be characterized as a hierarchy of three levels of software utilities. This structure is typical of large software systems. At the core or kernel of this system are routines that provide support for the most rudimentary of simulation tasks. Included among these functions are vector and matrix arithmetic and display control. The typical user of ROBOSIM interacts with these routines indirectly through his use of higher level utilities. A characteristic of routines at this level is their inflexibility in their interfacing requirements i.e., data must be provided in specific formats. By interfacing via the higher levels a user avoids these requirements, however direct access is available when needed. Typically, a ROBOSIM user who is performing simulation studies involving externally supplied mechanism control algorithms must communicate directly with the kernel routines.

The second level within ROBOSIM integrates the lower level routines into more complex algorithms that perform often needed tasks in display management and robot control. Examples of graphics routines that function at this level include subroutines to perform viewpoint and perspective transformations. Examples of routines that service robot kinematics and control issues include those which perform end-effector position computations and formulations of the manipulator's Jacobian matrix.

The highest level within ROBOSIM provides the human interface. At this level robots, workpieces, and fixturing assemblies may be modeled, placed within a workcell, programmed, dynamically simulated and viewed using fewer than forty distinct language instructions. The simplicity of this software interface greatly increases ROBOSIM's use and it is this interface that is perhaps the most important feature of ROBOSIM.

SIMULATION EXAMPLES

ROBOSIM V1.0 became operational in July 1985. In the year since, ROBOSIM has been applied to numerous robot simulation studies, the three listed below are typical. The studies include: the development of an off-line programming algorithms for welding on the SSME; the development of vision sensor guided control for a robot used to refurbish the SRB; and the design of a robot manipulator for the Orbital Maneuvering Vehicle. For each study a discussion of the application, the simulation goals, and the results will be presented.

Downhand Control for SSME Robotic Welding

The Space Shuttle Main Engine is constructed of stainless steel using over 2000 welded seams. At the present time 30% of these welds are performed by fixed automation while the remaining 70% are performed manually. A study performed of the manufacturing operation indicates that an additional 30% could be automated in a cost-effective manner using robotic welding techniques. The primary goal of this effort is the improvement of weld quality and reliability. A further improvement in manufacturing efficiency could be obtained by using automatic off-line robot programming techniques with downhand welding control. Downhand welding is the term applied to arc welding with the part in an orientation that maintains the weld puddle in a horizontal plane. This allows increased puddle size with a resulting greater deposition rate, fewer passes and reduced welding times.

Manual robot programming to perform downhand welding is extremely tedious and the results are only approximate. The algorithm for automatic off-line programming of the downhand position (Fernandez 1986) was developed using ROBOSIM as a test bed. The algorithm programs the robot and part positioner so that their coordinated motion results in a constant weld travel speed while maintaining the downhand position. Figure 1 depicts the robot cell with the six degree-of-freedom robot and a two degree-of-freedom part positioner. The part in figure 1 is a corrugated metal sheet. The part geometry may be read from a CAD data base using IGES format, or it may have been inferred from a manually generated path program sent to the downhand algorithm via the robot's communication interface. In either case the algorithm computes the desired local vertical in a reference frame moving along the weld seam at the specified weld velocity. Weld positioner commands are computed so that the desired downhand orientation is achieved. Robot position and velocity commands are also generated to keep the torch moving in the weld seam at a constant surface feed rate. Figure 2 depicts several frames from the simulation of the downhand welding algorithm. In figure 2 we note that the algorithm is functioning since the tangent to the weld seam remains horizontal at the point where the torch is in contact.

Vision Guided Off-Line Programming for SRB Refurbishment

The Solid Rocket Boosters used to assist in launching the Space Shuttle are designed to be re-used. To achieve this the Thermal Protection System (TPS) prevents the erosion of the booster's casing during the heat of re-entry. The main component of the TPS is the Marshall Sprayable Ablator (MSA) which reduces the booster's skin temperature by controlled evaporation. After recovery at sea the SRB is returned to the booster processing facility at the Kennedy Space Center (KSC).

High-pressure waterblast (20000 psi) is used to remove the partially burned ablative material prior to its re-application. Due to the difficulty in performing the cleaning operations manually, robotic workcells were developed (Fernandez 1983). Prototypes of these workcells were implemented at the MSFC Industrial Productivity Facility in Huntsville, Alabama. A computer graphic simulation of the prototype robotic cell is shown in figure 3. The robot, a Cincinnati Milacron HT3, is equipped with the high pressure nozzle. The aft booster section is shown mounted on a computer controlled rotary positioning table. In the initial implementation of this cell, manual robot programming methods were employed. The current operation includes both manual and off-line programming techniques. One problem in the operation of this cell occurs when the water blast fails to remove the MSA in the first cleaning pass. At this point the robot and turntable must be re-programmed manually to perform the touch-up cleaning.

A solution to the problem of programming the robot to perform touch-up cleaning of the TPS residue is the development of a vision sensor and off-line programming utilities. Graphic simulation via ROBOSIM was used to develop these programming utilities without the danger of damaging the actual workcell during initial development and de-bugging procedures. In operation the vision sensor will scan sections of the SRB that are presented by rotating the turntable. Due to the spray and debris real-time visual inspection is not possible, instead the inspection is performed after the entire cleaning pass is completed. The vision algorithm within the sensor provides information on the location of the MSA residues as x and y-coordinate locations referenced to the image plane of the sensor camera. Although the vision routines were developed under a separate effort, the camera is simulated in the graphic system by placing an "eye-point" in the same location and orientation as the hardware system. The focal length of the perspective transformation (Duda 1973) associated with the "eye-point" is adjusted to match the field-of-view of the sensor camera's lens. In evaluating the off-line programming algorithm a simulated MSA residue is placed on the modeled SRB. The residue is placed within the field-of-view of the "eye-point" by rotating the turntable in the graphics model. To simulate the sensor's output the screen x and y-coordinates are noted and passed as input to the off-line programming utilities in a manner similar to the actual sensor. The resulting turntable and robot motion commands were executed by the graphic model, and the resulting operation was viewed in graphics to determine if proper cleaning motion would have occurred. This result is established when the graphic representation of the spray (dotted line in figure 3) impinges on the simulated residue. The use of graphic simulation will continue when the sensor is integrated into the cleaning workcell. During operations the simulation will serve as a preview verification of the off-line generated cleaning paths.

Design of A Robot for the Orbital Maneuvering Vehicle

The Orbital Maneuvering Vehicle is designed as a re-useable, remotely controlled, free-flying vehicle capable of performing a wide range of on-orbit services in support of orbiting assets. It is projected as an important element of the Space Transportation System (STS), designed to operate from either the Shuttle , the Space Station or from the ground. The descriptions of the OMV or manipulator mechanism contained in this paper are not specific to any designs which may be currently under consideration by the U.S. National Aeronautics and Space Administration, however, the functional concepts described are correct and have been published elsewhere (Huber 1984).

The concept of the OMV includes the ability to accept mission kits to allow it to perform a variety of tasks in addition to its role as recoverable booster. One such kit is a manipulator/teleoperator, the "Smart Front-End" (SFE), which will allow remotely controlled manipulation to accomplish satellite and Space Station service tasks on-orbit. Figure 4 illustrates this concept. The OMV is shown equipped with a generic SFE manipulator. The SFE pictured consists of a bi-lateral pair of six degree-of-freedom (DOF) manipulators and a manipulator transport mechanism. The transport system provides three DOF: a rotary track which encircles the docking adapter; a hinged boom; and a sliding joint allowing the bi-lateral pair to traverse the boom. The generic satellite which is being serviced in figure 4 is shown detached from the OMV/SFE cluster for clarity. In normal operation a solid connection would be established by a docking mechanism.

ROBOSIM will be used extensively to assist in the development and evaluation of concepts for the SFE manipulator. Kinematic studies will reveal whether the SFE mechanism can be folded and stored within the space allocated on-board the Space Shuttle. Other kinematic studies will be required to determine if the OMV/SFE cluster can be successfully deployed from the cargo bay by the Space Shuttle's RMS. In figure 5 our generic OMV/SFE cluster is shown with the SFE folded in the stowable configuration. Further kinematic studies will determine if collisions between the SFE manipulator and satellite appendages occur during the execution of planned motion paths.

The implementation of an SFE manipulator will also require the development of several modes of mechanism control. An algorithm to control the SFE during deployment or un-folding will be developed. Although this type of algorithm usually involves a predetermined sequence of joint motions, provision must be included to override this sequence, if necessary, and execute new motions to correct or avoid anomalies. During docking operations the mechanism can take a passive or an active role. If a passive role is assumed, control algorithms for the SFE can improve the maneuverability of the OMV by arranging the arm's configuration to minimize inertial imbalance, avoid obstruction of the target satellite and prevent the reaction control system (RCS) thruster plumes from impinging on the SFE. Strategies of controlled compliance in the SFE joint servo control loops may further improve the controllability of the OMV during fine docking maneuvers by de-coupling the SFE's mass or actively using the SFE's momentum to affect additional control.

Once the OMV is docked with the target satellite a variety of different control issues must be resolved. As previously mentioned, algorithms that use mechanisms with kinematic redundancy to avoid collisions and minimize disturbance torques could significantly improve the system's performance. Real-time computer graphic simulation coupled to prototype teleoperator workstations can aid in resolving many issues relating to man-in-the-loop control. The placement of cameras may be simulated to insure that the field-of-view (FOV) is not obstructed. If a dual arm SFE design is chosen, graphic simulation could help to determine the most effective human interface for controlling the bi-lateral mechanism. Graphic simulation will not end with the successful SFE design, during servicing activities, a graphic display will allow the human operator to preview service tasks in simulation. Since communication delays in the man-in-the-loop control system may be large and varying, the use of a "predictive graphic display" to supplement the delayed visual feedback may improve the efficiency in performing operations remotely. When semi-autonomous or "supervisor control" methods are developed, the graphics display would allow the human to verify mechanism motions that are proposed by the controller. One final note relates to the design of the satellite rather than the OMV itself. Current satellite design philosophy is oriented toward multiple redundancy and no post-launch servicing, the advent of on-orbit service techniques will relax some of these design constraints, but satellite design must change to take advantage of these new possibilities. Hardware simulations of servicing missions on modular satellites have been performed (Fernandez 1980a,1980b,1984 and Scott 1985a,1985b),

but computer graphic simulation provides a cost-effective means of preliminary evaluation of the compatibility between a satellite and the servicer.

CONCLUSIONS

The experience gained at the Marshall Space Flight Center indicates that the use of computer graphic simulation in support of robot systems development is extremely important. Although hardware implementation is not replaced by these simulators, a considerable cost savings is experienced by delaying hardware implementation until the designs have matured. Once a robot system becomes operational the value of graphic simulation continues as a means of previewing planned task execution. It is expected that as the performance of computer graphic simulators increases and as hardware costs decrease the use of graphic methods will become widespread.

REFERENCES

Duda, R. O. and Hart, P. E. (1973). Pattern Classification and Scene Analysis. John Wiley & Sons, New York. Ch. 10, pp. 379-404.

Fernandez, K. R. (1980a). Computer Control of a Robotic Satellite Servicer. Proc. of the IEEE3 Southeastcon '80. Nashville, TN. pp.237-240.

Fernandez, K. R. (1980b). Application of a Computer Controlled Robot to Remote Equipment Maintenance. Proc. of the IEEE IASCON80. Cincinnati, OH. pp.1180-1184.

Fernandez, K. R., Jones, C. S. III, and Roberts, M. L. (1983). NASA's Use of Robotics in Building the Space Shuttle. Proc. of 13th ISIR/ROBOTS 7 of the SME. Vol. 1. Chicago, IL. pp.11-35 to 11-43.

Fernandez, K. R., Purinton, S. C., and Bryan, T. (1984). Simulation of a Robot System Used to Remotely Service Satellites. Proc. of the ANS Robotics and Remote Handling in Hostile Environments Nat. Topical Meeting. Gatlingburg, TN. pp.317-321.

Fernandez, K. R., et al (1985). In Robert L. Vaughn (Ed.) Space Shuttle: A Triumph in Manufacturing. SME Dearborn, Michigan. Chapter 5, pp.229-248.

Fernandez, K. R. and Cook, G. E. (1986). Computer Graphic Simulation of An Algorithm for Controlling Downhand Position in Robotic Welding. Proc. of the SME Conf. on Robotic Solutions in Aerospace Manufacturing. Orlando, FL. 10 pages.

Hartenberg, R. S. and Denavit, J. (1955). A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. ASME Jour. of Applied Mechanics June, pp.215-221.

Huber, W. G. (1984). User's Guide for Orbital Maneuvering Vehicle. NASA, MSFC. 12 pages.

Scott, D. R. (1985a). Remote Satellite Servicing. Proc. of the NASA Workshop on Proximity Operations in Earth Orbit. Houston, TX. 14 pages.

Scott, D. R. (1985b). Concepts in Remote Satellite Servicing. Proc. of the NASA Workshop on Satellite Servicing. Houston, TX. 16 pages.

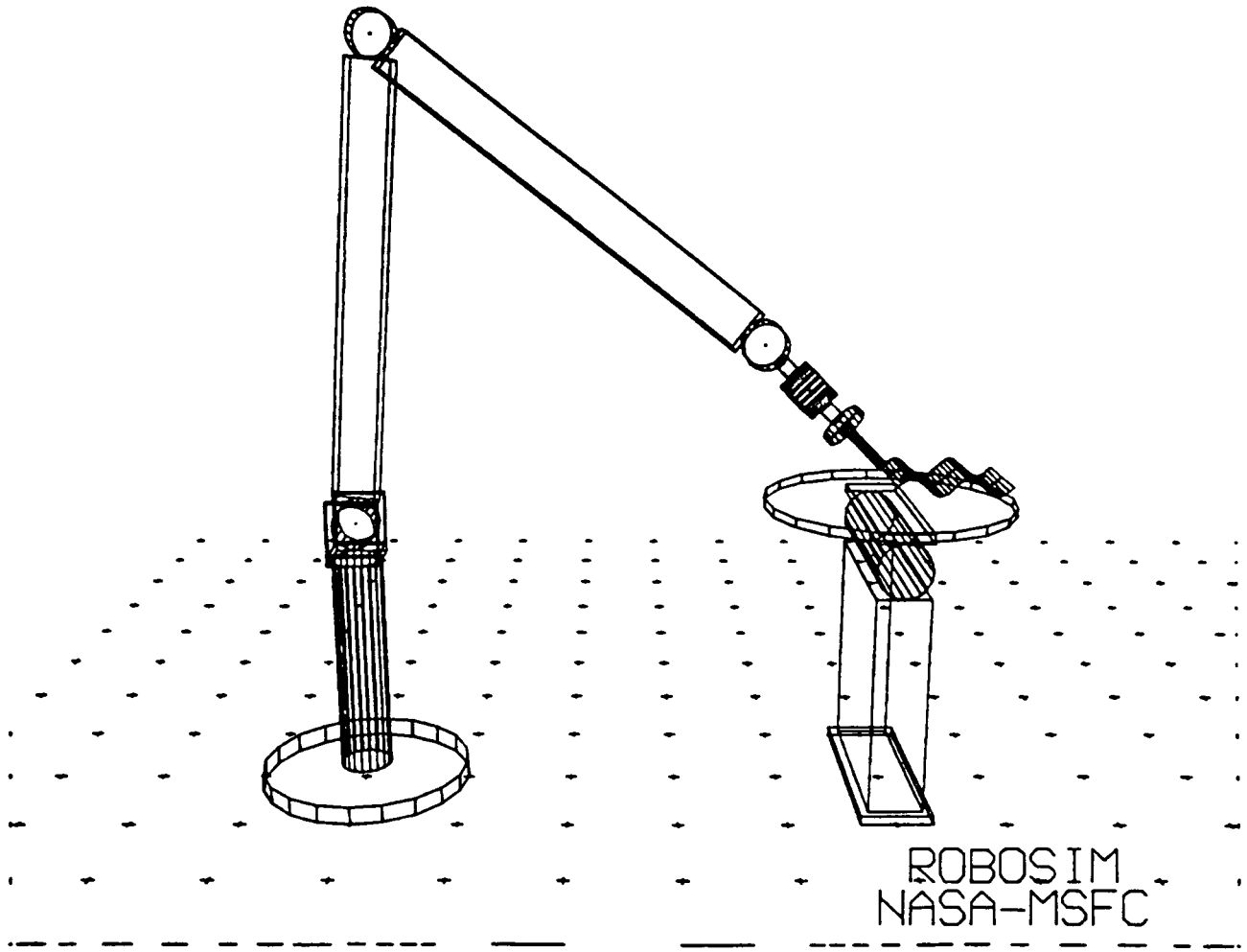


Fig. 1. Simulated Six Axis Robot and Two Axis Positioner

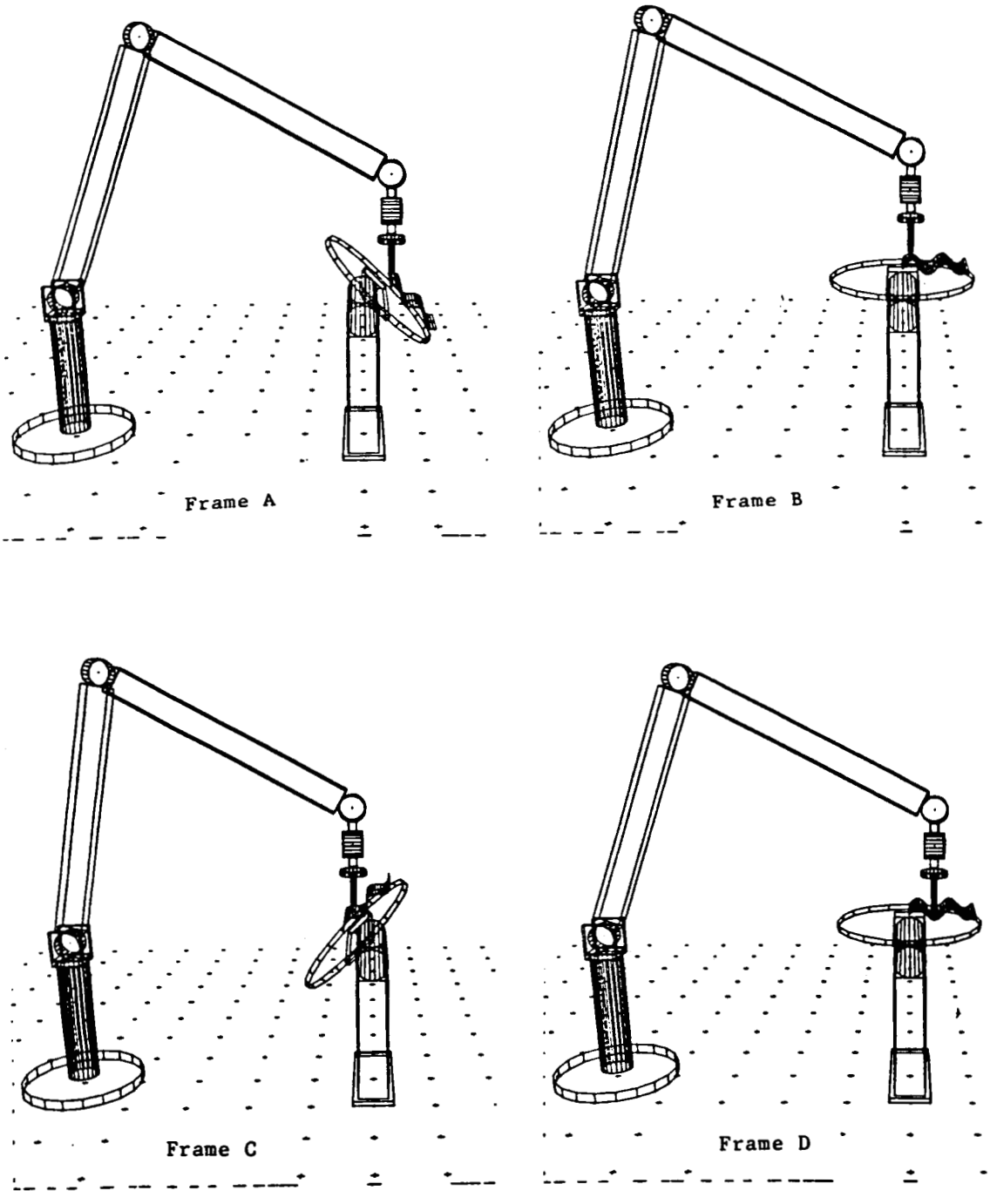


Fig. 2. Simulated Weld Operation with Automatic Downhand Position

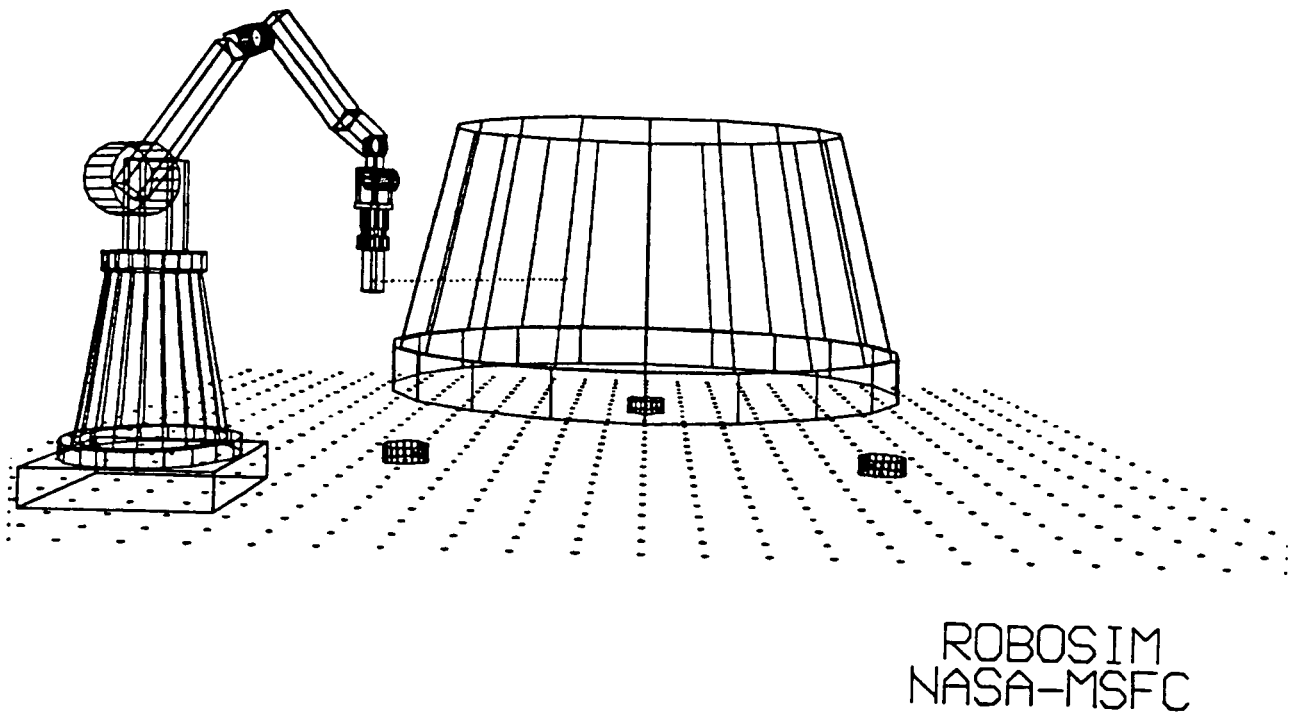


Fig. 3. Simulated SRB Refurbishment Workcell

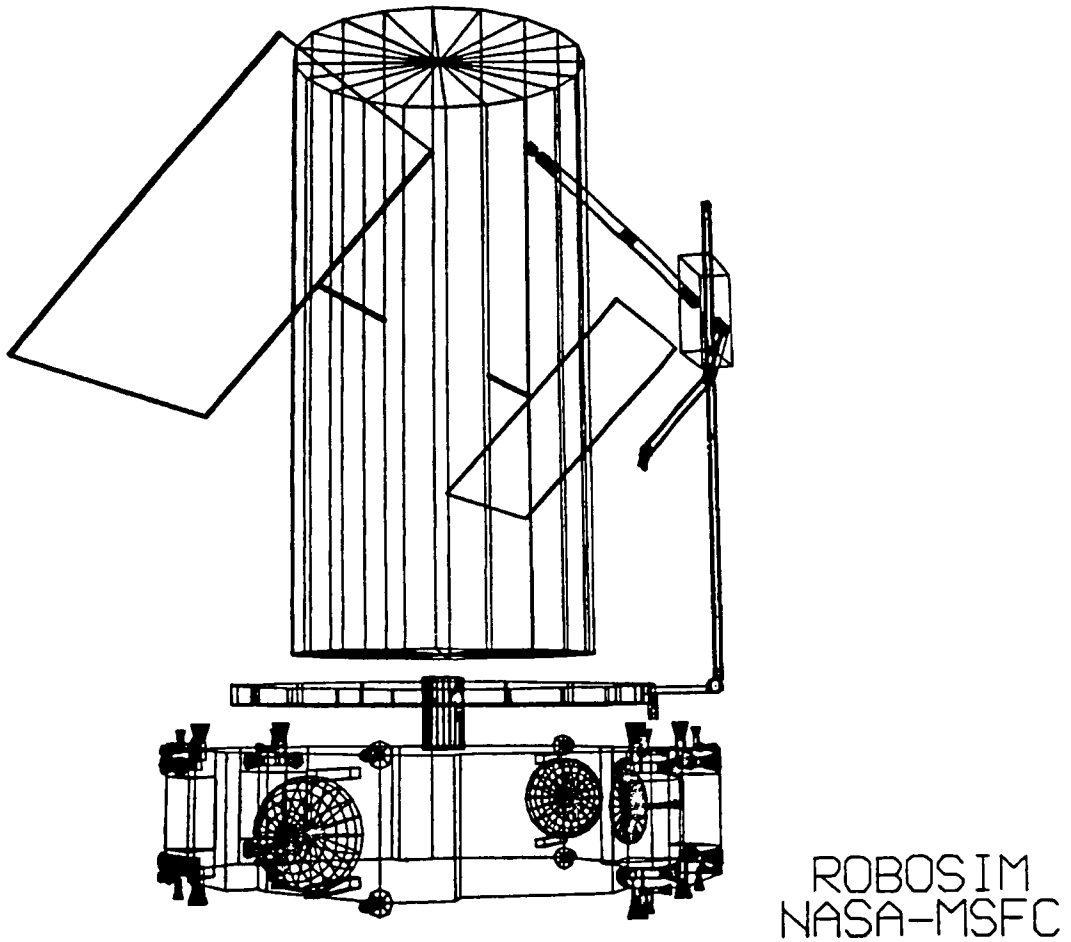


Fig. 4. Generic OMV Performing Satellite Servicing

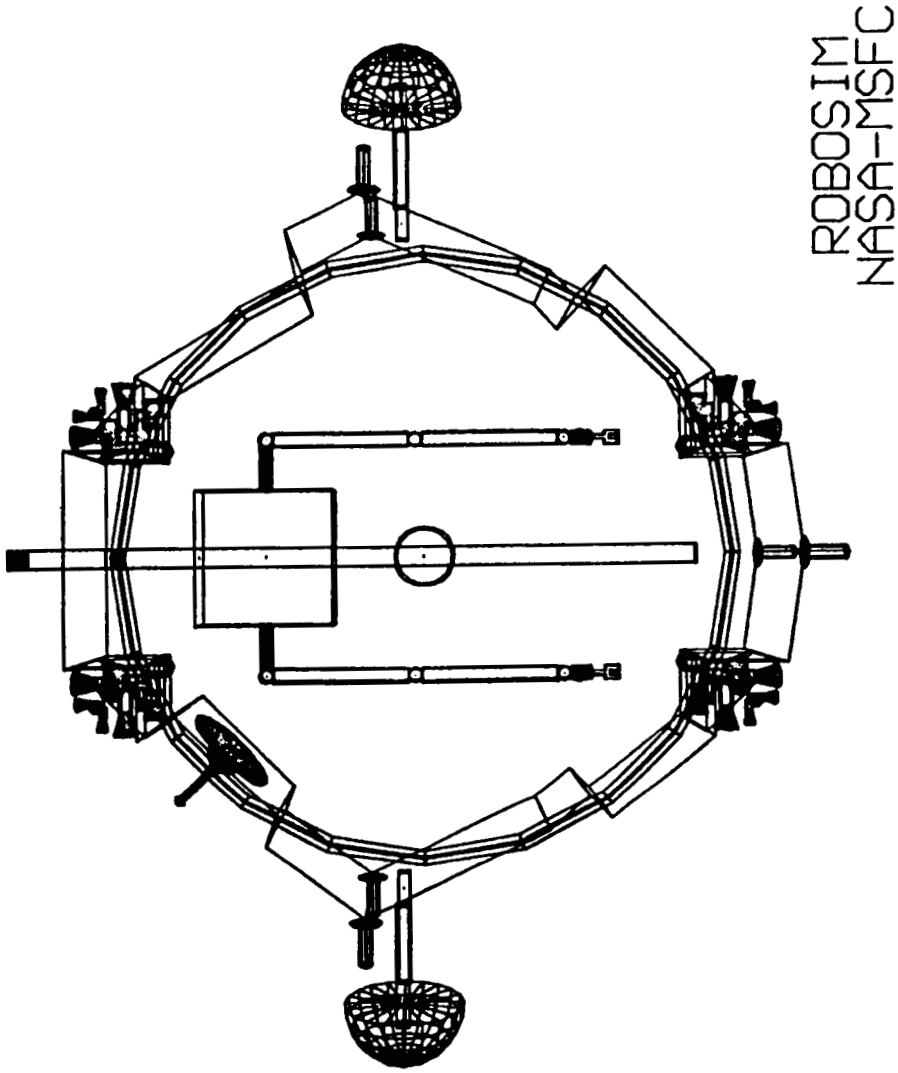


Fig. 5. OMV Shown with SFE Manipulator in Stowed Position

INTELLIGENT ROBOTIC TRACKER

W. S. Otaguro, L. O. Kesler
McDonnell Douglas Astronautics Company
5301 Bolsa Avenue
Huntington Beach, California 92647

K. C. Land, D. E. Rhoades
NASA-JSC

An intelligent tracker capable of robotic applications requiring guidance and control of platforms, robotic arms, and end effectors has been developed. This packaged system capable of "supervised autonomous" robotic functions is partitioned into a multiple processor/parallel processing configuration. The system currently interfaces to cameras but has the capability to also use three dimensional inputs from scanning laser rangers. The inputs are fed into an image processing and tracking section where the camera inputs are conditioned for the multiple tracker algorithms. An executive section monitors the image processing and tracker outputs and performs all the control and decision processes. The present architecture of the system will be presented with discussion on its evolutionary growth for space applications.
An autonomous rendezvous demonstration of this system was performed last year at JSC. More realistic functional demonstrations using the MMU simulator and the manipulator development facility planned for this year will be discussed.

INTRODUCTION

The purpose of this project was to functionally demonstrate the McDonnell Douglas Astronautics Company (MDAC) Robotic Tracking Sensor in autonomous homing and contour following modes. The major benefit of this demonstration was the application of existing technology to space operations in order to evaluate the ability of this equipment to meet near term autonomous tracking and sensing requirements with low hardware and software development costs. The scope of this project was purely a functional demonstration resulting in qualitative data. A more structured, quantative test can be performed at the completion of the upgrade of the Laser Optical Tracking Testbed facilities in Building 14 at JSC.
This demonstration addressed only the homing and contour following modes. More realistic space operations such as inspection, maintenance, assembly, and retrieval would be addressed later.

MDAC Tracking Sensor

The architecture of the multimode sensor tracker is shown in Figure 1. The multiprocessor tracker is composed of three functional parts:

1) Two Fairchild CCD 3000 cameras and video processor.
2) The MDAC 673 image and tracker processor.
3) The Z8000 executive control processor.

The video tracking functions are computation intensive requiring a high throughput special purpose signal processor. To match the video data with the bandwidth of the image processor, data compression is performed by the video preprocessor by either excluding regions of the scene that are of no interest or by performing a pixel averaging. This effectively performs video windowing and an electronic zoom. The preprocessor also performs a tracker controlled brightness and contrast adjustment to the video image. This enhances the tracker's capability to see and track the target.
The MDAC 673 is a high speed, 10 MOPS, special purpose microcodable signal processor. All tracking functions are performed in the MDAC 673. Existing algorithms are: (1) correlation, (2) centroid, (3) conformal gate, and (4) guard gate. The primary trackers requaired for these demonstrations were the correlation and centroid trackers. The correlation tracker is a feature tracker that tracks by finding the best match of a video reference image with the scene. The centroid tracker is a contrast tracker that finds the center of the target exhibiting intensities above or below a controllable threshold. The conformal gate and guard gate trackers were required for countermeasure techniques or when the target background exhibited a lot of clutter. The conformal gate tracker is a statistical tracker that classified the scene as either background, target, or unknown. This tracker finds the target boundary and maintains the tracker gate size to enclose all of the target. The guard gate tracker detects when the target passes behind obstacles and controls the other tracker's operations while the target is not visible.

The Z8000 executive control processor directs the operation of the multimode trackers, provides operator interface, and controls the responses of either vehicles or mechanisms. The executive processor controls acquisition of the target, monitors each tracker's aimpoint, and can reinitialize any tracker algorithm during the engagement. The operator interface is provided through the hand controller and the video monitor mounted on the tracking sensor. The tracking sensor was mounted on a mobile platform manufactured by Cybermation.

System Configuration

The MDAC tracking sensor was integrated with the Cybermation platform through a serial link between the MDAC Z8000 processor and Cybermation processor. The tracking sensor computer and electronics were mounted on the Cybermation turret mount. The long range camera was oriented with the platform wheels. Figure 2 shows a block diagram of the interfaces.
The executive control software was modified for autonomous tracking and control of the decision logic. Specifically, two modes are engaged by the executive control software: homing and contour following. While in the homing mode, the tracking sensor monitors the target position while guiding the platform to a specified distance from the target as determined by the camera image size. The executive then switches to the contour following mode. This mode commands the platforms to move laterally around the target by using a structural feature of the target. The tracking sensor remains at the specified range with the target in the center of view. At the completion of one revolution, the platform is stopped.
A third mode was added to the executive control software in order to accentuate the autonomous features of the tracking sensor. In this mode, the "heel" mode, the tracking sensor tracks and follows a target while maintaining a specified distance from the target as determined by camera image size. When the platform gets within this specified range, it stops with the tracker still maintaining lock-on of the target even if the target moves around the platform within the specified range. When the target moves away from the platform, the tracking sensor again commands the platform to follow the target.

Functional Demonstrations

The tracking sensor was demonstrated in the Laser Optical Tracking Testbed in Building 14 at JSC during the week of 21 July 1986. Figure 3 shows the demonstration format. The demonstration began with a manual acquisition of the target at a range of up to 100 feet. The tracking sensor locked on to the target in the acquisition gate and guided the platform to the target at a fixed rate while keeping the target in the center of the field of view. At a range of approximately ten feet from the target, the tracking sensor switched from the long range camera to the short range camera. At a range of approximately four feet, the platform was halted and the tracking sensor switched from the homing mode to the contour following mode. Here the tracking sensor tracked a structural feature on the target to guide the platform around the target. During this maneuver, the long range camera remained in a forward orientation while the short range camera was rotated on the rotary table to maintain tracking of the target. After completely circling the target, the platform was halted and the cameras realigned. At this point, the tracking sensor and platform were ready for further commands from the operator. The "heel" mode was also demonstrated to illustrate the autonomy of the tracking sensor. After a manual lock-on of the target, the demonstration proceeded to lead the tracking sensor and platform around the testbed. Various starts, stops, turns, pivots, and obstacle avoidance maneuvers were demonstrated.

Tracker Upgrades

The tracker configuration used in these demonstrations was developed in 1981. Upgrades to increase its computing capability, reduce its size, and lower its power consumption are being implemented. CMOS devices will be used allowing the processor speed to be increased from 5 to 10 MIPS for the array processor and from 300 KIPS to 1 MIPS for the executive processor. A floating point chip will be added. The pixel rate will be increased from 5 to 15 MHz. Several boards (video preprocessor and interface) will be reduced to a single 300 X 300 mil chip. Overall, the power consumption of the packaged tracker will be reduced from 200 to 25 W and the number of cards from 11 to 5. It is anticipated that a version of this packaged tracker could be used in the Orbiter cabin.

Advanced Robotic Demonstrations

In coordination with NASA-JSC, the MDAC tracker will be interfaced with two of NASA's operational systems: 1) Man Maneuvering Unit (MMU), 2) One-G version of RMS.
When interfaced to the MMU, the tracker will provide both target identification and guidance cues to the control systems for autonomous operations. Likewide, when interfaced to the 1-G RMS arm, the tracker will guide the end effector to targets which were manually acquired by an operator. Cameras mounted on the MMU and 1-G RMS will provide the necessary imagery for the tracker.

CONCLUSION

The capability of existing, packaged tracker hardware to perform autonomous homing and contour following with minor upgrades were functionally demonstrated. Upgrades in hardware and software will be required to address the requirements of space operations. However, a great deal of the basic development haave already been and are being performed and funded by other government agencies. The demonstrations with the MMU and 1-G RMS arm will provide additional information on the integration of this technology with existing systems for near term space operations.

ACKNOWLEDGEMENTS

The support of Harry Erwin, Bob Lacy, and Ann Marie Ching of NASA and Al Eisenberg of MDAC-Houston in the planning and execution of this effort is gratefully acknowledged.

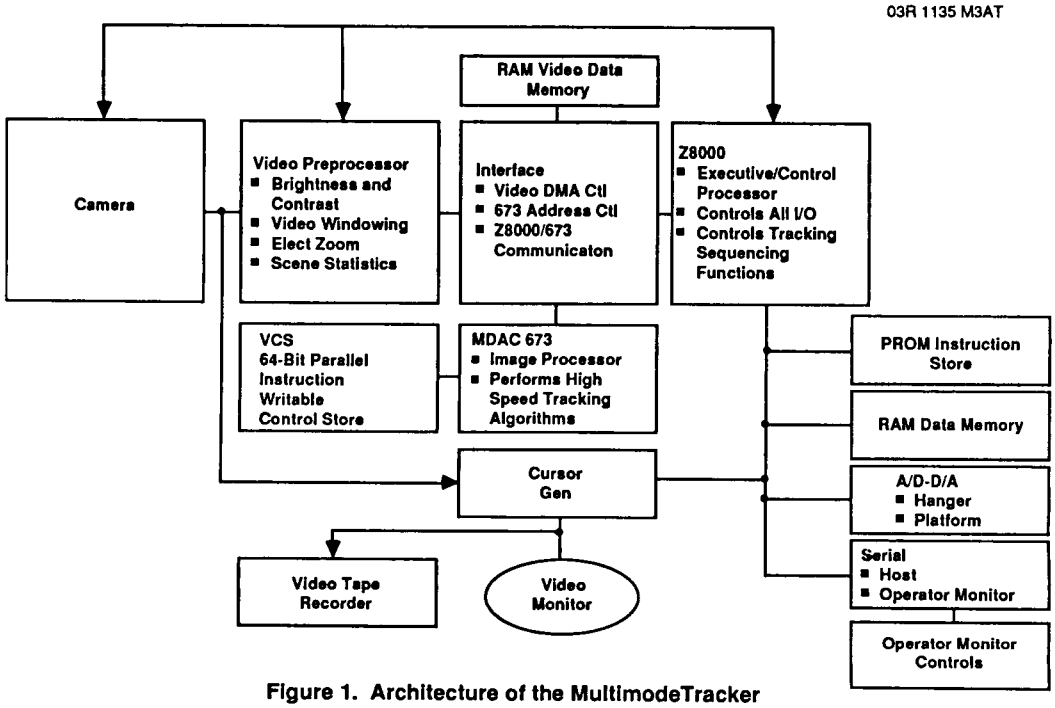


Figure 1. Architecture of the MultimodeTracker

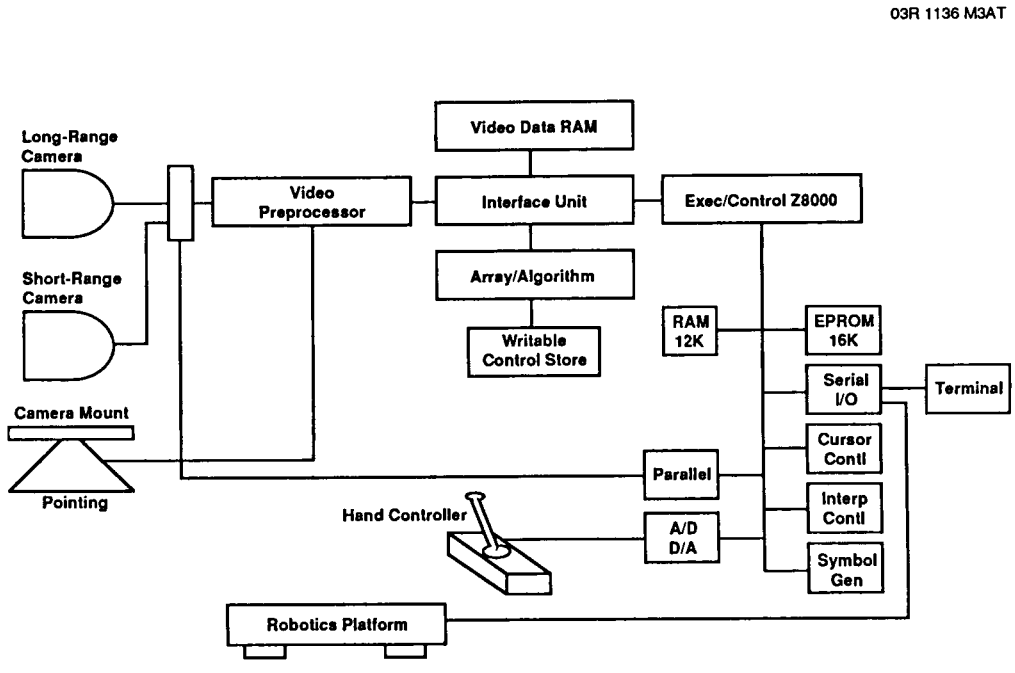


Figure 2. MDAC Tracking Sensor/Mobile Platform Integration Diagram

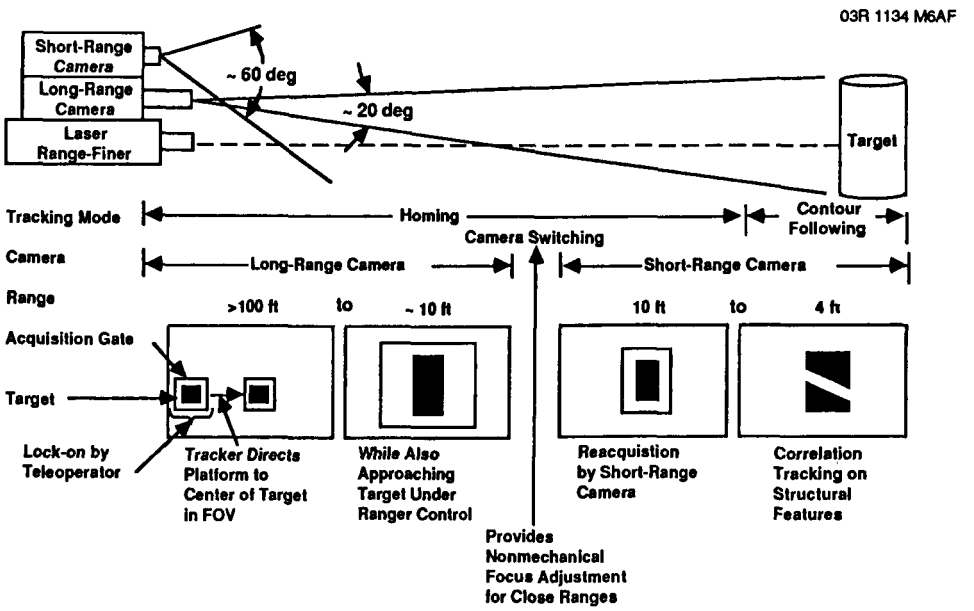


Figure 3. Demonstration Format

CALIBRATION OF NEURAL NETWORKS USING GENETIC ALGORITHMS, WITH APPLICATION TO OPTIMAL PATH PLANNING

Terence R. Smith

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93106

Gilbert A. Pitney

Department of Computer Science
University of California at Santa Barbara
Santa Barbara, CA 93016

Daniel Greenwood

Netrologic
4241 Jutland
San Diego, CA 92117

ABSTRACT

Genetic algorithms (GA's) are used to search the synaptic weight space of artificial neural systems (ANS) for weight vectors that optimize some network performance function. GA's do not suffer from some of the architectural constraints involved with other techniques and it is straightforward to incorporate terms into the performance function concerning the metastructure of the ANS. Hence GA's offer a remarkably general approach to calibrating ANS. GA's are applied to the problem of calibrating an ANS that finds optimal paths over a given surface. This problem involves training an ANS on a relatively small set of paths and then examining whether the calibrated ANS is able to find good paths between arbitrary start and end points on the surface.

1. INTRODUCTION AND PROBLEM STATEMENT

Massively parallel computing devices composed of many elementary processing elements (PE's), connected in a simple and local manner, offer the possibility of computing complex input-output relationships relatively quickly. One approach to achieving massive parallelism involves the use of many identical and simple processing elements (PEs) and relatively local communication links between PEs. The artificial neural system (ANS) represents one class of such systems that are currently the object of much investigation. Each PE of an ANS produces as its output a single, bounded, real-valued number. An ANS is a collection of PEs, each of which takes as input the weighted outputs of other PEs. The ANS architectures considered in this paper consist of networks of synchronous, binary threshold units (BTU's, see Egecioglu, Smith and Moody, 1987).

The behaviour of the ANS (i.e., the mapping it is able to compute) is largely determined by the set of weights by which the output of a given PE is multiplied before being taken as input to another PE. A major problem in ANS design involves the determination of an appropriate set of connection weights between the PE's for computing a given mapping.

In this paper, our primary focus of attention concerns an essentially unexplored technique for programming ANS's, namely genetic algorithms (GA's). A secondary focus of attention concerns the construction of an ANS that is able to compute "good" paths over some surface, using GA's and a set of input-output exemplars to program the system. In particular, we are interested in the ability of this programming technique to construct an ANS that significantly generalizes over the set of input-output pairs.

1.1. Research Reported in this Paper

The research reported in this paper is of an exploratory and empirical nature, since the behaviour of both ANS's and GA's are currently difficult to analyse in a formal manner. Our basic approach to the problem involves:

a) construction of a surface over which "good" paths are to be computed and computation of globally optimal paths between all given pairs of points on the surface (using the Dijkstra algorithm) to produce a training set of input-output patterns (start-end points, optimal paths)

b) establishment of a priori constraints on the architecture of the ANS

c) choice of which variant of GA to employ in calibrating the ANS

d) a set of training runs in which a subset of input-output patterns are used to program the connection weights

e) tests of the ANS on the remaining input-output patterns to determine how well the GA performs in generalizing over its training set.

The main purpose of the experiments reported here was to provide intuition into the application of GA's for calibrating ANS's, particularly in relation to the path planning problem. More systematic investigations of the problem are now in progress.

2. NEURAL NETWORK PROGRAMMING METHODS

Two main problems in ANS design are 1) finding a suitable network architecture, in terms of connection topology and type of PE's, and 2) determining the weights of the ANS. To date, no automatic procedure for designing a network architecture for a given input/output behaviour exists, although as discussed later, GA's may be applied to solve this problem (we apply intuition as a guide to designing the architecture). The main purpose of this paper is to propose a relatively new solution to the second problem.

We define the programming of an ANS as adjusting the weights such that the network can compute a desired input to output mapping. There are numerous techniques for programming an ANS, many of which are best suited for particular problem domains, or limited to specific network architectures. These techniques may be currently classified into two groups, the first of which is based on finding connection weights in terms of predetermined functions of the problem parameters and the second of which is based on some form of search over the space of weights.

The first class includes associative memory techniques (Hopfield, 1982) and techniques based on finding quadratic forms that express a problem in terms of a set of constraints (Hopfield and Tank, 1985).

Concerning the second class, one may classify the techniques according to the degree of "localness" of the search procedure. Most learning procedures perform a search over the weight space to minimize some performance criterion of the network. The search techniques include gradient descent, gradient descent with annealing, and guided random search. Examples of such techniques include, respectively, back propagation (Rumelhardt and McClelland, 1986); the master/slave formalism (see Lapedes and Farber, 1986); and guided accelerated random search (GARS, Mucciardi, 1972).

In this paper, we propose the genetic algorithm as a neural network programming procedure.

2.1. Genetic Algorithms

The GA can be viewed as a relatively global search procedure based upon population genetics (Holland, 1975). We apply the GA as a function optimizer to the weight space of an ANS to maximize some performance function of the network. The major strengths of the GA as a function optimizer are its ability to search efficiently and effectively high dimensional, multimodal, noisy, and discontinuous surfaces. Since the GA is being used purely to search the weight space, there are no restrictions on network architecture. There are also no restrictions on the terms of the performance function.

The basic GA maintains a population of individuals. In the case of the function optimization problem, each individual represents a point in the parameter space of the performance function, and is represented by a binary string encoding of the parameter vector. Each individual is evaluated, and a new generation is produced by selecting individuals on the basis of their performances for reproduction. Because higher performing individuals are selected more often for reproduction, and due to the recombination effects of the crossover operator, there is a pressure towards higher performing individuals being accepted into the population.

The basic algorithm is:

1. Randomly generate a population, P_0, of N members. Set t=0.
2. For all i=1..N, compute and save the performance measure $\mu(P_i^t)$.
3. If converged, then STOP. Best individual of last population is solution.
4. Compute selection probabilities
$$\rho_i^t = \mu(P_i^t) / \sum_{j=1}^{j=N} \mu(P_j^t)$$
5. Generate next generation, P^{t+1}, by choosing individuals via selection probabilities for reproduction using genetic operators. Set t=t+1. Goto 2.

The genetic operators used in this paper are crossover and mutation. Crossover recombines two parent vectors to produce an offspring vector by concatenating the segment to the left of a random crossover point in the first parent with the segment to the right of the same crossover point in the second parent. The mutation operator, with a low probability, alters bits in the offspring. The combined effect of crossover, mutation and selection allows genetic algorithms to search very high dimensional spaces efficiently.

One of the most challenging problems in ANS learning procedure design is the assignment of credit to processing elements which are responsible for a system's high performance, especially when those elements are only active early in a long chain of actions which eventually leads to reward from the environment. The GA solves the credit assignment problem by selection. Individuals which contain good weight vectors are rewarded by a higher probability of recombination and reproduction. Thus the weights are held accountable for network performance.

3. THE APPLICABILITY OF THE ANS AND ITS ASSOCIATED PROGRAMMING METHODS TO THE PATH PLANNING PROBLEM

A secondary goal of this investigation is to program an ANS in such a manner that it contains an efficient, internal representation of a "cost" surface characterized in terms of some set of efficient paths over the surface. This representation should permit the network to compute a "good" path between two arbitrary points on the surface, given only those two points. Since only a subset of the precomputed optimal paths over the surface are presented to the network during the learning phase, the network must be able to generalize.

In most of the work to date on the programming of ANS's to compute specific functions, researchers have employed the stable states of the ANS as a basis for representation. For any ANS, there is a fixed number of such states. Hence the ability of an ANS to compute a given function is ultimately limited by this capacity constraint. However, different approaches to representing a given computation may result in more or less efficient ANS. Hence part of our research has concentrated on different approaches to network representations and their relative efficiencies.

4. A PRELIMINARY INVESTIGATION OF GA'S FOR PROGRAMMING ANS'S TO SOLVE THE PATH PLANNING PROBLEM

GA's may be used to modify the synaptic weights of the ANS in order to maximize the net's performance in finding optimal paths. The resulting network ideally accepts an input pattern representing start and end positions on a given surface, and produces an output pattern representing a least cost path from the given start and end points.

4.1. A Priori Hypotheses Concerning the Topology of the Connection Weights

As noted above, the topology of the connection weights may be an important factor in determining the efficiency of a network with a given number of PE's. Hence we explored four alternative topologies, while keeping the number of "hidden" PE's constant at 20.

A quad tree structure was suggested by prior experience with computational architectures for solving path planning problems (Smith and Parker, 1987). This architecture embodies the hypothesis that the pertinent features of the landscape required for the ANS to predict optimal paths can be best represented in a hierarchical fashion, with the more abstract, higher order features of the surface encoded at the top of the hierarchy. It presumes that computation proceeds from the top downward, with higher levels guiding (constraining) the computation at lower levels of the tree. We examined three such architectures:

a) FFQ is a feed-forward quad tree structure with 4 layers (see Figure 1)

b) RQ (see Figure 2) is a modification of FFQ with recurrent connections between layers

c) RQNNN is the same as RQ, except for the addition of next nearest neighbor connections between units on a layer (see Figure 5)

d) FIH (see Figure 3) involves 20 fully-connected hidden units.

Quad tree topology is shown in Figure 4.

4.2. The Landscape

The surface investigated is derived from a topological map of a 40 square kilometer area of the Sierra Madre Mountains in California mapped onto an 8x8 square grid of pixels. Each pixel is represented as a node in a four-connected transition cost graph, in which each link represents a bidirectional, symmetric cost. The derived cost graph is then used as input to Dijkstra's algorithm to compute optimal paths, and to the GA's objective function in order to gauge the performance of each ANS.

4.3. Objective Functions

The GA uses an objective function to evaluate the performance of each member of the population. In this case, the individual is an ANS, and the task is to predict the optimal path over a surface between two points. After presentation of the input pattern to the ANS, and after the network relaxes, the objective function computes an error measure between the optimal path predicted by the network and the true optimal path. The network predicts a path by turning on those neurons in the output layer which correspond to nodes in the transition graph, which in turn correspond to points on the surface.

It is helpful to choose a performance measure which facilitates the genetic search. The objective function is a mapping from the weight space of the ANS to a single performance value. As a general rule, the objective function should posses some degree of 'smoothness' in the region about the solution point in the weight space. This means that any change in the weights in the direction of the optimum should yield a higher performance value. For a discussion of how various types of objective function surfaces affect search procedures, see Ackley (1987).

Two basic performance functions are used in these simulations. The first function, P1, incorporates three terms: 1) an incorrect link cost, 2) an incorrect pixel cost, and 3) convergence time, as defined below:

$$x = \sum_j [abs(l_j^{opt} - l_j^{net})(l_j^{opt} cost(l_j^{opt}) + l_j^{net} cost(l_j^{net}))] + 20\sum_i [abs(i^{opt} - s_i)] + T$$

$$P_1 = 100/(1+x) \qquad (4)$$

Where, all references to links and neurons refer to the output layer, and

l^{opt} = { 1 if link is between adjacent neurons on the optimal path; 0 otherwise}.
l^{net} = { 1 if link is between adjacent neurons turned on by the network; 0 otherwise}.
cost(l) = cost of traversing link l.
i^{opt} = { 1 if the ith pixel lies on the optimal path; 0 otherwise}.
s_i = state of neuron i.
T = relaxation time of the network.
j = index over all links in output layer.

The incorrect link cost term penalizes the network for predicting paths which either 1) contain a linkage between two adjacent neurons which does not exist in the optimal path, or 2) lack a linkage which does occur in the optimal path. The amount of penalization is just the sum of the costs of traversing such linkages. The incorrect pixel count term penalizes the network for predicted paths which either 1) contain points which do not appear in the optimal path, or 2) lack points which do appear in the optimal path. The amount of penalization is proportional to the number of such incorrect pixels. The third term is the number of time constants the net takes to relax. The network is said to have relaxed when the activity pattern of the output layer has remained constant for seven time constants.

The second performance function, P2, is designed to overcome the apparent deficiencies in P1, and also to stress to the network the importance of well-formed paths. Note that the second term in P1 enforces the constraint that pixels predicted by the network lie on the optimal path. However, it will also penalize a

predicted path of near optimal cost if that path does not geographically coincide with the optimal path. This, to some degree, violates the smoothness criterion for good objective functions. Thus, the second term in P1 is replaced by two terms which impose the constraint that the output pixels be on a well formed path, not necessarily the optimal. The first term is modified to some degree, violates the smoothness criterion for good objective functions. Thus, the second term in P1 is replaced by two terms which impose the constraint that the output pixels be on a well formed path, not necessarily the optimal. The first term is modified slightly from P1, but still enforces the optimal path constraint:

$$x = 20[\sum_k [s_k(1 - ncount(k))^2] + \sum_h [s_h(2 - ncount(h))^2]] + abs(C - \sum_j cost(l_j^{net})) + T$$

$$P_2 = 100/(1+x) \qquad (5)$$

Where

ncount(i) = number of neighboring neurons of i which are on.
C = the cost of traversing the optimal path.
k = index over both path endpoints.
h = index over all other points.

Note that in P_2 the terms enforcing well-formed paths are weighted most heavily.

4.4. Representation of the Weight Vector

Generally, the ordering of gene values in the GA control string can strongly affect convergence, especially in the absence of an inversion operator. The control string is a binary string encoding of the weight vector. Each weight is encoded in 8 bits in two's complement binary, and ranges in value from -128 to 127. The weights are then concatenated to make up the control string.

Two ordering schemes are used. In the first, called LR, the weights ordered from left to right correspond to a top down ordering in the network. For example, weights on connections to the 2x2 hidden layer in the FFQ network are encoded at the leftmost end of the control string. The second scheme, called Q, distributes the weights over the control string in quad tree order. Thus, weights on connections to the top of the network hierarchy are not grouped together, but are distributed throughout the corresponding sectors over the length of the binary string.

In the GA used in this study, crossover is the main operator for generating new weight vectors for evaluation. Since it is assumed that the abstract features of the landscape allowing the ANS to generalize will be encoded in the hidden unit weights, it is expected that encoding scheme Q will facilitate search more than scheme LR by allowing crossover to generate offspring with a greater variety of hidden weights. Scheme Q can only be applied to the quad tree networks.

4.5. Reproductive Plan 4 (R4)

The variant of GA used in these simulations is the elitist expected value model (Reproductive Plan - R4) discussed in De Jong (1975). Two genetic operators are used in this model: mutation and crossover. An elitist model transfers the best performing individual of the current population intact into the next generation. This policy slightly favours local search, and is found to speed convergence. The expected value model drastically reduces stochastic errors by replacing the use of the random variable in the selection process by a counting scheme based upon the expected value of the selection probability. This prevents any statistical fluctuation which might, for example, cause a high-performing member of the population to be overlooked during reproduction.

4.6. Training Methods

Two different training modes were used during the programming of the network. In the first, T1, the same training set of data is shown to the networks over all generations. In T2, at each generation the networks are evaluated on unique and disjoint subsets of the training set. Thus, in training mode T2, the total number of data points in the training set is the product of the constant size of the training subset per generation times the number of generations.

5. SIMULATIONS AND RESULTS

Table I summarizes the GA parameters used in the simulations. The parameters of the nine separate experiments are summarized in table II. The experimental procedure used to train and evaluate each network is discussed below.

5.1. Experimental Procedure

First, the training and test data sets were prepared. Dijkstra's algorithm was applied to the cost graph representing the landscape to find the optimal path between all pairs of points on the surface. Each data point is a tuple consisting of an input pattern encoding the start and end points, and an output pattern encoding the optimal path from the start to the end point. Subsets of the data were allocated to a training set and a test set.

The experiment proceeded in two phases. During the learning phase, the training set data was used by the GA's performance function to search the weight space of a particular ANS. After learning, the capability of the network to generalize was measured on the test data set.

5.1.1. Learning Phase

Before any particular run, the network architecture was specified. An initial population of weight vectors was randomly generated. Each member was evaluated by simulating the equations governing the corresponding ANS and measuring its performance in a series of trials in which the net attempts to complete the correct output pattern for a given corresponding input pattern.

After the net relaxed, or when the maximum time allot-

ted for the network to relax was exceeded (50 cycles in our simulations), the objective function was applied to the output layer. The average over all trials was taken as the performance of that particular net. This performance value was then used by the GA to assign selection probabilities to individuals for reproduction. The learning phase ends when the genetic search converges.

5.1.2. Test Phase

In the test phase, the best performing member of the last generation was evaluated on the test data set. To allow comparisons, the performance function used in the test phase, P3, was the same for all runs:

$$x = \sum_k [s_k(1 - ncount(k))^2] + \sum_h [s_h(2 - ncount(h))^2] + abs(C - \sum_j cost(l_j^{net}))$$

$$P_3 = 100 - x/2 \qquad (6)$$

P3 is a variant of P2, with equal emphasis on cost and path terms, without the convergence time term, and linear in x.

5.2. Description of Results.

The results of the nine runs are displayed in tables III, IV, and V.

Table V shows the number of generations each run took to converge, as well as the total number of paths in the training set. When training method T1 was used, the performance vs. generations curve was monotonically non-decreasing, and the GA was considered to converge when no improvement had been made in the performance value of the best network over 12 generations. With training method T2, since the training data was different for each generation, the performance vs. generations curve was not monotonic, and the GA was considered converged when no improvement could be seen in the average performance of the best network over about 20 generations. Run 6 took a long time to converge, and was stopped at 288 generations. It should be noted that run 6 converged with respect to performance on long length paths. Before it was stopped, it was continuing to increase in the performance on short to medium length paths.

Table III shows the value of the performance (P3) of the best network of the last generation of each run on the training data. Entries marked by a dash in the table signify that no paths of length indicated by the column heading were contained in the training set. Performance values are sorted by path length, and are averaged over the number of paths of that length in the training set, which varied from run to run. A performance of 100 is a perfect score, indicating that the network correctly predicted the optimal path.

The main results of this work are given in table IV. The performance (P3) is calculated as in table III, but using the data in the test set, and averaged over a constant number of trials per path length, as indicated in the last row of the table.

5.3. Discussion of Results

Although none of the ANS's did a perfect job of consistently predicting optimal paths during the test phase, we gained some insight into the problems of training a network to generalize, of weight vector representation in the GA and of network architectures for this problem.

5.3.1. Performance on the Training Data

Referring to table III, performances using the training data in runs 1, 3, 4, and 5 show the ability of the best network to correctly predict the single training path of length 15 pixels. Only one pixel was missing from the middle of the predicted path in run 3, giving that network a suboptimal score of 98. Run 2 shows the performance on 5 training paths of lengths ranging from 11 to 15 points. In this run, one path was predicted correctly, one had a few extra pixels in the output layer turned on, and 3 paths were halfway complete.

The networks of runs 1 through 3 did not use their hidden units in predicting the optimal path. Only when the architecture was changed from FFQ to RQ and FIH in runs 4 and 5, respectively, that is, when bottom up connections were added, did any hidden units come into play.

Runs 6 through 9 were trained using one or five different paths per generation. This training method always caused the best network to use its hiddens in the computation, and led to better generalization capabilities on the test data. Because of the training method used, the total training set sizes in runs 6 through 9 were much larger, as shown in table V. There are no significant differences between the performances of the nets in runs 6 to 9 on the training and test data. These networks made greater use of their hidden units, and learned early in the training phase to generalize. This was expected, since the T2 training method does not allow any one path to be seen by a network for more than one generation, thus discouraging the 'memorization' of a specific pattern.

5.3.2. Performance on the Test Data

The measures of performance of the best networks on the test data give some indication of their ability to generalize (see table IV).

5.3.2.1. Best Networks

The network which had the most consistently high average performance over all path lengths was that of run 8 (RQ,P2,T2,5,Q). Given any two points as input, the network often made a reasonable approximation to a path between them, keeping disconnected pixels to a minimum. Short to medium length paths would sometimes complete correctly, but the network had trouble with longer paths.

The next best network, in terms of generalization capability, was that of run 6 (FIH,P1,T2,1,LR), which performed very well on short to medium length paths, but did much more poorly on longer paths.

Though the network of run 5 (FIH,P1,T1,1,LR) shows high performance values for medium to long length paths, its ability to generalize was nil. No matter what the input pattern to this network, the output pattern would usually be the same path used in the training phase. P3 would score the pattern highest for long optimal path lengths because the path was connected, and usually had a traversal cost similar to the optimal path of the given inputs. The only cost penalty was in the lack of connections to the true path endpoints, which is small.

5.3.2.2. Comparisons Between Runs

Despite the paucity of runs, a comparison of simulation results in table IV suggests some interesting, though inconclusive, results.

Comparing runs 1 through 5 with runs 6 through 9 indicates that the training method T2, i.e., showing each generation a different training set, is sufficient to produce generalization capability in the networks. It is not known whether T1, with a much larger training set size, would also induce generalization.

A comparison of the results of runs 1 and 3 show that the quad tree ordering of weights in the genetic control string gave a slight improvement in performance over a simple top down encoding.

Comparing runs 1 and 4 suggests that bottom up connections increased performance.

Run 7 was somewhat of an anomaly, in that it should not have differed much from run 6. We believe that, for the number of alleles in the chromosome, the population size was too small, and the GA had insufficient gene variability to sustain a global search, and thus found a local minimum.

Comparing runs 8 and 9 indicate that adding next nearest neighbor connections within layers actually decreased generalization capability. In fact, a consideration of the qualitative observations on the data of runs 1, 4 and 5, which were all trained with T1 and one training pattern, shows that the addition of feedback connections widens the basin of attraction about the single learned memory vector. Thus, the networks with more feedback connections, and trained under T1, more often produced the same training pattern as output independent of the input pattern.

6. CONCLUSIONS

We have proposed a general technique for programming ANS's using GA's. Unlike most techniques, the GA imposes no constraints on network architecture or performance function. As a result, novel terms, relating not only to the network's performance in the particular task environment, but also to meta variables of the network, may be incorporated. For example, our objective functions P1 and P2 included the network convergence time as a term to be minimized. In run 5, on the training set, the GA found a weight vector capable of perfectly predicting the optimal path after 73 generations, with a network convergence time of 14 time constants. After 20 more generations, the GA had decreased the convergence time to 12, then finally to 10 time constants. Such a criterion as speed of computation would be difficult to incorporate into most other network programming procedures.

Concerning the solution to the optimal path planning problem, it is apparent to us that 20 hidden BTU's is insufficient to solve the problem as posed. A major limitation is the number of stable states that are feasible using an ANS with only 20 hidden units. The path planning problem is hard for the network to solve because of the minimum of input information, and because it convolves two problems, namely finding well-formed paths and finding optimal paths. The fact that the GA could not find a weight vector to solve the problem was because of architectural constraints. We doubt that any other learning procedure could have solved this problem with the given number and type of neurons.

Even though the networks were not able to always predict optimal paths, the simulations showed us the importance of knowledge guided search through the experimental parameter space.

This work suggests two directions for future research. First, for the short term, a more rigorous experimental approach is needed to explore network architectures for solving the path planning problem. It appears that a hierarchical network architecture, with bottom up feedback, is the most promising structure. The number of hidden units and the power of the PE's should also be increased.

The second and more fundamental area of research involves the first problem of ANS design: finding a suitable network architecture for a particular problem. This includes number and type of PE's, and connectivity. Here especially the GA appears to be a natural candidate solution, because of its role in the evolution of the human nervous system.

The solution would involve finding a good encoding of an ANS architecture in terms of a representation suitable for manipulation by the GA, and a developmental plan to translate that encoded representation (genotype) into the corresponding network (phenotype). Such a plan may be a set of growth rules, of the type discussed in Lindenmayer (1976) or proposed recently by Wilson (1987). The performance of each network can then be evaluated in the given task environment. Furthermore, if the network is able to learn during its evaluation phase, that is, if the connection strengths are not solely determined by evolution, Hinton and Nowlan (1987) argue that the learning capability would provide an easier 'evolutionary path' toward the optimal network architecture.

7. REFERENCES

Ackley, D.H., "Stochastic Iterated Genetic Hill-Climbing," Doctoral Dissertation, Carnegie-Mellon University, Pittsburgh, PA, 1987

De Jong, K., "Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph.D. thesis, Dept. Comp. and Com. Sciences, Univ. of Michigan, 1975.

Egecioglu, O., Smith, T. R., and Moody, J., "Computable Functions and Complexity in Neural Nets," in J. Casti

(Ed.), Proceedings of the Abisko Conference on Neural Computation, Elsevier (in press), 1987.

Hinton, G., and Nowlan, S.,"How Learning Can Guide Evolution," Complex Systems, Vol. 1, No. 3, June 1987, pp. 495-502.

Holland, J., *Adaptation in Natural and Artificial Systems*, Univ. of Michigan, Ann Arbor, 1975.

Hopfield, J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," Proc. Natl. Acad. Sci. USA, Vol. 79, 1982, pp. 2554-2558.

Hopfield, J. and Tank, D., "'Neural' Computation of Decisions in Optimization Problems," Biological Cybernetics,52,141, 1985.

Lapedes, A., and Farber, R., "Programming a Massively Parallel, Computation Universal System: Static Behavior," Conf. Proc. of Neural Networks for Computing, Snowbird, UT, 1986, pp. 283-298.

Lindenmayer, A., and Rozenberg, G. (eds.), *Automata, Languages, Development*, Amsterdam: North-Holland, 1976.

Mucciardi, A.N., "Neuromime Nets as the Basis for the Predictive Component of Robot Brains," *Cybernetics, Artificial Intelligence, and Ecology*, H.W. Robinson and D. E Knight(Ed.s), Spartan Books, Bensalem, PA., 1972, pp. 159-193.

Rummelhart, D., and McClelland, J., *Parallel Distributed Processing: Exploration in the Microstructure of Cognition*. MIT Press, 1986, Chapter 8.

Smith, T. R. and Parker, R. E., "An Analysis of the Efficacy and Efficiency of Hierarchical Procedures for Computing Trajectories over Complex Surfaces," European Journal of Operations Research, 30,1987, pp. 327-338.

Wilson, S., "The Genetic Algorithm and Biological Development," Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms, July 28-31, 1987, MIT, Cambridge, MA. pp. 247-251.

8. FIGURES

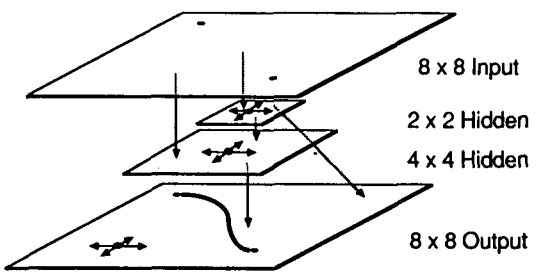


Figure 1 - Feed Forward Quad Architecture

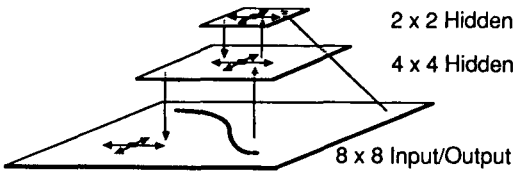


Figure 2 - Recurrent Quad Architecture

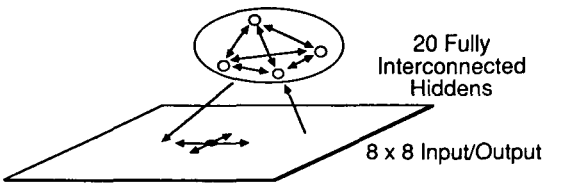


Figure 3 - Fully Interconnected Hiddens

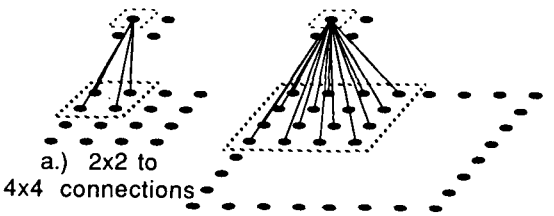


Figure 4 - Quad Tree Topology

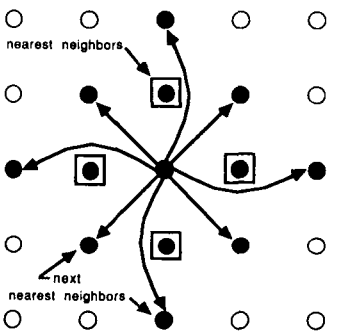


Figure 5 - Next Nearest Neighbor Connectivity

9. TABLES

Table I
Genetic Algorithm Parameters

Parameter	Value
Model	R4
Populations Size	400
Crossover Rate	0.95
Mutation Rate	0.01

Table II - Simulation Parameters

Run	Architecture	Objective Function	Training Method
1	FFQ	P1	T1
2	FFQ	P1	T1
3	FFQ	P1	T1
4	RQ	P1	T1
5	FIH	P1	T1
6	FIH	P1	T2
7	FIH	P2	T2
8	RQ	P2	T2
9	RQNNN	P2	T2

Run	# Paths Shown Per Generation	Weight Vector Encoding	Generalization Capability
1	1	LR	
2	5	LR	
3	1	Q	
4	1	LR	
5	1	LR	
6	1	LR	2nd
7	5	LR	
8	5	Q	1st
9	5	Q	3rd

Table III - Performance on Training Data Paths

Run No.	Optimal Path Length 2	3	4	5	6	7
1	-	-	-	-	-	-
2	-	-	-	-	-	-
3	-	-	-	-	-	-
4	-	-	-	-	-	-
5	-	-	-	-	-	-
6	92	92	91	90	87	87
7	11	22	17	22	20	24
8	86	84	85	87	85	87
9	42	49	51	53	55	58

Run No.	Optimal Path Length 8	9	10	11	12	13	>13
1	-	-	-	-	-	-	100
2	-	-	-	81	81	81	73
3	-	-	-	-	-	-	98
4	-	-	-	-	-	-	100
5	-	-	-	-	-	-	100
6	82	83	78	75	75	74	73
7	33	26	40	39	50	37	53
8	85	86	83	84	85	79	84
9	59	63	66	69	71	73	76

Table IV - Performance on Test Data Paths

Run No.	Optimal Path Length 2	3	4	5	6	7
1	-44	-39	-37	-38	-36	-28
2	4	-7	-5	-14	-12	-13
3	10	12	7	16	18	13
4	22	29	33	31	29	28
5	72	77	80	82	85	88
6	91	93	92	91	90	85
7	14	18	17	29	27	32
8	86	85	82	85	86	87
9	45	48	51	51	56	60
No. of Trials Averaged	32	71	79	61	98	75

Run No.	Optimal Path Length 8	9	10	11	12	13	>13
1	-33	-36	-28	-27	-27	-16	4
2	4	0	13	16	19	23	23
3	11	22	20	16	23	29	35
4	40	47	40	37	45	41	52
5	88	89	89	89	88	88	90
6	82	81	78	77	74	71	72
7	29	35	34	34	37	44	45
8	85	86	82	80	80	81	86
9	60	62	66	69	69	72	75
No. of Trials Averaged	83	44	55	32	35	19	17

Table V - Convergence Times and Total Training Set Size

Run	Generations To Converge	Training Set Size
1	21	1
2	82	5
3	25	1
4	63	1
5	73	1
6	288*	288
7	40	200
8	100	500
9	199	995

*This run was stopped before convergence; see text.

ACKNOWLEDGEMENTS

We wish to thank NASA and VERAC, Inc. for supporting this research.

AN EFFICIENT REPRESENTATION OF SPATIAL INFORMATION FOR EXPERT REASONING IN ROBOTIC VEHICLES

by Steven Scott
and Mark Interrante

Texas Instruments
Corporate Artificial Intelligence Lab
Dallas, Texas

ABSTRACT

The previous generation of robotic vehicles and drones were designed for a specific task, with limited flexibility in executing their mission. This limited flexibility arises because the robotic vehicles do not possess the intelligence and knowledge upon which to make significant tactical decisions. Current development of robotic vehicles is toward increased intelligence and capabilities, adapting to a changing environment and altering mission objectives. The latest techniques in Artificial Intelligence (AI) are being employed to increase the robotic vehicle's intelligent decision making capabilities. This document describes the design of the SARA spatial database tool at Texas Instruments, which is composed of request parser, reasoning, computational, and database modules that collectively manage and derive information useful for robotic vehicles.

1. INTRODUCTION

In order for future autonomous systems to efficiently reason about their environment it is necessary for them to maintain a consistent world model. Such models impose increasing demands for the supply, storage, and maintenance of large knowledge or information bases. While conventional database management systems (DBMS) perform much of the storage and retrieval of information, they are generally not powerful enough to deal with multi-dimensional (N-dimensional) world data or intelligently reason about the data they contain.

In support of Texas Instrument's work in avionic expert systems, a tool is being built to support the construction, maintainence, and usage of world models. The SARA spatial database retrieves, analyzes, abstracts, and maintains large multi-dimensional information bases that require efficient retrieval based of the spatial location of the data objects. The system manipulates both factual data, such as "the location of an airport", and derived abstractions, such as "areas of danger to low flying slow aircraft". Also provided is inferencing based on changing database values and the ability to perform complex algorithmic computations on the data. As the SARA tool continues to develop, the bonds between databases, algorithmic computations, and expert system reasoning will certainly be strengthened and more closely unified.

The examples are from a sample application of SARA in the airspace environment. The system is quite modular and flexible, allowing easy customization to specific applications.

2. SYSTEM ARCHITECTURE OVERVIEW

The SARA tool is composed of four modules: the request parser, triggering, algorithmic computation, and database. Each of these modules contain components for module communication and request management. Figure-1 shows the four SARA modules and how application programs using SARA communicate directly and only with the request parser.

2.1. REQUEST PROCESSING

The SARA system has an integrated syntax with nesting of event triggering, computation, and database operations. Each module has an associated request queue manager that is responsible for regulating its work flow and keeping track of paperwork. The module itself is allowed to process operations uninterrupted by the hassles of task management. This flexibility has led to the design of the SARA system shown in figure-2.

In the SARA system, requests arrive from the application program and conform to the SARA grammar. Requests enter the system through the request parser. After the request parser validates the syntax of the request, it ships the request to its request queue manager. The request queue manager then splits the requests into tasks for specific modules to handle. Tasks may be nested, requiring the services of another module. When the request queue manager of the intended module (i.e. the triggering module) receives tasks from the request queue manager of another module (i.e. the request parser), it schedules time with its module for completion of the task. At this point, the task does not contain uncompleted tasks for other modules. This task is atomic for the module and referred to as an operation.

2.1.1 REQUEST PARSER MODULE

The request parser module verifies the request syntax , initially splitting the request into tasks, and forwarding the tasks to the appropriate module for processing. After validating request syntax and contracting out the tasks to the other modules, the request parser continues with the next request. When the final result from the tasks is produced, the request parser responds to the application program with the result.

2.2 ALGORITHMIC COMPUTATION MODULE

The computation module is designed to contain compute bound algorithms which may benefit from the use of special numerical hardware. Depending on the application program, these algorithms could be short-term or background low priority processing. Since the SARA system is highly modular, the computation module could reside on a different machine running whatever language is available for the machine. Additionally, multiple computation modules could be established, each performing a specific range of tasks and thus giving the user the ability to configure a SARA system to solve the application problem.

2.3 SPATIAL DATABASE MODULE

The spatial database (SDB) provides rapid processing of multi-dimensional data within a relational framework. Applications needing spatial retrieval of information range from VLSI computer aided design to robotics. The SDB module accepts a SQL-like grammar sentence as input and translates it into syntax for the Texas Instrument's Relational Table Management System (RTMS). The SARA SDB approach is based on manipulating entire objects rather than breaking the spatial representation down, as in quad-trees. This processing is supported by the spatial index, which is a variation of the R-tree [Interrante 87]. The spatial index provides for efficient retrieval of information or objects possessing clustering characteristics that, in theory, would yeild less page faults and better performance than traditional database indexes. The SDB module consists of four components: the machine specific database tool (RTMS), the SQL to RTMS syntax translator, boundary data types and operators, and the spatial index.

2.3.1 RELATIONAL TABLE MANAGEMENT SYSTEM

The Texas Instrument's Relational Table Management System (RTMS) is a relational database for the Explorer lisp machine. Since RTMS is built on top of the lisp environment, any predicate lisp expression can be used in the where clause of a query as a condition for tuple selection. RTMS has been enhanced with special purpose data types and predicates for the spatial database. The following RTMS query produces the airplanes at Ohara airport. In other words, retrieve the airplanes whose position is enclosed by the boundaries of Ohara airport.

```
(RETRIEVE airplanes WHERE
   (enclose
      (RETRIEVE airports
         PROJECT (boundaries)
         WHERE (= airport-id "Ohara"))
      aircraft-position))
```

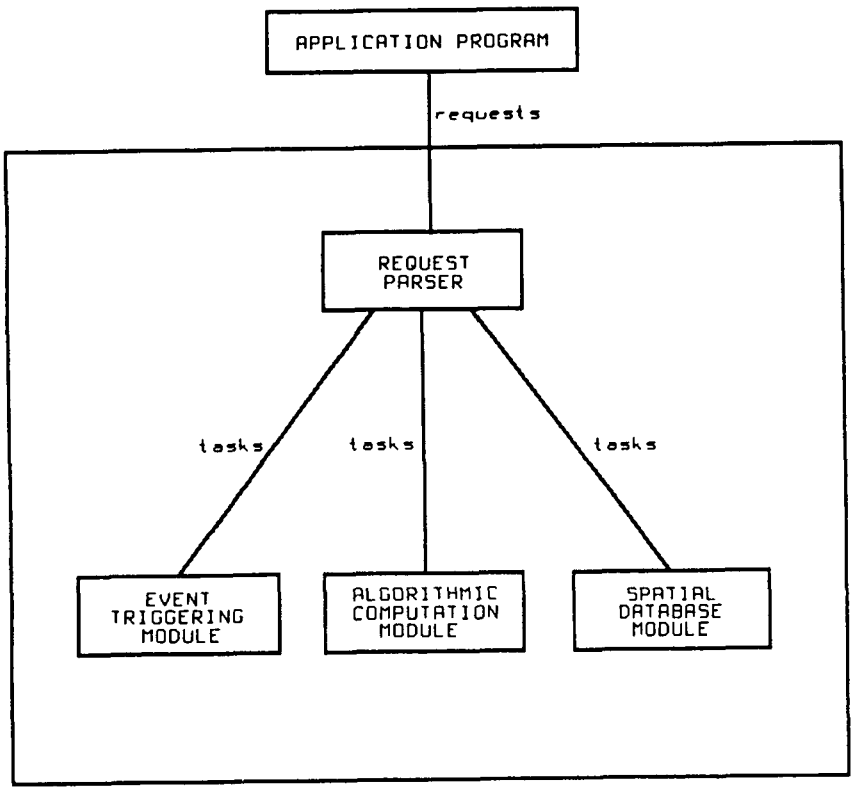


FIGURE 1. Overview of the SARA System Architecture.

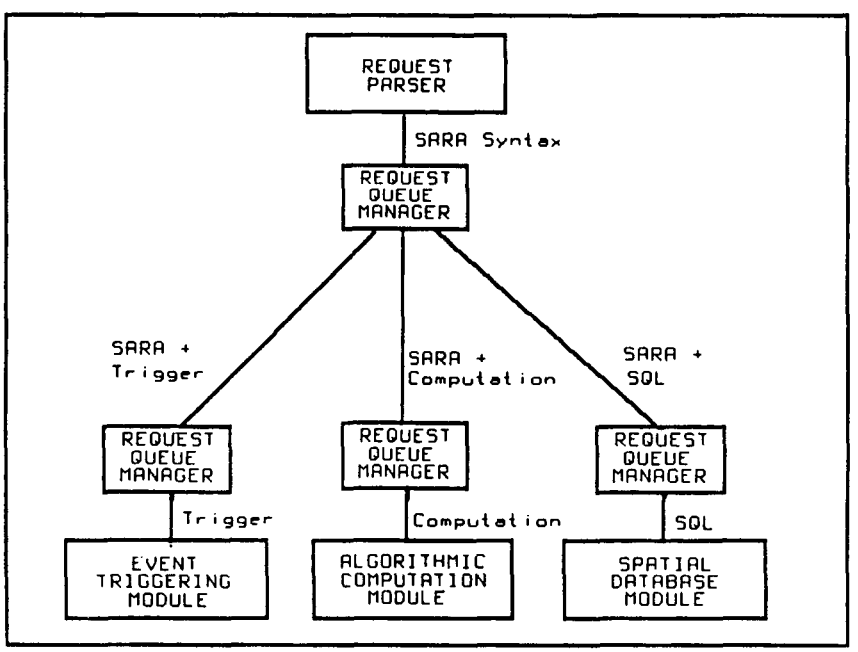


FIGURE 2. Internal Components of SARA Showing How the Sentence Composition Changes During the Life of a Request.

2.3.2 BOUNDARY DATA TYPES AND OPERATORS

In addition to the the usual database data types, a set of data types representing the boundaries of multi-dimensional objects have been defined specifically for the spatial database. The currently defined boundary data types are: rectangular solid, point, and cylinder. Future releases of the SARA system plan on supporting trajectories, cones, polygons, and ellipsoids. The operators UNION, ENCLOSE and OVERLAP are defined on the boundary data types. These operators form the basic level of a spatial database query understood by SARA.

2.3.3 SPATIAL INDEX

The SARA database provides faster access of spatial objects through the spatial index, which clusters objects based on locality. The spatial index used is a modified R-Tree, which is comparable to a multi-dimensional B-Tree. The R-tree differs from a Quad tree in that, with the R-Tree, individual objects are not decomposed into smaller components [Guttman 83]. In this implementation the actual shape of the object must be approximated by using one of the boundary data types. Support for modeling of objects at various levels of detail may be possible using persistent objects to organize the objects into a hierarchy [Thatte 86].

The spatial index provides partitioning mechanisms, allowing the relations to be clustered together based on similiar characteristics (i.e. static/dynamic and objects/regions). These partitions, defined by the database administrator (DBA) when the database is created, only influence speed and not functionality. Partition can be based on how often an object is updated (static or dynamic) or how large the object boundaries are (objects or regions). The static, dynamic, object, and region classifications are only examples of the partitioning advantages, the definition and use being under full user control.

2.4 EVENT TRIGGERING MODULE

The event triggering module is responsible for setting up, maintaining, and checking conditions on which the application program has asked to be notified. The action taken when the event or condition is triggered can be a notification to the application program or some SARA task to perform. One action might be to reanalyze certain database information and derive data abstractions needed some time in the near future. Other conditions could require the application program to be periodically or continuously notified of the event. The triggering module is sufficiently general as to suit most event recognition situations.

Triggers can be set to fire in four different ways. They can fire the first time a condition becomes true or every time the condition becomes true. In addition, triggers can fire continuously when a condition is true or they can fire only when the boolean condition changes value (i.e. from true to false).

Event triggering is implemented in a forward chaining rule based system with some support from the spatial database. Triggers are written as rules and may have embedded database and computation requests. Triggers that depend on spatial location are highly efficient because the database selectively postpones evaluation of the condition until it could possibly become true, thus eliminating unnecessary evaluations.

Two of the main uses of triggers are in notifying the application user of significant changing events and the creation of data abstraction levels in the database. The creation of abstraction levels is a powerful aspect of triggers as new tables and objects in the database can be built with derived relationships and groupings of objects. Triggers can also update the abstractions as the significant events change.

3. SUMMARY

This paper described the SARA spatial database tool which is composed of request parser, database, reasoning, and

computational modules that collectively manage and derive large multi-dimensional information bases. Efficient retrieval of objects by spatial location is provided in the database. The system manipulates both the factual data and derived abstractions. Also provided is inferencing based on changing database values and the ability to perform complex algorithmic computations on the data.

ACKNOWLEDGEMENTS

The authors would like to thank all of those who have contributed to the SARA project at the Texas Instruments Computer Science Center, especially Karl Schricker, Trish Dutton and Mike Ernst.

REFERENCES

[Bentley79] Bentley, J. L., Friedman, J. H., Data Structures for Range Searching. Computing Surveys 11,4 (December 1979)

[Donelson 78] Donelson, W. C. "Spatial Management of Information", ACM SIGGRAPH (1978)

[Guttman83] Guttman, A., Stonebraker, M., R-trees: A Dynamic Index Structure for Spatial Searching. Memorandum No. UCB/ERL M83/64 Unviersity of California at Berkley (1983)

[Herot 80] Herot, Christopher F., "Spatial Management of Data", ACM Transactions on Database Systems, Vol. 5 No. 4, (December 1980)

[McKeown83] McKoewn, D. M. Jr., MAPS: The Organization of a Spatial Database using Imagery, Terrain and Map Data. Technical Report CMU-CS-83-136, Carnegie-Mellon University (July 1983)

[Interrante87] Interrante M., A Real-Time Spatial Index for a 3-dimensional Airspace Database. Proceedings from the National Aerospace and Electronics Conference. p. 1331 (1987)

[Orenstein86] Orenstein, Jack A., "Spatial Query Processing in an Object Oriented Database System", ACM SIGMOD, pp. 326-333 (1986)

[Reuss77] Reuss J. L., Chang, McCormick, "Paging Techniques for a Pictorial Database", Technical Report. Knowledge Systems Laboratory, University of Illinois at Chicago Circle, (June 1977)

[Scott87] Scott S., A Generalized Airspace Expert System Proceedings from the National Aerospace and Aeronautics Conference. p. 1315 (1987)

[Thatte86] Thatte S., Persistent Memory: A Storage Architecture for Object Oriented Database Systems. Proceedings from the International Workshop on Object-Oriented Database Systems. p.148 (1986)

TASK-LEVEL ROBOT PROGRAMMING: INTEGRAL PART OF EVOLUTION FROM TELEOPERATION TO AUTONOMY

James C. Reynolds
MITRE Corporation
1120 NASA Rd. 1
Houston, Texas 77058

ABSTRACT

In 1984 Congress recognized the merit of using the Space Station as a stimulus to develop a new generation of automation and robotics technology that would be efficient and flexible enough not only to meet the needs of the Space Station but also to benefit the U.S. economy. Task-level robot programming should be part of this new generation. Although it is a feasible technology for the mid-90's, it is not as well known within the NASA research and development community as it should be. This paper explains what task-level robot programming is and how it differs from the usual interpretation of "task planning" for robotics. Most importantly, it is argued that the physical and mathematical basis of task-level robot programming provides inherently greater reliability than efforts to apply better known concepts from Artificial Intelligence (AI) to autonomous robotics. Finally, an architecture is presented that allows the integration of task-level robot programming within an evolutionary, redundant, and multi-modal framework that spans teleoperation to autonomy.

INTRODUCTION

In 1984 Congress recognized the merit of using the Space Station as a stimulus to develop a new generation of automation and robotics technology that would be efficient and flexible enough not only to meet the needs of the Space Station but also to benefit the U.S. economy (NASA, 1985). The Congressional desire for technology transfer was at least partially motivated by the need to boost American labor productivity so that U.S. manufacturing would be more competitive with manufacturing in other nations.

A task-level robot programming system, which could be used with any robotic manipulator system on the Space Station, is an outstanding candidate for this new generation of technology. Such a system requires a complete world model of the workspace and task-level commands that consist of the identification of relevant objects and their desired relationships. An example would be, MOVE OBJECT ORU24 AGAINST FACE3 OF TRUSS66. These commands would then automatically be translated to the low-level motion and sensing operations required to reliably and safely achieve them. Such a system would prevent the tedious and time-consuming coding that flexible robot control normally demands.

Ten years of research in this area at Stanford, MIT, Carnegie-Mellon, IBM, and other robotic research centers has placed the development of a practical task-level robot programming system on the technological agenda. A unified conceptual framework has been developed and applied to the component problems of motion planning with obstacle avoidance, grasp planning for reachability and stability, and fine motion planning using compliance. Recently, an integrated system that implements much of the results of this work has been built at MIT. For reasons discussed below, NASA could be the decisive force in pushing this research out of the laboratory for the benefit of the Space Station Program and U.S. industry.

Research and development of autonomous robotics should build on the successes of this work. Unfortunately, alternate techniques borrowed from AI have often been applied to the problem of generating robot plans. Some of these techniques are knowledge-based and heuristic and are therefore inappropriate for robotics, especially in applications where reliability and safety are paramount. In addition, domain-independent planning is often applied to robotics, but this no longer can be viewed as a viable approach. Theorems from a recent

important thesis suggest that efficient domain-independent planning with expressive power for real-world robotics is impossible.

A task-level robot programming system would support the evolutionary approach to autonomous robotics that must be taken on the Space Station. Components of a system could be used as a plan checker for robot-level programs written by ground-based personnel. As confidence in the system increased, it could be used as a plan generator of robot-level instructions that would then be simulated and modified by ground-based programmers with the advice of Station-based astronauts. Finally, when the system is judged mature, its output of instructions could be fed directly into a robot controller interactively monitored by Station personnel. This paper presents an architecture for enabling the graceful phasing in of this technology to the Space Station Program.

DESCRIPTION OF TASK-LEVEL ROBOT PROGRAMMING

The best description of a task-level robot programming system together with a discussion of alternate implementation issues is in (Lozano-Perez and Brooks, 1985). That paper describes a framework called TWAIN for the development of such systems. The input to TWAIN would be a complete model of the robot and its environment together with a complete specification of the tasks to be accomplished.

For a practical system the model would not only include a geometrical description of objects, but also any other features of the environment that impose constraints on the motion of the robot. Mass, inertia, the coefficient of friction, restrictions on movement caused by linkages of objects, including the manipulator, and, most importantly, tolerances on the objects and bounds on capabilities such as accuracy, range, and force of both the sensors and manipulator would all be part of a task-level robot programming system's world model. Without the latter information it is impossible to plan motions in the face of the uncertainty that is the key problem of flexible robotics. The recent heightened awareness of the need for design knowledge capture during the development of the Space Station has made it possible that a model like this could be built.

The command input to the system would consist of robot-independent operations specifying the desired spatial relationships of relevant objects. A simple block-stacking example could be commanded as follows:

```
PLACE OBJECT-A AGAINST TABLE
PLACE OBJECT-B SO THAT
    FACE-1 OF B IS AGAINST FACE-2
    OF OBJECT-A
    AND
    FACE-2 OF B IS COPLANAR WITH FACE-3
    OF OBJECT-A
    AND
    FACE-3 OF B IS COPLANAR WITH FACE-1
    OF OBJECT-A
```

The importance of the commands being robot-independent is that the user does not have to specify grasp positions, complicated obstacle-avoiding paths, or terminating conditions based on dynamic and geometrical constraints. This enormously simplifies the practice of robot programming.

The commands to the task planning system advocated here are intermediate between low-level controller instructions and the input to traditional AI planners. It is important to recognize that this is not what is usually meant by "task planning" for robots. Since most task planners for robotics are based on the long chain of AI planners going back to STRIPS and ABSTRIPS, which were used to control the famous SRI robot, SHAKEY, their use of the word "task" is for a higher level of abstraction. A space-oriented example would be REMOVE ORU-24 FROM CHAMBER-3. Within the TWAIN framework this "task" might require several commands to specify.

The focus of much research and development for autonomous robotics has been on this higher abstraction task planning. The problem of focusing on this level rather than the intermediate level that TWAIN addresses is that the really hard problems of robotics are avoided. These higher level planners are capable of generating sequences of actions, but are not capable of planning under uncertainty or where there are side effects of the consequences of the planned actions. This will be discussed in greater detail in a later section of this paper.

THE IMPORTANCE OF TASK-LEVEL ROBOT PROGRAMMING

The development of a task-level robot programming system, which could be used with any robotic manipulator system on the Space Station, e.g., the Flight Telerobotic Servicer (FTS), would increase the productivity of the crew and ground personnel for Space Station operations and be a critical technology to transfer to U.S. industry.

That this capability would be crucial for the efficiency and usefulness of the Space Station itself is indicated by an analysis of the baseline configuration of the Space Station in terms of the crew hours available for maintenance and housekeeping (Reynolds, 1986). It is now widely recognized that there is, in fact, a contradiction between the hours estimated for those activities and the hours required for customer services. The logical resolution of this contradiction is to use A&R to increase crew productivity. A FTS that requires one or more crew members to continuously control it through teleoperation would possibly increase functionality of the Space Station and reduce extra-vehicular activity. It is not likely, however, to increase crew productivity enough to eliminate the contradiction between maintenance requirements and customer needs. The FTS must be programmable, and a task-level programming system would be the most productive way of programming the FTS.

Evidence that the development of a task-level programming system would indeed promise major advantages for industry can be found in the 1982 study by the Society of Manufacturing Engineers and the University of Michigan (Smith, 1982). That report recognized that robot control was the most important technological factor needed for the rapid utilization of robotics by U.S. industry, ranking ahead of computer vision, tactile sensing, and mechanical manipulation. The report was, however, overly optimistic about achieving this technology; a practical implementation of a task-level robot programming system by NASA for the Space Station could make the predictions of that report a reality.

Presently, robots in industry are either programmed by guiding or programmed in a robot-level language which includes instructions for accessing sensors and controlling the motions of the robot. Each of these methods has key shortcomings in the context of either industrial or space applications.

Programming by guiding involves the operator of the robot moving it through a sequence of positions needed to accomplish a specific task. The motions are recorded on tape and then are played back to execute this sequence repeatedly. This is one step above hardwired automation in that the robot can be used for more than one sequence of positions. However, it is equivalent to straight line programming in that no branching is allowed. For the robot to accomplish a task with this type of programming, the task must be characterized by little uncertainty in the geometry of the environment. It is impossible, with this type of programming, to use sensor feedback to correct positional errors or to choose alternate paths in the face of unexpected conditions.

Incorporating programming by guiding capability in the Space Station Program would accomplish little in benefiting U.S. industry or improving crew productivity on the Space Station. It is a mature technology which would not be advanced by its inclusion in Space Station A&R. Further, it is most appropriate for highly repetitive tasks, where the capability of deviating from a specified path is not of great importance. Even for maintenance and assembly tasks that require no great intricacy, there must be fine control with sensory feedback of any manipulator external to the Space Station, because of the sensitivity of the Station to the inertial effects of manipulator motion.

In addition, it is unlikely that many of the maintenance tasks will be done over and over for long periods of time (Holt, 1986). Typically, there will be several different maintenance tasks to be accomplished over a short time period like one day, and then it may be a much longer period before any of those tasks are done again. This situation is similar to what exists in small batch manufacturing, where robotics has not yet been applied because of the limitations of both programming by guiding and robot-level programming.

Robot-level programming is a commercially available alternative to programming by guiding. It represents an advance in that branching on sensor input is allowed, thus allowing for more robust behavior in the face of uncertainty and error. VAL-II for the Unimation series of robots and AML for certain IBM robots are two of the best known examples of these languages. These languages, although commercially available, need improvement. Some of the problems of these languages were outlined in (Lozano-Perez, 1983b). These include the following:

1) Robots and CAD/CAM systems are not able to communicate. Information in CAD systems should speed the computations necessary for motion.

2) Robot programming is highly device dependent, describing operations in terms of the motion of individual arms rather than tasks. The addition of new objects in the environment, including new robotic devices such as additional manipulators, generally requires the rewriting of the entire program.

3) Obstacle avoidance is difficult to specify, resulting in long complex programs consisting of moving and then checking. The programmer must anticipate all situations.

The key shortcoming of robot programming today is that, like any other type of software development, it is expensive and time-consuming. In industry very few of the explicit robot programming languages that are sold with robotic systems are ever used; it is easier to use guiding for programming (Rossol, 1984). For operations that require sensor data for reliability, the necessary programs are extremely complex. (Carlisle, 1985) mentions a VAL-II program in an assembly plant that was over one hundred pages long.

If the functions of the robot consist of manufacturing or assembling a few parts many times, the expense of programming can be justified. The problem is that a large part of industry manufactures and assembles many different parts in relatively small quantities. This has prevented the introduction of robots and advanced automation into many industries. Analogously, if the tasks to be performed on the Space Station are indeed diverse and non-repetitive, then the time required to program the robotic device might be greater than the time saved by the astronaut not being a slave to the robotic device.

Task-level programming is essentially a means to speed up the software development process that is ever more often the bottleneck in the engineering of large-scale systems. It is necessary for the next leap in the use of robots in industry. Without this capability the application of autonomous robotics to Space Station operations will be impossible. Further, research in the AI and robotics laboratories has reached a maturity that demands a new phase of actual development of these systems. Unfortunately, it is also clear that the impetus to utilize the fruits of over ten years of research will not come from industry. Thus, NASA can play a crucial role, which is exactly what Congress intended by its A&R mandate.

FEASIBILITY OF TASK-LEVEL ROBOT PROGRAMMING

The feasibility of a task-level robot programming system for the Space Station Program depends on over ten years of increasingly fruitful research.

The first big advance in the synthesis of robot programs from task-level specifications was the system described in (Taylor, 1976). It introduced the use of parameterized procedure skeletons, which were generalized robot programs including motions, error tests, and necessary computations but without the parameters bound to any numeric values. Depending on the geometry of the model and the tolerances and uncertainty bounds, a skeleton was chosen, and the remaining parameters were determined for grasp and approach positions. Taylor utilized linear programming methods to compute legal ranges for the parameters.

This work was significantly extended in (Brooks, 1982). Rather than using numerical methods to propagate constraints caused by position uncertainty, control uncertainty, and model uncertainty (tolerances), he used formal logic techniques to reason both forward to check error bounds and backward to restrict the range of plan variables and introduce sensing operations.

Fundamental to the line of research described here is the concept of configuration space (Lozano-Perez, 1983a). The configuration of an object is the set of parameters necessary to completely specify the position of all points of the object. The configuration space of an object is the space of all possible configurations of the object. Obstacle avoidance can be accomplished by transforming a robot into a point and, for each element of a discretized set of possible orientations of the robot, growing the obstacles by the shape and size of the robot wherever contact with the obstacles is feasible.

Rapid progress in the development of algorithms for gross motion planning with obstacle avoidance and fine motion strategies was made during the next few years based on these concepts. (Brooks, 1984) is a good example of gross motion planning work and provides further references. An algorithmic approach to the automatic synthesis of fine motion strategies (guarded moves with compliance) was described in (Lozano-Perez, Mason, and Taylor, 1984); since then this line of research has matured further (Erdmann, 1986).

Automatically synthesized fine motion can be achieved by extending the configuration space concept to the notion of recursively determined pre-images. These are the set of all starting configurations which can reach a goal configuration within the constraints of control and model uncertainty but allowing for compliant motion. The pre-image must also exclude configurations and velocities which would lead to sticking. If the set does not include

the actual starting configuration then the algorithm is applied recursively to determine the pre-image of a configuration within the first pre-image. Thus multi-step plans can be generated.

TWAIN was proposed in (Lozano-Perez and Brooks, 1985) and embodies all these ideas. Each single task specification is turned by the executive planning module of the system into a sequence of gross motion, grasping, gross motion, fine motion (either synthesized or pre-determined), and ungrasping. There are separate modules for each of these three types of planning. There is a skeleton library and skeleton matcher to choose unrefined plans, and there is a constraint propagator which uses the principle of least commitment and symbolic propagation to instantiate skeleton parameters and add sensing operations. Finally, because success is not guaranteed, dependency-directed backtracking is employed to reduce search.

TWAIN has not been implemented. However, since its proposal, component problems for a task-level programming system based on the work of Brooks, Lozano-Perez, Mason, and Taylor have been tackled with increasing success. There has been an explosion of work on motion planning with obstacle avoidance. The automatic generation of grasping (Nguyen, 1987) and regrasping (Tournassoud et al., 1987) strategies has also been tackled. Finally, a systematic attack on the problem of error detection and recovery (as opposed to the ad hoc "hacks" of traditional AI planners) based on the algorithms and concepts developed in the above referenced work was informally presented in (Donald, 1986).

Some of these more recent ideas and some of the ideas of the original TWAIN proposal have been implemented in an integrated robot system by Lozano-Perez and his colleagues at MIT (Lozano-Perez et al., 1987). The Handey system is capable of locating a part that has been accurately modeled in an unstructured environment, choosing a grasp on the object, planning a collision free path to the object, grasping the object, planning a collision free path to the specified destination, and placing the object in the commanded position. The system does not incorporate the ideas of constraint propagation, but it is under development with an error detection and recovery capability being one of the planned additions.

This large and growing body of work tackling the hard problems of robotics utilizes sophisticated mathematics and is thoroughly grounded in geometry and mechanics. Its algorithms can be analyzed for correctness and completeness, and hence their robustness can be verified. It does not use expert systems or the simplistic techniques of traditional AI planners. It is not surprising that this approach has led to an actual robotic system that is capable of more impressive "intelligence" than any system depending on those techniques. It is this work that NASA should be pushing and extending.

AI AND AUTONOMOUS ROBOTICS

During the last few years many of the concepts and techniques of AI have become commonplace in engineering and data processing publications. It is of interest then to discuss the relevance of AI to robotics. The first concept and associated techniques from AI to be widely applied was "expert systems." This phrase has now given way to "knowledge-based systems," perhaps in recognition that many useful applications could be built without needing expert competence. In either case these systems contain a knowledge base of facts and rules and an inference engine to reason from the knowledge base and the data input to the system.

For robots to be autonomous it is essential that the algorithms controlling their actions are correct (the robot does what is intended by the human) and reasonably complete (if it is possible to accomplish a specified task, a robot-level program will be generated to do it). This is extremely important for robotic applications that must be robust in the face of uncertainty. In a hazardous environment like space the need for correctness and completeness is even greater. The use of expert systems, which are inherently heuristic, is an ill-advised application of a useful technology. It is simply not good enough for a motion planner to plan an obstacle avoiding path ninety per cent of the time.

Further, even without performance and reliability considerations for robotic applications, the use of expert systems in robotics is inappropriate. This is because there are few heuristic rules that can be generalized to guide motion planning; instead, it appears that small changes in the environment lead to significantly different motion strategies. For example, in the simple peg-in-hole task the geometrically trivial change of adding a chamfer to the hole would result in a radically different motion strategy.

Domain-independent planning is an area of AI that seems to be naturally applicable to robotics. Indeed, the famous early planners like STRIPS and HACKER were often applied to planning the actions of a "robot" in a "blocks world." ABSTRIPS (Sacerdoti, 1974) actually controlled a physical robot, SHAKEY. Unfortunately, an investigation of the SHAKEY project shows that the environment was too carefully engineered to be realistic and that errors in the model were handled by expensive re-planning. In addition, all of the well known planners utilized impoverished semantics; their worlds could be described in a few sentences. Representing the complex, largely quantitative model required for task-level programming would be impossible using these planners.

There are two reasons that traditional AI planners are so inappropriate for robotics. First, robot planning requires geometric representations and those are largely numeric, which has not been the emphasis of AI in general, including its planners. Second, robot planning must handle uncertainty and error, and AI in general has not been able to solve this problem. AI planners do not even pretend to try; if they attempt to handle it at all, it is by ignoring uncertainty while planning and then trying to recover during execution.

Even more devastating to those who wish to apply the techniques of AI planning to robotics is an important thesis (Chapman, 1985). That work shows that all well known AI planners work in essentially the same way. Further, their action representations do not allow for indirect or input dependent effects or for uncertain execution. Finally, extending them with more expressive action representations while keeping them computationally tractable is probably impossible and would also invalidate the proofs of correctness and completeness of these planners within their limited, artificial world.

AN EVOLUTIONARY IMPLEMENTATION OF TASK-LEVEL ROBOT PROGRAMMING FOR THE SPACE STATION

It is important that the development of autonomous robotics for space applications proceed in an evolutionary manner. In addition, any robotic system on the Space Station must be capable of allowing human control at any point during its operation. A task-level programming system for the Space Station could be implemented within these requirements. (See Figure 1.)

The first step would require the construction of a teleoperated control system augmented with the advanced ideas of computer-assisted human interaction described in (Conway et al., 1987). On top of that the components of TWAIN that are necessary for plan checking would be built. This would allow the astronaut to input robot-level programs written by ground personnel and determine their correctness. If no bugs were detected, there would be feedback provided by simulation before allowing execution. If bugs were detected, the plan checker would act as a smart compiler in suggesting corrections. Finally, as confidence in the technology reached an acceptable level, the full-scale planner could be built which would allow the astronaut to input task-level plans, check their simulated effects, and monitor execution as closely as desired.

CONCLUSION

Task-level robot programming is an important technology that promises great benefits both on the ground and in space. Considerable progress has been made toward realizing a system that could automate the flexible control of robots. This progress has been characterized by a solid grounding in geometry and mechanics, which makes it verifiably robust and a natural choice for space applications. It is feasible to implement this technology in an evolutionary way for the Space Station Program. NASA should take steps to build on the successes of the research described in this paper in order to develop autonomous robotics.

REFERENCES

Brooks, Rodney, "Symbolic Error Analysis and Robot Planning," INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, Cambridge, Ma., Vol. 1, No. 4, December, 1982, pp. 29-68.

Brooks, Rodney, "Planning Collision Free Motions for Pick and Place Operations," ROBOTICS RESEARCH: THE FIRST INTERNATIONAL SYMPOSIUM, M. Brady and R. Paul, Eds., MIT Press, Cambridge, Ma., 1984, pp. 5-37.

Carlisle, Brian, "Key Issues of Robotics Research," ROBOTIC RESEARCH: THE SECOND INTERNATIONAL SYMPOSIUM, H. Hanafusa and H. Inoue, Eds., MIT Press, Cambridge, Ma., 1985, pp. 501-503.

Chapman, David, "Planning for Conjunctive Goals," Technical Report 802, MIT Artificial Intelligence Laboratory, Cambridge, Ma., November, 1985.

Conway, Lynn, Volz, Richard, and Walker, Michael, "New Concepts in Tele-Autonomous Systems," SECOND AIAA/NASA/USAF SYMPOSIUM ON AUTOMATION, ROBOTICS AND ADVANCED COMPUTING FOR THE NATIONAL SPACE PROGRAM, Paper IAA-87-1686, Arlington, Va., March 9-11, 1987.

Donald, Bruce, "Robot Motion Planning with Uncertainty in the Geometric Models of the Robot and Environment: A Formal Framework for Error Detection and Recovery," 1986 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, San Francisco, Ca., April 7-10, 1986.

Erdmann, Michael, "Using Backprojections for Fine Motion Planning with Uncertainty," THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, Cambridge, Ma., Vol. 5, No. 1, Spring 1986, pp. 19-45.

Holt, Alan, "Mobile Robotics and Artificial Intelligence for Large Payload Assembly and Maintenance," Paper 6-0607, ROBEX '86, Houston, Tx., June, 1986.

Lozano-Perez, Tomas, "Spatial Planning: A Configuration Space Approach," IEEE TRANSACTIONS ON COMPUTERS, Vol. C-32, No. 2, February, 1983a, pp. 108-120.

Lozano-Perez, Tomas, "Robot Programming," Proceedings of the IEEE, Vol. 71, No. 7, July, 1983b.

Lozano-Perez, Tomas, Mason, Matthew, and Taylor, Russell, "Automatic Synthesis of Fine-Motion Strategies for Robots," THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH, Cambridge, Ma., Vol. 3, No. 1, 1984.

Lozano-Perez, Tomas and Brooks, Rodney, "An Approach to Automatic Robot Programming," Artificial Intelligence Laboratory, Massachusetts Institute of Technology, A. I. Memo No. 842, April, 1985.

Lozano-Perez, Tomas, Jones, Joseph, Mazer, Emmanuel, O'Donnell, Patrick, and Grimson, Eric, "Handey: A Robot System that Recognizes, Plans, and Manipulates," 1987 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, Raleigh N.C., March 31 to April 3, 1987.

NASA, "Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy, Progress Report 1," NASA Technical Memorandum 87772, Houston, Tx., October, 1985.

Nguyen, Van-Duc, "Constructing Stable Grasps in 3D," 1987 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, Raleigh N.C., March 31 to April 3, 1987.

Reynolds, James, "Analysis of Crew Time on the Space Station," unpublished MITRE Corp. report to the Artificial Intelligence and Information Sciences Office at Johnson Space Center of NASA, Houston, Tx., March, 1986.

Rossol, Lothar, "Technological Barriers in Robotics: A Perspective from Industry," ROBOTIC RESEARCH: THE FIRST INTERNATIONAL SYMPOSIUM, M. Brady and R. Paul, Eds., MIT Press, Cambridge, Ma., 1984, pp. 963-972.

Sacerdoti, Earl, "Planning in a Hierarchy of Abstraction Spaces," ARTIFICIAL INTELLIGENCE, Vol. 5, No. 2, 1974, pp. 115-135.

Smith, Donald and Wilson, Richard, "Industrial Robots: A Delphi Forecast of Markets and Technology," Society of Manufacturing Engineers and The University of Michigan, 1982.

Taylor, Russell, "The Synthesis of Manipulator Control Programs from Task-level Specifications," AIM-282, Stanford Artificial Intelligence Laboratory, Palo Alto, Ca., July, 1976.

Tournassoud, Pierre, Lozano-Perez, Tomas, and Mazer, Emmanuel, "Regrasping," 1987 IEEE CONFERENCE ON ROBOTICS AND AUTOMATION, Raleigh N.C., March 31 to April 3, 1987.

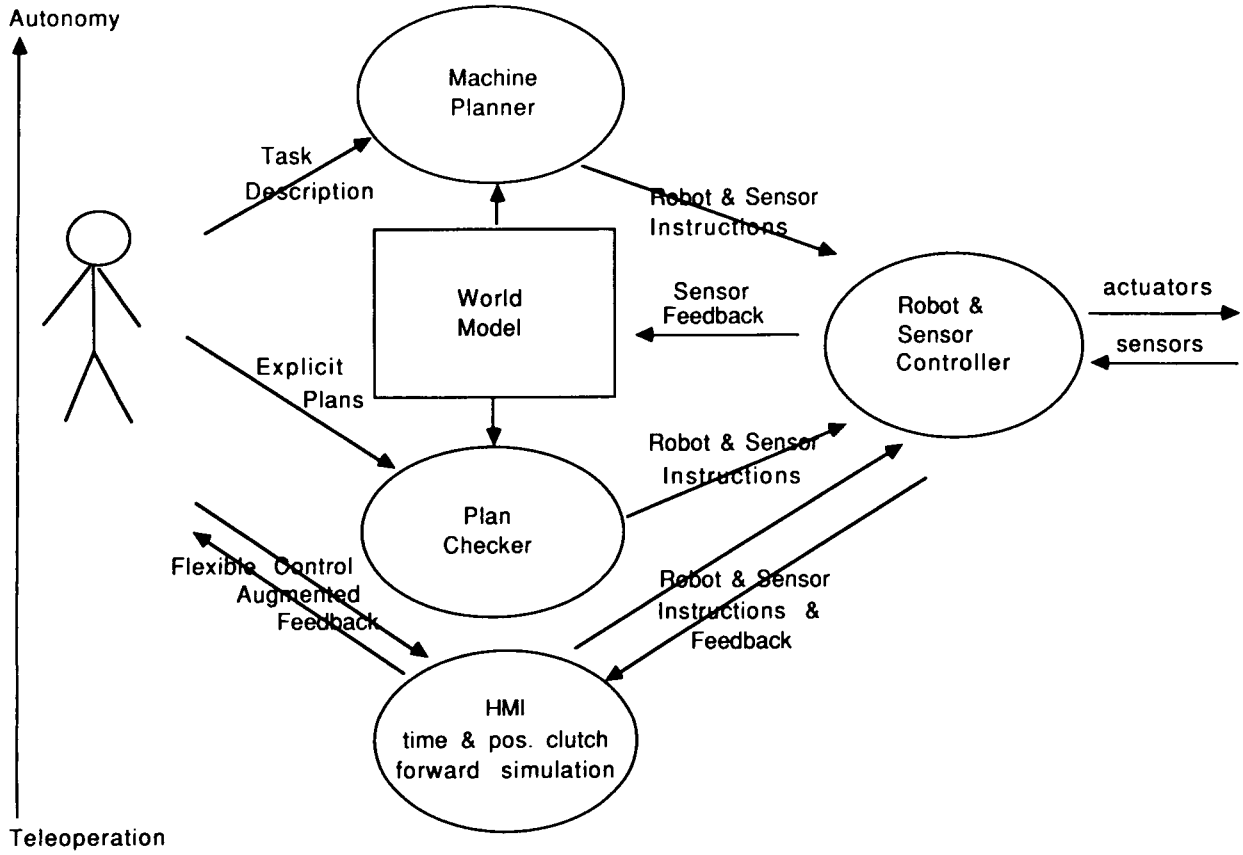


FIGURE 1 - EVOLUTIONARY ARCHITECTURE WITH TASK-LEVEL PROGRAMMING

HYPERVELOCITY TECHNOLOGY (HVT) CREW ESCAPE

Lanny A. Jines, P.E.
Aerospace Engineer
Air Force Wright Aeronautical Laboratories
Flight Dynamics Laboratory
Crew Escape and Subsystems Branch
Air Crew Escape Group
WPAFB, OH 45433

ABSTRACT The Flight Dynamics Laboratory is currently conducting a research and development effort investigating conceptual designs for escape systems applicable to hypervelocity technology class aerospace vehicles. The contractor, Boeing Military Airplane Company, has recently completed Task I, Concept Definitions and Preliminary Evaluation; and Task II, Enabling Technology Identification; of contract F33615-86-C-3410 (Reference 1). The concepts selected for further development through out the effort will provide survivable escape and recovery throughout all phases of flight including launch, upper atmospheric hypervelocity, orbit, atmospheric entry, terminal approach, and landing. The specific objective for Task I was to conduct conceptual development of the candidate escape system concepts which meet the various crew escape and protection requirements. The contractor initially identified sixteen (16) conceptual escape systems. Of the sixteen, there were two viable options. The study vehicles included a horizontally launched vehicle (HLV) and a vertically launched vehicle (VLV). The contractor has developed graphic computer aided design models of the candidate escape systems with Zenith 248 computers utilizing the CADC IIc software package (Reference 2). During Task II the contractor has identified the necessary state-of-the-art or near-term enabling technologies; i.e., propulsion, life support, thermal protection, deceleration, etc.; that would allow for the implementation of the conceptual designs. The contractor in Task III, Trade Studies, shall prepare performance simulation models of the conceptual designs using the EASY5/EASIEST Computer Program (Reference 3) software with the escape system component and analysis input files appropriately modified for configurations of interest to conduct an in-depth trade study of the candidate concepts.

INTRODUCTION The aerospace vehicles of the future will incorporate hypervelocity technologies, providing the capability of flying at much higher altitudes and much faster speeds than the current military aircraft. These vehicles will have the capability to be in orbit from one to three revolutions around the earth. Appropriately, the escape systems for such vehicles will require an expanded flight envelope when compared to the existing escape system performance envelopes of current military aircraft. Presently, open ejection seats provide inadequate performance for hypervelocity class vehicles. The ejection trajectory range is cannot provide for safe escape from the launch pad or for the initial phase of ascent. State-of-the-art open ejection seats are also inadequate for high speed or high altitude escape conditions. During a seven (7) year period from 1973 to 1979, the statistics from non-combat ejections of open ejection seats at airspeeds between 400 and 500 keas showed that 57% of the crew members sustained major or fatal injuries. From 500 to 600 KEAS, the major injury and fatality rate was approximately 70% and above 600 KEAS, the probability of major or fatal injury was 100% (Reference 4,p.27). Pressurization is required for protection when ejection occurs above 50,000 feet altitude. Attempts to provide emergency escape capability for high velocity atmospheric aircraft has led to the development of enclosed ejection seat escape systems (B-58) and B-70) and crew escape modules (F-111 and prototype B1). The problems posed by these types of escape systems have been: accelerations imposed on the crew during separation from the aircraft and upon landing impact, increased time to full recovery parachute inflation due to larger recovery parachute systems, weight penalty, and high life cycle costs. Various concepts and techniques for providing escape capability for the crew of space vehicles have been studied in significant detail since before the first United States (U.S.) Manned Space Program, Project Mercury. The reason for the numerous space escape study efforts in the 1960's and 1970's are obvious; practically all aspects of manned space flight were unknown. The United States was "in a hurry" to establish space superiority. And, of course, all space flights were done in view of the entire world. The greatest concern for crew safety in the early space projects was the on-the-pad or launch phase of the mission. The Mercury and Apollo escape systems were for the on-the-pad and early boost phases only (the rocket powered escape towers were jettisoned shortly after launch). Gemini employed ejection seats for the crew, therefore it had a post

* C-7

atmospheric entry escape capability which Mercury and Apollo did not have. Sky Lab astronauts utilized the Gemini-B during launch and atmospheric entry thus they had the same escape capability as the Gemini system. Sky Lab astronauts also had an escape capability in space. They could enter the Gemini-B that was docked to the Lab, separate from the Lab, and subsequently return to earth in the capsule. The Space Shuttle used ejection seats for atmospheric escape capability in early flights, however NASA deactivated the seats when the crew manifest was expanded beyond two individuals. Of the many space escape studies performed in the past, the separable atmospheric entry escape capsule/module seems to have been the most prevalent. The goal in most escape studies was to provide a single escape concept/technique which would provide the crew with escape capability at any phase of the mission. Another escape concept which has received much attention is a non atmospheric entry separable capsule or module, which would separate from a disabled orbiting space vehicle and remain in orbit until recovered in space by another space vehicle. During mission phases other than the orbit phase of such a vehicle the escape system designer relied upon techniques as used on Mercury, Gemini, and others.

DISCUSSION The statement of work (SOW) requires the contractor to postulate single and dual place escape system concepts for contractor defined HVT aerospace vehicles. The selected vehicles are to be representative of the class designed for transatmospheric capabilities which include missions of one to three orbits plus upper atmospheric brakemaneuvering for at least one orbital plane change. The selected hypervelocity vehicles for which the escape systems are to be conceptualized include one that is vertically launched and one that is horizontally launched. Figure 1 shows a range of applicable similarity parameters for the atmospheric entry of the selected vehicles.

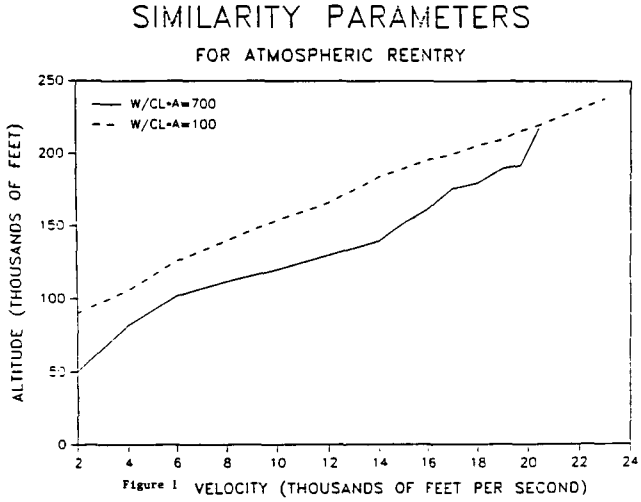


Figure 1

The corridor between W/CL*A = 100 and W/CL*A = 700 (W = Weight, CL = Coefficient of Lift, and A = Reference Area) is representative of the range of flight parameters for HVT aerospace vehicles. The lower value corresponds to a vehicle typical of the NASA Space Shuttle design yielding an entry trajectory that has a higher angle of attack, higher altitude approach, minimum heating, and minimum aerodynamic loading. The higher value represents a vehicle with a maximum Lift-to-Drag ratio (L/D) providing an entry path yielding greater range or crossrange flight capability which is more characteristic of desired military performance in the HVT class vehicles. The vehicles allow for a payload approximately equal to 1% of the total takeoff weight which is estimated to be 1.3 to 1.6 million pounds. The Air Force SOW Task I requires the contractor to postulate escape system concepts to provide for survivable escape and recovery throughout the phases of flight allowed by the selected VLV or HLV performance envelopes; i.e. 1) launch, 2) upper atmospheric hypervelocity flight, 3) orbit, 4) atmospheric entry, 5) terminal approach, and 6) landing. Initially the contractor is to develop basic escape system concepts which provide for crew escape from initial conditions within the selected vehicle's flight performance envelope that result in final crew landing within the continental United States (CONUS) from orbital flight, or anywhere on earth for all other flight conditions. Subsequently, the contractor shall separately consider advanced escape system concepts for each of the selected vehicles. These advanced escape system concepts shall possess sufficient performance capabilities to: 1) allow for recovery within the CONUS for escape initiated from orbit, 2) allow for extended cross range flight for escape initiated during upper atmospheric hypervelocity flight, and 3) allow for immediate recovery anywhere on earth for all other escape conditions. Within these requirements the desired goal of achieving escape system concepts exhibiting minimum weight and minimum volume shall be sought. During Task II the contractor is required to investigate promising technologies in the fields of aerodynamics, thermodynamic protection, propulsion, materials, structures, flight controls, life support and human protection that are necessary to implement the various concepts with maximum escape performance and minimal weight penalty to the overall vehicle performance. The identification of alternative technologies for implementing each fundamental functional requirement as well as the preliminary sizing designs of each alternative technology is required. SOW Task III involves a comparative trade study of the concepts defined in Task I and their associated technologies investigated in Task II to select the best alternative technology to implement each fundamental functional requirement identified in Task I. Volume, cost, weight, risk, compatibility with the gross concept and development requirements are to be used as trade criteria with suitable merit weights selected by the contractor. The contractor shall evaluate performance of the various proposed escape systems throughout the vehicles' operational envelopes with attention to minimal impact to the overall added weight of the vehicle; crew station integration; crew mobility; vision; comfort; ingress and egress in normal and

emergency situations; and potential R&D problems. The contractor is to develop FORTRAN IV Extended computational component models of the selected escape concepts compatible with the EASY5 Computer Program. These models are used to compute vehicle accelerations, angular rates, trajectories, and thermal loads for the purpose of evaluating the selected escape concepts in terms of state-of-the-art human protection design criteria with emphasis given to short term (less than one second) and long term acceleration, vibration, thermal energy, and atmospheric pressure. The short term acceleration exposure limits have been specifically developed by the Harry G. Armstrong Aerospace Medical Research Laboratory (Reference 1, Appendix A).

The contractor has selected the designs shown in Figure 2 and Figure 3 for the the HLV and VLV, respectively.

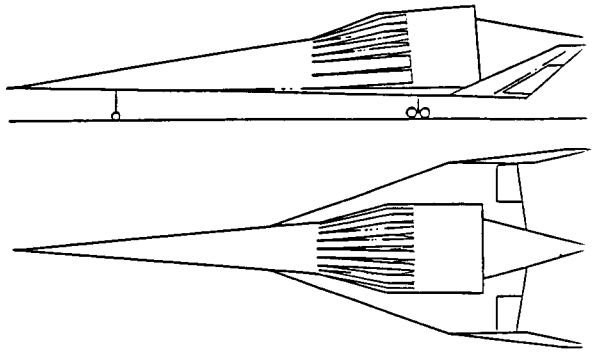


Figure 2 Selected Horizontally-Launched HVT Vehicle Configuration

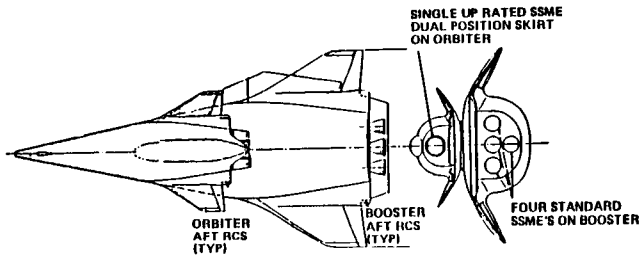


Figure 3 Selected Vertically-Launched HVT Vehicle Configuration

The HLV is a single-stage-to-orbit vehicle which makes extensive use of combined cycle airbreathing propulsion. Its takeoff launch weight is approximately 1.6 million pounds and the design has provisions for a crew of two. The VLV, which is approximately a 1.3 million pound launch weight design, is a two-stage launch vehicle consisting of a single crewmember orbiter and an unmanned booster. For both vehicle configurations, active cooling of critical areas and compartments is required during flight at high Mach number or during atmospheric entry. During launch, the HLV cruise climbs a dynamic pressure launch profile of 1200 lbs/sq.ft. until Mach 12 is reached. At this condition the flight path steepens to gain altitude. Airbreathing propulsion ceases at 200,000 feet (ft) altitude and Mach 25. A transition is made to rocket propulsion to achieve a higher orbital altitude of 100 to 300 nautical miles. For atmospheric hypersonic flight the vehicle will operate between 125,000 and 180,000 feet altitude at Mach 20. The VLV experiences a traditional vertical launch followed by a slight pitchover, a gravity turn, and then a phase which uses pitch to maintain a flight-path angle of zero (0) degrees until the desired velocity is achieved. The maximum dynamic pressure during the ascent is 400 pounds per square foot (psf) which occurs at 40,000 ft and 90 seconds after liftoff. The vehicle reaches 80,000 ft at 125 seconds after liftoff and continues to 300,000 ft in an additional 150 seconds.

The crew escape and protection requirements as specified in the SOW are the applicable military specifications MIL-S-9479B (Reference 6), MIL-C-25969B (Reference 7), and the Air Force Systems Command Design Handbook 1-3, Human Factors Engineering (Reference 8). For brevity only the modifications to these requirements necessary for HVT escape systems will be discussed. The low altitude performance requirements for escape capsules in MIL-C-25969B are essentially the same as required for ejection seats in MIL-C-9479B. Applied to HVT vehicles, the following Table 1 has been proposed by the contractor as the low altitude requirements:

TABLE 1. Low level Escape Performance Requirements for HVT Escape

Cond. No.	Pitch Angle, deg	Roll Angle, deg	Flight Path Angle, deg	Velocity, knots	Altitude Required, feet
1	0	0	0	0	0
2 *	90	0	90	0	0
3	-10	180	-10	250	600

* Applicable to vertically-launched vehicle only. Not applicable to horizontally-launched vehicle.

The HVT escape system range requirements are as previously discussed. Standard explosive hazard design requirements in terms of safe

distances as a function of TNT equivalent explosives are utilized. It is noted that the main dangers due to explosion are: shockwave, peak and duration, thermal radiation, shrapnel, and fireball. The contractor has considered a complete array of crew protection requirements which must be satisfied by the designed escape systems to ensure no or minimal injuries to the crewmenbers, i.e. accelerations, angular rates, total pressure, oxygen partial pressure, carbon dioxide, environmental temperature, ionizing radiation, windblast, exposure to shock waves, flashblindness protection, space motion sickness, and waste management. The contractor has initially investigated 16 escape system concepts of various capabilities which exhibited possibilities to satisfy the crew escape and protection requirements for escape during part of the HVT vehicle flight envelopes. These concepts are:

1. Extraction system
2. Open ejection seat
3. Encapsulated seat with thermal protection
4. Separable nose capsule with thermal protection
5. Pod-type capsule with thermal protection
6. Inflatable capsule with reentry capability
7. Paracone with reentry capability
8. Mating with orbiting space rescue station
9. Rocket-pack escape to space rescue station
10. Rocket-pack escape to a reentry rescue capsule
11. Mating with rescue vehicle
12. Non-reentry capsule escape to rescue vehicle
13. Ejection seat with orbital rescue
14. Extraction system with orbital rescue
15. Ejection seat with inflatable re-entry capsule
16. Ejection seat with rocket-pack transfer to rescue capsule

The results of a trade study of the features of these concepts against the desired SOW requirements identified that only the concepts numbered 3 and 5 were determined to be feasible for all phases of flight.

The contractor has conducted detailed design of the candidate escape concepts including definition of the operational escape sequence. The advanced encapsulated seat designs for hypervelocity vehicles is shown in Figure 4, 5, and 6.

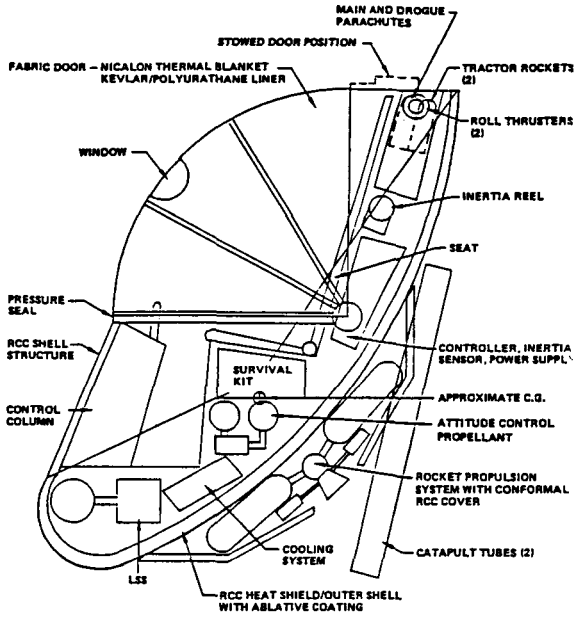


Figure 4 Encapsulated Seat Design for Hypervelocity Vehicles

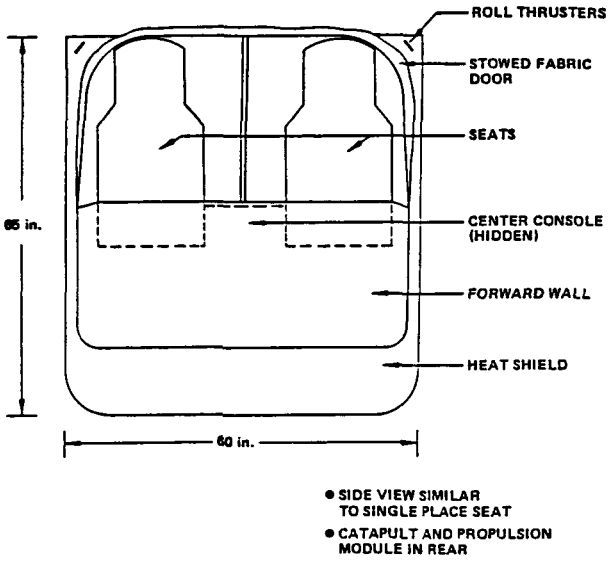


Figure 5 Front View of 2-Place Encapsulated Seat

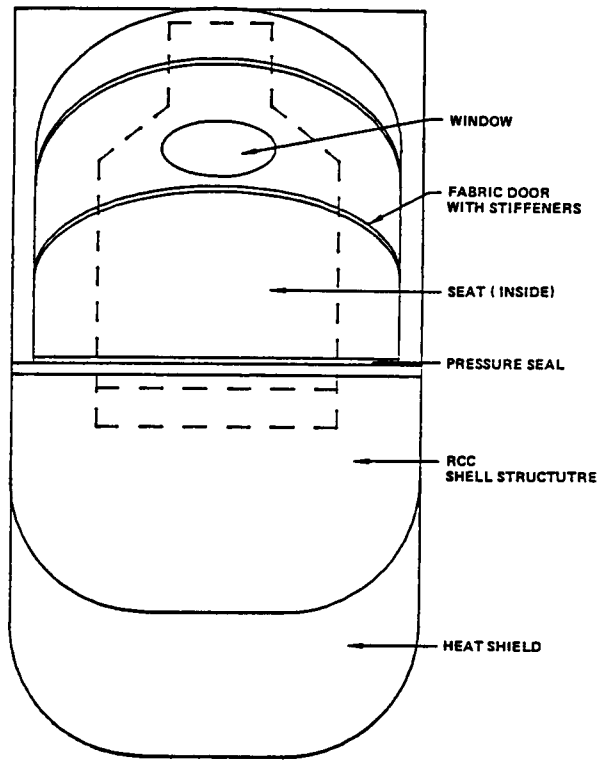


Figure 6 Front View of Single Place Encapsulated Seat

Figure 4 shows basically a modified B-58 ejection seat with doors to shield the crew member from the environment during escape and to provide emergency life support environment. It includes a heat shield, solid-propellant retrorocket engine, reaction control jets, life support system and a control system. A front view of a two place side by side version of the encapsulated seat for the HLV is shown in Figure 5 while a single place version for the VLV is shown in Figure 6. The emergency escape sequence and system operation for the encapsulated seat which follows after a crewmember pulls the ejection handle initiating the digital control sequencer is summarized below (Figure 7):

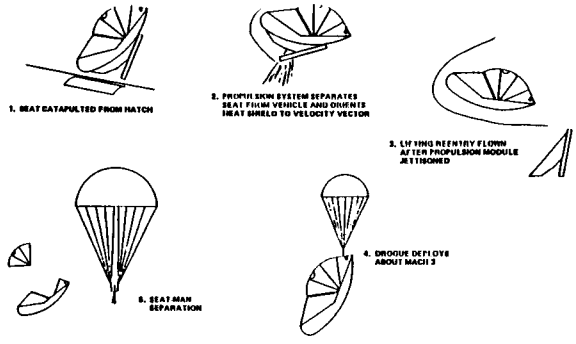


Figure 7 Encapsulated Seat Escape Sequence (Hypersonic/Reentry Flight Phase)

1. Escape condition evaluated based on information from the vehicle data bus and seat-mounted sensors, and life threat assessment conducted (start at 0.010 sec, complete at 0.020 sec after initiation).
2. Thermal batteries initiated (0.010 sec start, 0.050 sec complete).
3. Crewmember haulback devices initiated (0.030 sec start, 0.200 sec complete).
4. Limb capture devices initiated (0.030 sec start, 0.200 sec complete)
5. Close and lock seat door (0.200 sec start, 0.250 sec complete)
6. Initiate seat oxygen and pressurization system (0.200 sec).
7. Jettison ejection hatch (0.200 sec start, 0.300 sec complete)
8. Initiate catapult (0.300 sec)

The following events depend upon the initial flight condition occurring at the time of escape:

Atmospheric Flight below Mach 3 (includes zero altitude/zero airspeed)

9a. Propulsion system ignites after catapult stroke (0.5 seconds)
10a. Deceleration drogue parachute deployed if airspeed is between 300 and 500 KEAS.
11a. Main recovery parachute deployed if airspeed is below 300 KEAS and the altitude is below 15,000 feet altitude. Note: fabric door and drogue are jettisoned.
12a. Restraints severed and crewmember removed with survival kit.
13a. Crewmember makes conventional parachute landing.

Hypersonic flight (including atmospheric entry)

9b. Propulsion system ignites after catapult stroke (0.5 seconds) Seat is positioned with heat shield forward.
10b. Propulsion module is jettisoned (1.5 seconds)
11b. With inertial sensing unit and attitude control thrusters, the seat varies its lift vector orientation to control aerodynamic heating rate and to provide cross range capability (1.5 sec - 20 mins)
12b. After velocity decreases below Mach 3 sequence 10a to 13a occurs.

Orbital flight

9c. Following catapulting from vehicle the seat orbits until appropriate time to deorbit (0.5 sec - 12 hrs)
10c. Attitude thrusters orient seat for deorbit (10 secs)
11c. Propulsion deorbit burn (10 sec)
12c. The heat shield is positioned forward.
13c. Proceed with 10b to 12b.
The single place encapsulated seat, shown in Figure 6, for the VLV varies in the design details of the thickness of ablative coating and attitude control system capability due to the differences in maximum dynamic pressure (400 psf for VLV compared with 2000 psf for HLV) and aerodynamic drag area anticipated. The escape

sequencing and operation of the single place VLV encapsulated seat is the same as previously described for the dual place HLV system.

Figures 8 and 9 show Concept 5, the pod-type capsules with thermal protection.

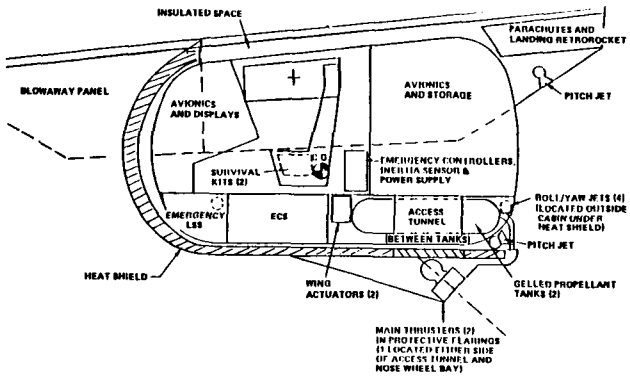


Figure 8. Pod-Type Capsule for Horizontally-Launched Hypervelocity Vehicle

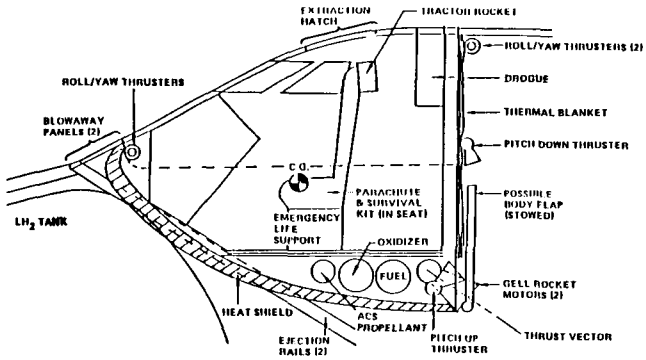


Figure 9. Pod-Type Capsule for Vertically-Launched Hypervelocity Vehicle

These capsules share basic structure with the crew cabin. The figures depict the detailed side views featuring the component subsystems.

The HLV pod capsule utilizes folding wings as shown in Figure 10.

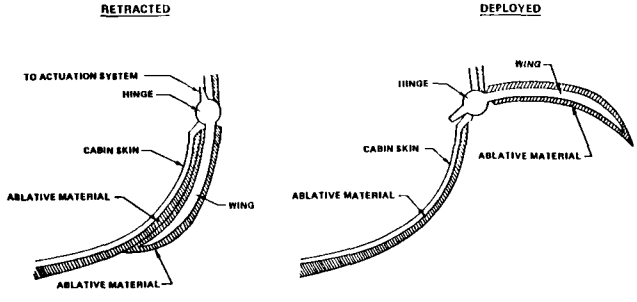


Figure 10. Pod Capsule Wings in Retracted and Deployed Position

These deploy to achieve lift to drag ratio of 2 to 4. This capability coupled with a typical roll maneuver yields a side force for cross range requirements. The escape sequence and operation following initiation is (Figures 11):

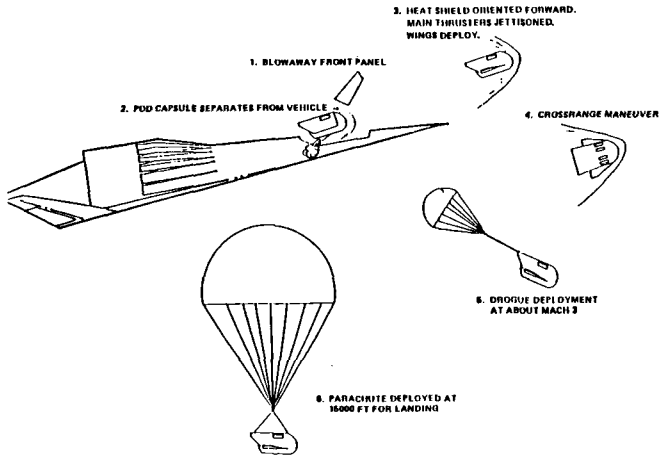


Figure 11 Escape Sequence for HLV Pod Capsule During Upper Atmospheric Escape

1. Initial condition evaluation (0.010 sec start, 0.020 sec complete).
2. Initiate thermal batteries (0.010 sec start, 0.050 sec complete).
3. Initiate crewmember haulback (0.030 sec start, 0.200 sec complete).
4. Initiate oxygen and pressurization (0.030 sec).
5. Severe capsule structure (0.050 sec).
6. Initiate propulsion system (0.2 sec start, 0.4 sec end)

The subsequent sequence depends upon escape initial conditions:

Atmospheric flight below Mach 3

7a. Propulsion system stabilizes flight, steers, and reduces deceleration (0.4 sec start, 1,2 sec end).

8a. Propulsion cut off and drogue deployed. When velocity and altitude are below 300 KEAS and 15,000 feet, or during low speed low altitude escapes the recovery parachute is deployed.

9a. Retrorockets attenuate ground impact.

Hypersonic flight (including atmospheric entry)

7b. Propulsion continues for thrust, stabilization,deceleration, and rolling (0.4 - 1.2 sec)

8b. Wings deployed and main nozzles jettisoned (0.4)

9b. Pod attitude control used to orient lift vector for the desired deceleration and cross range (up to 20 min).

Orbital Flight

7c. Pod remains in orbit until appropriate time to deorbit (0.5 sec. - 12 hrs)

8c. Thrusters orient pod for deorbit (10 sec).

9c. Deorbit maneuvers (2.0 sec)

10c. Thrusters reorient pod for heat shield positioning.

11c. Follow sequence 7b to 10b.

Figure 9 shows the pod-type capsule as designed for the VLV. The figure illustrates the location of various subsystems and components. This pod capsule design is considered a hybrid system in that it utilizes a rocket extraction system to remove the crewmembers for final recovery under personal parachutes. The escape sequence and operation following initiation is (Figure 12):

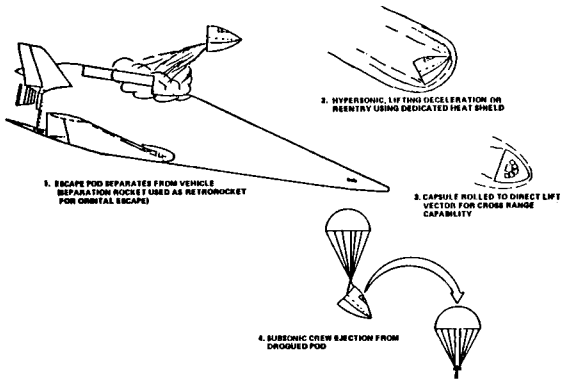


Figure 12 Recovery Sequence Diagram for Vertically-Launched Vehicle Capsule

1. Initial conditions evaluated (0.010 sec start, 0.020 sec complete).
2. Thermal batteries initiated (0.010 sec start, 0.050 sec complete).
3. Crewmember haulback (0.030 sec start, 0.200 sec complete)
4. Initate oxygen and pressurization (0.03 sec).
5. Severe capsule structure (0.050 sec)
6. Propulsion initiation (0.2 sec start, 0.4 sec complete)

The subsequent sequence depends upon escape initial conditions:

Atmospheric flight below Mach 3.

7a. Propulsion system continues (0.4 sec start, 1.2 sec complete).
8a. Drogue deployed except at low altitude the extraction of crew occurs immediately. Below 300 KEAS and 15,000 feet altitude the ejection hatch blows.
9a. Extraction tractor rocket for each crewmember fires.
10a. Crewmembers make conventional parachute landing.

Hypersonic flight including atmospheric entry.

7b. Propulsion system continues (0.4 sec start, 1.2 sec complete).
8b. Pod lift vector controlled for desired deceleration profile and cross range (up to 20 minutes).
9b. Below Mach 3 follow 8a through 10a.

Orbital flights

7c. Pod remains in orbit until appropriate time for deorbit maneuver (0.5 sec - 12 hours).
8c. Thrusters orient pod for deorbit (10 sec).
9c. Deorbit burn (2.0 sec)
10c. Thrusters reorient pod for forward-facing heat shield position.
11c. Follow sequence 7b through 9b.

During Task II, the contractor investigated emerging technologies in the structures, materials, thermal protection, propulsion, aerodynamics, flight controls, sensors, crew station integration, and life support to the extent that the selected escape concepts could be developed within minimum weight and volume constraints yet be capable of meeting SOW requirements. The results of Task II is presented as a weight summary in Table 2.

Table 2. Weight Summary

Concept	Weight (lbs)
Dual Encapsulated Seat	1741
Single Encapsulated Seat	1055
HLV Pod Capsule	5576
VLV Pod Capsule	2972

CONCLUSIONS Conceptual designs of escape systems for hypervelocity technology class aerospace vehicles have been identified with state-of-the-art or near term state-of-the-art enabling technologies. The initial weight estimates for the selected subsystem components provide sufficient confidence for further development of the concepts. The development of computer models of the selected concepts for performance studies is being pursued by the contractor as a part of Task III. Additionally, a detailed study of the escape system weights compared to necessary common structural weights of the airframe crew station will be performed to identify the actual escape system weight penalty to the HLV and VLV.

REFERENCES

1. Statement of Work, "Hypervelocity Technology Escape System Concepts," USAF Contract No. F33615-86-C-3410, February 1986.

2. Evers, John E., CIMS CADC IIc User's Guide, Computer Intergrated Manufacturing Systems (CIMS), 20859 Old Spanish Trail, New Orleans, LA, 70129.

3. West, Christopher L., Brian R. Ummel, Roger F. Yurczyk, Analysis of Ejection Seat Stability Using Easy Program (EASIEST), AFWAL-TR-80-3014, Vol. I and II., September 1980.

4. White, B.J., USAF Crew Escape State-of-The-Art Requirements For Technology Advancement, AFWAL-TR-82-3089, December 1982.

5. Interim Technical Report For Hypervelocity Technology Escape System Concepts, Boeing Military Airplane Company, Seattle, Washington, USAF Contract No. F33615-86-C-3410, May 1987.

6. MIL-S-9479B (USAF), General Specification for Seat System, Upward Ejection, Aircraft, March 1971.

7. MIL-C-25969B (USAF), General Requirements for Capsule Emergency Escape Systems, March 1970.

8. AFSC Design Handbook, Series 1-0 General - Design Handbook DH 1-3 Human Engineering, Volumes 1 and 2, ASD/ENESS, WPAFB, OH, January 1977.

Aircraft Transparency Analysis Methods

(Paper not provided by publication date)

SURROGATE MEASURES: A PROPOSED ALTERNATIVE IN HUMAN FACTORS ASSESSMENT OF OPERATIONAL MEASURES OF PERFORMANCE

Robert S. Kennedy, Norman E. Lane, and Lois A. Kuntz
Essex Corporation
1040 Woodcock Road, Suite 227
Orlando, Florida 32803

ABSTRACT

In human factors studies, operational performance measures have de facto validity, but they also tend to have very low numerical reliability (r=.10-.30). We have proposed the use of surrogate measures as an alternative to direct assessment of operational performance for purposes of screening agents (e.g., drugs, environmental stress, etc.), particularly when the surrogate can be empirically validated against the operational criterion. The focus is on cognitive (or throughput) performances in humans as opposed to sensory (input) or motor (output) measures, but the methods should be applicable for development of batteries which will tap input/output functions.

For several years, we have been developing under NASA, NSF, and U.S. Army sponsorship, a menu of performance tasks implemented on a battery-operated portable microcomputer. Currently, 21 tests are available on a disk for microcomputer presentation. These tasks are reliable and become stable in minimum amounts of time, appear sensitive to some agents, comprise constructs related to actual job tasks, and are easily administered in most environments. We will review our progress with this program, and describe implications for human factors engineering studies in environmental stress.

INTRODUCTION

The presence of related environmental stressors in space necessitates the generation of a standardized human factors testing tool in order to detect subtle differences in the integrity of human performance and welfare. This testing tool could be used to predict premonitory onset of decrements in performance, physiology, mood, and behavior before such stressors effect operational efficiency; to enhance identification of susceptible personnel, to explore the possibility of resistance training, and to monitor the neurological status of persons subjected to hazards in their occupations, as well as for longitudinal monitoring in connection with regular physical examinations.

Environmental stressors are most often studied with a pre-, per-, post paradigm. This approach makes maximum use of the "each subject serves as his or her own control" philosophy. Repeated-measures studies, where environmental stress has served as an agent which alters performance, include weightlessness (Nicogossian & Parker, 1982), high altitude (Fowler, Paul, Porlier, Elcombe, & Taylor, 1985), temperature (Ellis, 1982), toxic chemicals (Guillion & Eckerman, 1985), pharmacological agents (Kohl, Calkins, & Mandell, 1986), pressure at depth (Logie & Baddeley, 1985), physical exercise (Englund, Ryman, Naitoh, & Hodgdon, 1985), sleep loss (Woodward & Nelson, 1974), motion (Kennedy & Frank, 1986), fatigue (West & Parker, 1975), dehydration (Banderet, MacDougall, Roberts, Tappan, Jacey, & Gray, 1984), simulated environments (McComas, 1986), vibration (LaRue, 1965, Hornick & Lefritz, 1966), and many other agents have received research consideration. Organismic and induced states within subjects have also received attention and these have invariably been with repeated-measures designs. Kiziltan (1985), Thorne, Genser, Sing, and Hegge (1983), and Donnell (1969) have examined the effects of sleep deprivation on performance. One of the forerunners of Automated Performance Testing (the Neptune Project: McKenzie, White, and Hartman, 1968), built on earlier work by Fleishman and Ellison (1962), who undertook the identification of factors common to psychomotor tests, and French (1951; French, Ekstrom, & Price, 1963) addressed the identification and measurement of cognitive abilities to produce (McKenzie et al., 1968) a fully functioning battery in the late 1960s.

However, as a practical matter, measures of operational performance are elusive and several problems remain in the assessment of human performance; chronically low retest reliability, instability across days due to learning, wide individual differences of unknown or uncontrolled variation, not knowing what to measure, etc. Reviews of the older literature concerning assessment of operational performance document the unreliability of performance measures (Lane, 1986). This problem is not a data acquisition

problem and does not disappear in our era of increasing technology, where it is now possible to measure nearly every variable we wish, 1000 times per second, and save all or play it back at will. The problem is a "what to measure" problem and is partly related to human variablity. The two metric issues which are central to an understanding of this problem are concerns with "stability" and "reliability."

Stability. A development program was undertaken several years ago by the Navy to evaluate the repeated-measures stability of paper-and-pencil based human performance tests. This program (Performance Evaluation Tests for Environmental Research [PETER]) began in 1977 (Kennedy & Bittner, 1977) and has provided specific paradigms and methods for the evaluation and selection of performance tests applicable to repeated-measures research (Carter & Kennedy, 1980). Typically, alternate forms of a test would be repeatedly administered to the same group of subjects (i.e., 1 to 15 trials). These data would then be analyzed for three types of stability -- Means, Standard Deviations, and Correlations (Kennedy, Bittner, Harbeson, & Jones, 1981).

Reliability. For the past nine years studies of military flying performance in a sophisticated flying trainer, the Visual Technology Research Simulator (VTRS), have been conducted (Lintern, Nelson, Sheppard, Westra, & Kennedy, 1981). In this research tool, the ability to hold many confounding variables constant, the precise data acquisition systems available, and the size of the retest reliability from that which is available in the field setting for the same task changes very little. For example, single carrier landing approach performances on the simulator had retest reliabilities of .23 to .32 for the mean of four trials; air-to-ground bomb miss distance reliabilities were lower, slightly above .20 for the mean of eight trials.

As has been mentioned (Lane, Kennedy, & Jones, 1986), not enough attention has been paid to the reliability of criteria or dependent variables in experimental studies. The consequence of this omission can be seen in the well known correction for attenuation formula reported by Guilford (1954) and symbolized as:

$$R_t = \frac{r_{xy}}{(r_{xx})\ (r_{yy})^{1/2}}$$

where r_{xy} is the predictive validity, r_{xx} is the reliability of the predictor, r_{yy} is the reliability of the criterion, and R_t is the true relationship. Without good reliability in x or y, the greatest possible relationship between the two is limited. For measuring operational performance this relationship is considerable. Although automated systems will permit significant amounts of data to be recorded in real time,we do not believe that the use of automated systems per se will rectify this problem.

Insufficient attention to reliability can lead to reduction of statistical power, higher sample size requirements, cost of experiments, and when hazard is involved, other problems. Without attention to reliability, the outcome can be misinterpreted. Utilizing the correction for attenuation formula (above) often changes conclusions, but not always negatively. For instance, the true predictive validities of operational criteria from paper-and-pencil aptitude tests are often misinterpreted because of criterion unreliability. Again, using our example, an operational reliability may be improved from .20 to .30 even at great expense, but predictor reliability might go from .70 to .90 with much less investment. The alliance in the denominator suggested to us a focus on developing highly reliable measure sets, separate from the operational criteria but highly similar to the criteria in skill requirements. If the measure sets correlate well with the criteria, and behave similarly under changing task conditions, perhaps they could be used for the criteria. The focus should be on developing highly reliable measurement sets, separate from the operational criteria but highly similar to the criteria in skill requirement.

We do not believe that meaningful examinations of environmental effects on human performance can be undertaken unless these performances can be adequately measured. A possible solution would be surrogate measures. Surrogate measures are those which are related or predictive of a construct of interest but are not direct measures. In our plan these are composed of tests or batteries that exhibit five characteristics: 1) stable so that the "what is being measured" is constant; 2) correlate with the performance construct; 3) sensitive to the same factors that would affect performance as the performance variable would; 4) more reliable than field measures; 5) involve minimal training time.

Surrogate measures differ from conventional performance measures in that tests need not involve operations in common with the performance measures, only components or factors in common. They also differ from "synthetic" or "job samples" because the surrogate takes so little practice and is easy to score. Given the great difficulty of obtaining reliable enough field measures to carry out stressor-sensitivity studies on the operational task itself, the case for using a surrogate is strong. Large portions of the variance in extremely complex tasks can be predicted from performance on relatively simple tests. An external test (or battery), though it cannot be as "valid" as the measure itself from a practical standpoint, may tap more of the true variance of the field performance because its reliability is much greater.

BACKGROUND

Performance Evaluation Tests for Environmental Research. In order to study the metric properties of existing performance tests, the Navy evaluated over 140 tests and tasks in a repeated-measures paradigm where a small group of subjects were examined repeatedly on alternate forms of the tests over a fifteen day period. The full report of the work of that program appears in over 90 publications, most of which are listed in Harbeson, Bittner, Kennedy, Carter, and Krause (1983) and fully reviewed and summarized in Bittner, Carter, Kennedy, Harbeson, and Krause, (1986). Only about one fourth of the tests were worthy according to the criteria mentioned above.

Microcomputerized vs. Paper-and-Pencil Versions of Tests. Although in the past much of the early work in the field of environmental effects made use of paper-and-pencil based tests, or tests which made use of modest apparatus (Bittner et al., 1986; Harbeson et al., 1983), the wide availabilty of low-cost, high-speed computer systems has encouraged psychologists to transfer to such apparatus in their laboratory studies of human performance measurement. In recent years there has been widespread interest in computerized performance tests. The Army (Thorne, Genser, Sing, & Hegge, 1985), Navy (Kennedy, Wilkes, Lane, & Homick, 1985; Kiziltan, 1985; McComas, 1986), Air Force (O'Donnell, 1981; Christal, 1981; Payne, 1982; Shingledecker, Acton, & Crabtree, 1983), and Environmental Protection Agency (Guillion & Eckerman, 1985) have active programs. These programs constitute valuable resources for the research and development of a computerized testing system.

The Automated Performance Test System (APTS) Background. The tests of the NASA battery have been implemented on a NEC PC8201A portable laptop computer and is now called the Automated Performance Test System (APTS). The 8201A was selected because of the amount of onboard memory available (64K bytes), and the low cost of the unit and peripherals (approximately $850.00). The display screen consisted of 240 x 64 pixel (40 characters by 8 lines) liquid crystal display (LCD) with adjustable contrast control. The unit is lightweight (3.8 pounds) and durable. After several tosses down a flight of 22 noncarpeted stairs, the only damage to the NEC was a crack in the housing, four keys fell off, and one horizontal line on the LCD was lost. The NEC also meets minimum requirements for approval for flight on the Space Shuttle.

All tests are written in the BASIC software language. Many functions such as prompting for input, converting lower case letters to upper case, test timing, and response timing were common to all the tests. Assembly language programs were written to perform these common functions, thereby providing more room in memory for data storage and the tests themselves. With careful utilization of the Read Only Memory (ROM) routines written by Microsoft, 10 different tests could be stored in the onboard memory, and enough memory was available to allow the data of 40 subjects to be stored for later off-line storage.

Since the initial implementation of the test battery on the NEC, the IBM Personal Computer has become an industry standard. The widespread use of the IBM and compatibles has made it the "hands-on" favorite in all sorts of laboratories. More and more requests for IBM compatible software has prompted us to convert our test battery to a format which makes it usable on IBM and compatible equipment. Indeed, today it is possible to purchase an IBM clone with 640K bytes memory, two 360K byte floppy drives, keyboard, monitor, serial and parallel interface for the same price as that of the NEC. For the same price, more capability, increased data storage, faster timing increments and better instructions are possible!

We are committed to maintaining the portability aspect of the test battery. Because of this commitment, we have selected the Zenith Data Systems ZFL-181 as the current host of the portable assessment battery. The 181 contains 640K onboard memory, two 720K byte 3.5 inch floppy drives, serial and parallel interfaces, an RGB interface, and 80 characters by 25 line super twist, backlit LCD display, and is completely IBM PC compatible. The batteries are capable of powering the unit with both drives running and the brightness control set on high for 4.2 hours. APTS - Usage. The test battery has been used in a variety of environments ranging from a classroom setting to the cockpit of jet airliners. This versatility provides the experimenter with a multitude of options with respect to the APTS usage. Because of the portability of both systems, on-the-spot testing, rather than in the laboratory, is possible.

Advantages to computer administered testing include: 1) standardized testing conditions leading to higher reliabilities; 2) reduced variability between test procedures and administrators which enhances comparison of results between similar studies; 3) accurate and objective response scoring eliminating unintelligible responses, improper scoring, and subjective interpretation; 4) complete automation of all testing, scoring, and data collection procedures resulting in a reduction of problems associated with lost or misplaced data; 5) utilization of a variety of response measures such as speed and latency; 6) presentation of complex and innovative stimuli involving a variety of sensory modalities; 7) capabilities for precise timing and control of stimulus materials; 8) immediate scoring of responses with easy access to data for rapid analysis or feedback to the subject or administrator; 9) automatic data storage with capabilities for handling quantities of diverse data over repeated trials, with large N's; 10) self-administration of interesting and challenging materials resulting in increased subject motivation and reduced boredom; 11) increased convenience and efficiency in data collection reducing the need for highly skilled

professionals or psychological technicians; 12) portability of the system with the accompanying advantages of reduced size and weight; and 13) adaptive testing, where difficulty level changes with performance, which can shorten testing time.

Despite the advantages to microbased testing, it was our view that in order to avoid unknown influences from medium differences it was necessary to compare the "good" paper-and-pencil tests of the PETER program to serve as markers in building a battery (or menu) of computer-based performance tests. In addition to the issues related to reliability and stability previously mentioned, the studies cited above also call attention to the necessity for careful preparation and evaluation of research tools during development. For example, Moran and Mefford (1959) identified the need for comparability of alternate forms. Repeated measurements must possess certain characteristics to be meaningful and easily and clearly interpretable (American Psychological Association, 1974; Jones, 1972; Lord & Novick, 1968). To summarize the characteristics, the statistical requirements for easily interpretable results of repeated-measures include level or linearly increasing means, level variances, and differential stability (Bittner et al., 1986).

Objectives of the Automated Performance Test System. In summary, the philosophy of our approach to performance test development involves three different goals. The first is to deal with only tests or tasks that can be shown to be psychometrically sound. This requires that we demonstrate stability of means and standard deviations within few administrations, and most important, that correlational stability, the stability of trial-to-trial intercorrelations, be shown to occur quickly and with high test-retest prescreening correlations (i.e., reliability). The second goal will be to demonstrate that the battery has factorial multidimensionality and that the subscales cross-correlate with earlier performance tests and other recognized instruments of ability. Then it is necessary to demonstrate and document sensitivity to factors known to compromise performance potential in the laboratory and ultimately real-world situations. Lastly, the tasks must be shown to be predictive of the types of work performed in the real world.

PSYCHOMETRIC STUDIES

NASA I. Originally, for proof of concept, we began our first testing under NASA sponsorship using the methodology of stability and reliability with a microbased computer (Kennedy, Wilkes, Lane, & Homick, 1985). Twenty subjects were tested over four replications using paper-and-pencil versions as well as the computerized version of six tests. All tests achieved stability within the four test sessions, reliability efficiencies were generally high ($\underline{r}$ >.707 for 3-minute testing), and the computerized tests were largely comparable to the paper-and-pencil version from which they were derived. The tests that were evaluated for inclusion in this experiment were Grammatical Reasoning, Pattern Comparison, Code Substitution, and the Tapping series, tests which had largely proven their metric properties in paper-and-pencil versions earlier in the PETER work. As these tests all exhibited stability and reliability within our proposed standards, all were proposed for further testing.

NASA II. In this study, in addition to evaluating stability and reliability of the tests, predictive validity was also examined. Twenty-five subjects were tested over significantly more replications (10) and tests (11) than previously. The 11 tests were concurrently administered in paper-and-pencil (marker battery) and microcomputer-based versions and compared to the Wechsler Adult Intelligence Scale (WAIS). Nine of the 11 microcomputer-based tests achieved stability. Reliabilities were generally high, with $\underline{r} \geq .77$ for 3 minutes of testing for the recommended tests. Cross-correlations of microbased tests with traditional paper-and-pencil versions and indices of stability suggest equivalency between the tests in the different modes. Correlations between certain microbased subtests and the WAIS identified common variance.

NASA III. In this experiment, we administered 21 different tests, including six short-term memory tests which had not been administered before. Air Combat Manuvering, Pattern Comparison, and Reaction Time Four Choice took the longest of the original battery to stabilize, but all tests stabilized by trial 5; the memory tests took a little longer and with only modest reliabilities.

ARMY I & II. Currently, under contract to the U.S. Army Medical Research and Development Command, we are evaluating other tests from the tri-service Performance Assessment Battery (PAB) (Englund, Reeves, Shingledecker, Thorne, Wilson, & Hegge, 1986) along with our existing menu of tests for potential inclusion into a menu of tests for the battery. This menu would permit investigators to customize a battery to their specific needs. These studies, while completed, are still in draft form. In general, the tests from the NASA battery performed better than candidate PAB tests despite differences in test administration. The best PAB tests were Recall, Mathematical Processing, and Matrix Rotation, with average reliabilities of 0.60, 0.63, and 0.71, respectively. In contrast, Item Order, Memory Search, and Successive Pattern Comparison average reliabilities were 0.43, 0.49, and 0.44, respectively. The best NASA battery tests were Grammatical Reasoning, Simultaneous Pattern Comparison, and Manikin tests, with average reliabilities of 0.84, 0.79, and 0.95, respectively. The two Reaction Time tests were also good measures, although they are highly intercorrelated. Perhaps just the 4-Choice version, which is the more reliable of the two, should be used in the future. Tapping tests, which were used in both test batteries, exhibited consistently high levels of stability and reliability. What is most intriguing, however, is the lack of significant overlap of Tapping tests with other tests, indicating their relatively pure nature. Because of the ease of administering the

measure and its independence from perceptual and cognitive tests, it may be a measure of motivation. Further testing should address this issue.

NSF II. A factor analysis was conducted on 11 selected tests from the PAB, and NASA performance test batteries were administered three times to each of 108 Central Pennslyvania college students (48 males and 60 females). The Wonderlic Personnel Test was administered just before the first and just after the last administration of the performance tests. The test-retest reliability of the Wonderlic in the total sample was .78, which yields a Spearman-Brown estimated reliability for the sum of the two Wonderlic scores, the "combined" Wonderlic, of .88. The multiple R in the total sample between the combined Wonderlic as criterion and Grammatical Reasoning (NASA) and Math Processing (PAB) as predictors ranged between .48 and .55 on the three test administrations. Factor analyses carried out on each administration yielded three consistent factors: a spatial/numerical factor on which Pattern Comparison (NASA) loads most heavily, a verbal factor of which Grammatical Reasoning (NASA) loads most heavily, and a motor factor defined by the Tapping tests (NASA). Based on these results we would recommend a core battery consisting of Grammatical Reasoning (NASA), Mathematical Processing (PAB), Pattern Comparison (NASA), and the Preferred and Nonpreferred (but not the Two-Finger) Tapping tests. This battery provides a good short estimate of IQ, based on Grammatical Reasoning and Mathematical Processing and three well identified factors, one verbal, another spatial/numerical, and the third motor. This core battery can be usefully augmented, especially in operational situations, by Code Substitution and Choice Reaction Time tests, both from the NASA battery. Manikin (NASA) is another recommended test for augmentation because it measures a different factor from IQ.

SENSITIVITY STUDIES

Altitude. Until recently, lack of an adequate human performance research tool has resulted in the employment of a variety of techniques, methods, and measures that limit systematic comparisons across altitude studies. Such limitations have delayed the development of a cohesive body of knowledge regarding human performance at altitude. Measurement and data collection inadequacies have further contributed to research difficulties. While highly controlled studies systematically relating sustained exposure to human performance are largely lacking, we believe that exceptions are beginning to appear (cf. e.g., Banderet & Burse 1984; Banderet, Benson, McDougall, Kennedy, & Smith, 1984).

The NASA battery has been tested at simulated altitude by scientists of the U.S. Air Force, and the U.S. Army Institute for Environmental Medicine (Banderet et al., 1984). The initial results show a definite cognitive performance decrement with sustained periods at altitudes of 23,000 feet, and with abrupt, short periods at 27,000 feet. However, motor performance remained essentially unchanged. An important point to note is that typical measures of performance would not have detected the effect altitude had on the mental capabilities of the participants.

Drugs. With regular doses of certain motion sickness drugs, virtually all of the scores for both motor and cognitive tests changed in a theoretically rational direction in studies conducted by Dr. Charles Wood at Louisiana State University Medical School. That is, amphetamine scores increased and scopolamine scores decreased over placebo. A simple ANOVA revealed no significant outcomes (other than that Pattern Comparison, one of the APTS tests, scores appeared to be significantly poorer with hyoscine). The within-subject variables were scopolamine and dexedrine, arranged factorially in a totally within-subject design (a more powerful approach). The results indicate that amphetamine significantly increased Nonpreferred Hand Tapping (a motor skill test) and there was a trend for increased scores on the Sternberg (an item recognition test). This would mean there were more "hits" or that latency improved. There was not a significant effect of scopolamine on Preferred-Hand Tapping. The study further showed an interaction of scopolamine and dexedrine with Two-Hand tapping. Though not statistically significant, overall it appears that scopolamine facilitates performance more when dexedrine is also present then it does without dexedrine.

Chemoradiotherapy Treatments. From the University of Washington, Dr. Parth has been studying patients who are receiving bone marrow transplants and chemoradiotherapy treatments. In this study, the tests of the basic NASA battery were administered, along with other tests, to both a patient population undergoing chemotherapy subsequent to bone marrow transplants and to a control population of sibling donors. Four replications of the battery were given spaced over one year, including prior to transplant therapy, during therapy, and in a follow-up examination. The primary purpose of NASA's use was to determine battery sensitivity to physiological stressors different from those examined in previous studies. The battery as a whole was strikingly effective in detecting performance shifts in patients and significantly differentiating patients from controls throughout the two therapy test periods. Greatest discrimination was apparent in the complex cognitive measures (i.e., Code Substitution) than in the "motor" (i.e., Tapping). Discrimination was present for both accessory and latency measures, although effects were stronger for accuracy performance.

Study with Sleep Loss. Two different studies of sleep loss have been conducted. In the first study, Kiziltan (1985) at the U.S. Naval Postgraduate School in Monterey, California, observed statistically effects on Code Substitution but obtained only directional changes (nonsignificant) on the other tests following one night without sleep. Another study was performed with the NASA battery tests at Ames Research Center in

Moffett Field, California. The experiment lasted 41 days, 30 of which was the bedrest phase. The results of this study revealed modest or no change on most tests.

In summary, for the past few years our research efforts have concerned study and identification of reliable performance measurement instruments for exotic environments. Under the sponsorship of NASA, NSF, and Army MRDC a menu of performance tasks implemented on a battery-operated portable microcomputer has been developed. These measures differ from conventional performance measures in that tests need not involve operations in common with the performance measures, only components/ factors in common. The tests also exhibit higher reliabilities ($r \geq .70$) than traditional performance measures ($r = .10-.30$). Currently, 21 tests are available on a disc for microcomputer presentation. These tasks are reliable and become stable in minimum amounts of time, appear sensitive to some agents, comprise constructs related to actual job tasks, and are easily administered and scored. Collectively these tests are known as the Automated Performance Test System (APTS). In numerous experiments the APTS has been shown to be a stable and reliable indicator of performance. If a person performs in a predictable manner and an intervening factor is introduced that may have an adverse effect on performance (i.e., zero gravity, stress) it may be detected by the APTS. Using a stable, sensitive, battery of performance tests would be analogous to taking a person's temperature, blood pressure, or weight. If administered on a daily basis it would be a form of record keeping that would show whether a person's performances were being affected by the environment or factors such as fatigue or workload. The APTS tests cognitive factors related to job performance and is therefore more predictive of performance than traditional methods of respiration, heart rate, blood pressure, et cetera.

REFERENCES

American Psychological Association, STANDARDS FOR EDUCATIONAL AND PSYCHOLOGICAL TESTS, American Psychological Association, Washington, DC, 1974.

Banderet, L. E., Benson, K. P., McDougall, D. M., Kennedy, R. S., & Smith, M., "Development of cognitive tests for repeated performance assessment," PROCEEDINGS OF THE 26TH MILITARY TESTING ASSOCIATION CONFERENCE, Munich, Germany, 1984, pp. 375-380.

Bandaret, L. E., & Burse, R. L., "Cognitive performance at 4500m simulated altitude," PROCEEDINGS OF THE 92ND ANNUAL AMERICAN PSYCHOLOGICAL ASSOCIATION, Toronto, Canada, 1984, August.

Banderet, L. E., MacDougall, D. M., Roberts, D. E., Tappan, D., Jacey, M., & Gray, P., "Effects of dehydration or cold exposure and restricted fluid intake upon cognitive performance," PROCEEDINGS OF THE NATIONAL ACADEMY OF SCIENCES, COMMITTEE ON MILITARY NUTRITION RESEARCH WORKSHOP, National Academy of Sciences, Washington, DC, 1984.

Bittner, A. C., Jr., Carter, R. C., Kennedy, R. S., Harbeson, M. M., & Krause, M., "Performance Evaluation Tests for Environmental Research (PETER): Evaluation of 114 measures," PERCEPTUAL AND MOTOR SKILLS, 63, 1986, pp. 683-708.

Carter, R. C., & Kennedy, R. S., "Selection of Performance Evaluation Tests for Environmental Research," PROCEEDINGS OF THE 24TH ANNUAL MEETING OF THE HUMAN FACTORS SOCIETY, Los Angeles, CA, 1980, pp. 320-324.

Christal, R. E., "The need for laboratory research to improve the state-of-the-art in ability testing," NATIONAL SECURITY INDUSTRIAL ASSOCIATION, DoD CONFERENCE OF PERSONNEL AND TRAINING FACTORS IN SYSTEMS EFFECTIVENESS, San Diego, CA, 1981.

Donnell, J., "Performance decrement as a function of total sleep loss and task duration," PERCEPTUAL AND MOTOR SKILLS, 29, 1969, pp. 711-714.

Ellis, H. D., "The effects of cold on the performance of serial choice, reaction time and various discrete tasks," HUMAN FACTORS, 24, 1982, pp. 589-598.

Englund, C. E., Reeves, D. L., Shingledecker, C. A., Thorne, D. R., Wilson, K. P., & Hegge, F. W., "Unified Tri-service Cognitive Performance Assessment Battery (UTC-PAB)," REPORT NO. 86-1, U.S. Army Research and Development Command, Fort Dietrick, MD, 1986.

Englund, C. E., Ryman, D. H., Naitoh, P., & Hodgdon, J. A., "Cognitive performance during successive sustained physical work episodes," BEHAVIOR RESEARCH METHODS, INSTRUMENTS & COMPUTERS, 17, 1985, pp. 75-85.

Ferris, S. H., Crook, T., Clark, E., McCarthy, M., & Rae, D., "Facial recognition memory deficits in normal aging and senile dementia," JOURNAL OF GERENTOLOGY, 35, 1980, pp. 707-714.

Fleishman, E. A., & Ellison, G. D., "A factor analysis of fine manipulative tests," JOURNAL OF APPLIED PSYCHOLOGY, 46, 1962, pp. 96-105.

Fowler, B., Paul, M., Porlier, G., Elcombe, D. D., & Taylor, M., "A reevaluation of the minimum altitude at which hypoxic performance decrements can be detected," ERGONOMICS, 28, 1985, pp. 781-791.

French, J. W., "The description of aptitude and achievement in terms of rotated factors," PSYCHOMETRIC MONOGRAPHS NO. 5., 1951.

French, J. W., Ekstrom, R. B., & Price, L. A., "Manual for kit of reference tests for cognitive factors," RESEARCH CONTRACT NONR-2214-00, Educational Testing Service, Princeton, NJ, June, 1963.

Guilford, J. P, PSYCHOMETRIC METHODS (2nd ed.), McGraw-Hill, New York, 1954, pp. 400-402.

Guillion, C. M., & Eckerman, D. A., "Field testing for neurobehavioral toxicity: Methods and methodological issues," BEHAVIORAL TOXICOLOGY, Z. Annau (Ed.), Johns Hopkins Press, Baltimore, MD, 1985.

Harbeson, M. M., Bittner, A. C., Jr., Kennedy, R. S., Carter, R. C., & Krause, M., "Performance Evaluation Tests for Environmental Research (PETER): Bibliography," PERCEPTUAL & MOTOR SKILLS, 57, 1983, pp. 283-293.

Hornick, R. J., & Lefritz, N. M., "A study and review of human response to prolonged random vibration," HUMAN FACTORS, 8, 1966, pp. 481-492.

Jones, M. B., "Individual differences," THE PSYCHOMOTOR DOMAIN, R. N. Singer (Ed.), Lea & Febiger, Philadelphia, PA, 1972, pp. 107-132.

Jones, M. B., Kennedy, R. S., & Bittner, A. C., Jr., "A video game for performance testing," AMERICAN JOURNAL OF PSYCHOLOGY, 94, 1981, pp. 143-152.

Kennedy, R. S., & Bittner, A. C., Jr., "The development of a Navy performance evaluation test for environmental research (PETER)," PRODUCTIVITY ENHANCEMENT: PERSONNEL PERFORMANCE ASSESSMENT IN NAVY SYSTEMS, L. T. Pope & D. Meister (Eds.), Navy Personnel Research & Development Center, San Diego, CA, 1977. (AD A111180)

Kennedy, R. S., Bittner, A. C., Jr., Harbeson, M. M., & Jones, M. B., "Perspectives in performance evaluation tests for environmental research: Collected papers," RESEARCH REP. NO. NBDL-80R004, Naval Biodynamic Laboratory, New Orleans, LA, 1981. (AD A111180)

Kennedy, R. S., & Frank, L. H., "A review of motion sickness with special reference to simulator sickness," 65th Annual Meeting of the Transportation Research Board, Washington, DC, 1986.

Kennedy, R. S., Wilkes, R. L., Lane, N. E., & Homick, J. L., "Preliminary evaluation of microbased repeated-measures testing system, REPORT NO. EOTR-85-1, Essex Corporation, Orlando, FL, 1985.

Kiziltan, M., "Cognitive performance degradation on sonar operated and torpedo data control unit operators after one night of sleep deprivation," Unpublished master's thesis, Naval Postgraduate School, Monterey, CA, 1985.

Kohl, R. L., Calkins, M. A., & Mandell, A. J., "Arousal and stability: The effects of five new sympathomimetic drugs suggest a new principle for the prevention of space motion sickness," AVIATION, SPACE, AND ENVIRONMENTAL MEDICINE, 57, 1986, pp. 137-143.

Lane, N. E., "Skill acquisition curves and military training," IDA PROFESSIONAL PAPER NO. P-1945, Institute for Defence Analyses, Alexandria, VA, 1986.

Lane, N. E., Kennedy, R. S., & Jones, M. B., "Overcoming unreliability in operational measures: The use of surrogate measure systems," PROCEEDINGS OF THE 30TH ANNUAL MEETING OF THE HUMAN FACTORS SOCIETY, Santa Monica, CA, 1986, pp. 1398-1402.

LaRue, M. A., "The effects of vibration on accuracy of a positioning task," JOURNAL OF ENVIRONMENTAL SCIENCES, 8, 1965, pp. 33-35.

Lintern, G., Nelson, B. E., Sheppard, D. J., Westra, D. P., & Kennedy, R. S., "Visual Technology Research Simulator (VTRS) human performance research: Phase III," NAVTRA-EQUIPCEN 78-C-0060-11, Naval Training Equipment Center, Orlando, FL, 1981.

Logie, R. H., & Baddeley, A. D., "Cognitive performance during simulated deep-sea diving," ERGONOMICS, 28, 1985, pp. 731-746.

Lord, F. M., & Novick, M. R., STATISTICAL THEORIES OF MENTAL TEST SCORES, Addison-Wesley, Reading, MA, 1968.

McComas, L. A., "Effects of simulated ship motion on the performance of underway Officer of the Deck," Unpublished master's thesis, Naval Postgraduate School, Monterey, CA, 1986.

McKenzie, R. E., White, D. D., & Hartman, B. O., "Neptune: A multielement task system for evaluating human performance, RESEARCH REP. NO. SAM-TR-69-25, U.S.A.F. School of Aerospace Medicine / Aerospace Medical Division, Brooks Air Force Base, TX, October, 1969.

Moran, L. J., & Mefford, R. B., "Repetitive psychometric measures," PSYCHOLOGICAL REPORTS, 5, 1959, pp. 260-275.

Nicogossian, A. E., & Parker, J. F., SPACE PHYSIOLOGY AND MEDICINE, NASA / Scientific & Technical Information Branch, Houston, TX, 1982.

O'Donnell, R. D., "Development of a neurophysiological test battery for workload assessment in the U.S. Air Force," PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON CYBERNETICS SOCIETY, Atlanta, GA, 1981, pp. 398-402.

O'Neal, M. R., & Cannon, T. R., "Refractive error change at the United States Air Force Academy Class of 1985," AMERICAN JOURNAL OF OPTOMETRY & PHYSIOLOGICAL OPTICS, 64, 1987, pp. 344-354.

Payne, D. L., "Establishment of an experimental testing and learning laboratory," 4th International Learning Technology Congress & Exposition of the Society for Applied Learning Technology, Orlando, FL, February, 1982,

Shingledecker, C. A., Acton, W. H., & Crabtree, M. S., "Development and application of a criterion task set for workload metric evaluation," SAE TECHNICAL PAPER SERIES NO. 831419, Warrendale, PA, 1983.

Thorne, B., Genser, S., Sing, H., & Hegge, F., "Plumbing human performance limits during 72 hours of high task load," THE HUMAN AS A LIMITING ELEMENT IN MILITARY SYSTEMS, Defense and Civil Institute of Environmental Medicine, Toronto, Ontario, Canada, 1983.

Thorne, D. R., Genser, S. G., Sing, H. C., & Hegge, F. W., "The Walter Reed Performance Assessment Battery," NEUROBEHAVIORAL TOXICOLOGY AND TERATOLOGY, 7, 1985, pp. 415-418.

West, V., & Parker, J. F., "A review of recent literature: Measurement and prediction of operational fatigue," ONR REPORT NO. 201-067). Office of Naval Research, Arlington, VA, 1975. (NTIS No. AD A008405)

Winer, B. J., STATISTICAL PRINCIPLES IN EXPERIMENTAL DESIGN (2nd ed.), McGraw-Hill, New York, 1971.

Woodward, D. P., & Nelson, P. D., "A user oriented review of the literature on the effects of sleep loss, workrest schedules, and recovery on performance, ONR REPORT NO. ACR 206, Office of Naval Research, Washington, DC, 1974. (NTIS No. AD A009778)

AUTHOR INDEX

Acharya, Narayan A. 389
Allen, Bradley P. 39
Arbabi, Mansur 315
Bachert, Robert F. 153
Baffes, Paul 53, 111
Basham, Bryan D. 247
Baudendistel, Stephen 135
Beale, G. 125
Benner, Kevin M. 45
Benton, Ronald C. 421
Biezad, Daniel J. 477
Boatman, Wayne 133
Bochsler, Dan 11
Bourgeois, B. J. 457
Boys, Randy M. 261
Brown, Richard 307
Castore, Glen 197
Chang, C. L. 209
Clarke, Margaret M. 471
Combs, J. B. 209
Corker, Kevin 253
Cornell, Matthew 355
Crouse, K.R. 239
Culbert, Chris 191
De Hoff, Ronald L. 25
deFigueiredo, Rui J. P. 441
Divona, Charles J. 471
Evers, Ken H. 153
Ferguson, Jay C. 363
Fernandez, Ken 125, 501
Gallaway, Glen R. 147
Glasson, Douglas P. 389
Goodwin, Mary Ann 203, 281
Gott, Sherrie P. 61
Graham, Joyce 335
Green, Peter E. 389
Greenwood, Daniel 97, 519
Grice, Ervin O. 293
Grieg, Joy Y. 477
Hack, Edmund C. 375
Harding, Kevin G. 413
Harrington, Nora 93
Harris, Richard A. 247
Hartrum, Thomas C. 31
Healey, Kathleen J. 375
Holtzman, Peter L. 39
Hsieh, B. -J. 125
Hua, Grace 135
Interrante, Mark 527
Jellison, Timothy G. 25
Jenkins, Lyle M. 497
Jines, Lanny A. 541
Kawamura, K. 125
Kennedy, Robert S. 551
Kesler, L. O. 513
Kozick Jr., Bill 15
Krieger, David 307
Krishen, Kumar 441
Kruchten, Robert J. 321
Kuntz, Lois A. 551
Lamont, Gary B. 31, 477
Land, K. C. 513
Lane, Norman E. 551
Lawler, Dennis G. 217
Lea, Robert N. 281
Lewantowicz, Zdsislaw H. .. 477
Loftin, R. Bowen 53
Loveland, Harry 77
Macy, S. A. 231
Malin, Jane T. 247
Malin, Janice A. 327
Marshall, Paul F. 349
McCown, Michael G. 161
McCoy, Michael S. 261
McIntire, Gary 111
McLaughlin, Larry L. 183
McNulty, Christa 335
Mitchell, Eric V. 281
Mok, Eva 493
Mraz, Richard T. 165
Muthuswamy, Yogesh K. ... 405
Myers, John K. 433
Naylor, A. W. 397, 399
Otaguro, W. S. 513
Padalkar, S. 125
Parnell, Gregory S. 287
Perrin, Bruce M. 269
Pfeifer, Michael 315
Pierce, Roger 299
Pitney, Gilbert A. 519
Pomarede, Jean-Michel L. .. 389
Reynolds, James C. 533
Reynolds, Walt 15
Rhoades, D. E. 513
Ricart, Richard 85
Riley, Gary 191, 383
Rodriguez-Moscoso, J. 125
Roewer, Paul 335

Robertson, Charles C. 203
Rowell, William F. 287
Rua, Monica 53, 111
Savely, Robert T. 191
Schaffer, J. 125
Schudy, Robert 253, 371
Scott, Steven 527
Seward, Walter D. 173
Shepanski, J. F. 231
Sheridan, Philip L. 487
Shin, Kang G. 405
Shute, Valerie J. 71
Simmons, Robert M. 341
Sliwa, Nancy E. 465
Smith, Terence R. 519
Stachowitz, R. A. 209
Stock, T. S. 209
Swigger, Kathleen 77
Thompson, William M. 471
Throne, Robert D. 405
Valusek, John R. 287
Vaughan, David S. 269
Villarreal, James 93, 111
Vinz, F. 125
Volz, R. A. 397, 399
Wagner, Robert E. 363
Wallace, Richard M. 119
Wang, Lui 11, 53
Wechsler, D. B. 239
White, Douglas A. 45
Williams, Linda J. F. 217
Wingert, William B. 223
Wise, Ben P. 269
Yadrick, Robert M. 269
*Van Holle, J. 1

*See paper #1 for instructions.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
CP-2491		

4. Title and Subtitle	5. Report Date
First Annual Workshop on Space Operations Automation and Robotics (SOAR '87)	October 1987
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Sandy Griffin, editor	S-567
	10. Work Unit No.

9. Performing Organization Name and Address	11. Contract or Grant No.
Lyndon B. Johnson Space Center Houston, Texas 77058	
	13. Type of Report and Period Covered
12. Sponsoring Agency Name and Address	Conference Publication
National Aeronautics and Space Administration, Washington, DC 20546 U.S. Air Force, Washington, DC 23304 University of Houston-Clear Lake, Houston, TX 77058	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

Papers presented at the First Annual Workshop on Space Operations Automation and Robotics (SOAR '87), hosted by the NASA Lyndon B. Johnson Space Center, Houston, Texas on August 5, 6 and 7, 1987 are documented herein. During the 3 days, approximately 90 technical papers were presented by experts from NASA, the USAF, universities and technical companies. Two panel discussions entitled "Expert System Tools" and "Robotics: Applications in Space" also were held but are not documented herein. Technical topics addressed include AI/Expert Systems covering the areas of knowledge acquisition, uncertainty, reasoning, monitor and control and architectures for real-time AI. Also addressed were topics of Human Factors and Environment. The Technical Robotics sessions included kinematics, dynamics, mobility and control, sensing and perception, manipulation, end effectors, telerobotics, task planning and reasoning. Application Development and Transition sessions included areas of automation of checkout, ground support and logistics, automated software development, man-machine interfaces, neural networks and related studies, functional utilization, systems engineering and distributed/parallel architectures, validation and verification.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Expert system, knowledge acquisition, human factors, kinematics, dynamics, mobility and control, sensing and perception, manipulation, end effectors, telerobotics, automated software development, man-machine interfaces, neural networks, validation, verification	Unclassified - Unlimited Subject Category: 59

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price*
Unclassified	Unclassified	574	

*For sale by the National Technical Information Service, Springfield, Virginia 22161