NASA Conference Publication 2492

Third Conference on Artificial Intelligence for Space Applications

Part I

Compiled by
J. S. Denton,
M. S. Freeman,
and M. Vereen

Proceedings of a conference sponsored by
The University of Alabama in Huntsville and
The National Aeronautics and Space Administration
and held in Huntsville, Alabama
November 2 -3, 1987

National Aeronautics
and Space Administration

**Scientific and Technical
Information Branch**

1987

CONFERENCE COORDINATOR

THOMAS S. DOLLMAN
INFORMATION AND ELECTRONIC SYSTEMS LABORATORY
MARSHALL SPACE FLIGHT CENTER

CONFERENCE CO-CHAIRMEN

JUDITH S. DENTON, MSFC

GARY L. WORKMAN, UAH

PROCEEDINGS COORDINATOR

MARY VEREEN

LOGISTICS AND ARRANGEMENTS COMMITTEE

DAVID WEEKS, MSFC

KAREN MACK, UAH

TECHNICAL AND PROGRAM COMMITTEE

MICHAEL FREEMAN, MSFC

JAMES JOHANNES, UAH

ROWLAND BURNS, MSFC
DAN HAYS, UAH
ELAINE HINMAN, MSFC
BERNARD SCHROER, UAH
CAROLINE WANG, MSFC

PUBLICATIONS COMMITTEE

LUSTER INGRAM, MSFC

BERNARD SCHROER, UAH

JEANETTE REISZ, MSFC
WILLIAM SELIG, MSFC

PUBLICITY COMMITTEE

JUDITH S. DENTON, MSFC

GARY L. WORKMAN, UAH

GERRY HIGGINS, MSFC
KAREN MACK, UAH

ADVISORS

JONATHAN HAUSSLER, MSFC
ELAINE HINMAN, MSFC
GABE WALLACE, MSFC

FOREWORD

This document contains the Proceedings of the Third Conference on Artificial Intelligence for Space Applications (CAISA), sponsored by the National Aeronautics and Space Administration's (NASA's) George C. Marshall Space Flight Center (MSFC) and the University of Alabama in Huntsville (UAH). There is widespread interest throughout the aerospace community in utilizing scientific and technical developments from the field of Artificial Intelligence (AI) to enhance our space program. For NASA, future activities in space will rely on the effective utilization of key AI components. The intent of this conference is to provide an opportunity for those groups and individuals who employ AI methods in space applications to identify common goals, to compare the effectiveness of the various approaches being investigated, and to discuss issues of general interest in the AI community. The Third CAISA brings together a diversity of scientific and engineering work and is intended to promote thoughtful discussion concerning the possibilities created by this work.

AI contains many components, some of which can be selectively applied to develop more competent, less demanding flight/ground systems. This is the message of the invited speakers at our keynote session. As the participants in this conference have recognized, there is no more fascinating - nor more potent - combination of technologies than is found in the use of Artificial Intelligence to support our exploration of space. The potential benefits to our society and all mankind are literally limitless. The presentations in our technical sessions discuss various aspects of this technology. The papers presented were selected through a careful review of the submitted abstracts by at least five members of the Technical Committee. The selected presentations, represented by the papers or abstracts herein, were organized into twenty-one technical sessions. Every effort was made to minimize the conflicts arising from parallel sessions. The broad range of topics presented is indicative of the interest in NASA's goals commonly found in the AI community.

This conference would not have been possible without the dedicated efforts of many people. First, I would like to thank the authors whose research and development efforts are presented here. Second, I thank the members of all the committees, the advisors, and the volunteers who planned and implemented the numerous activities which enable a conference such as this one. I thank the exhibitors for their efforts to develop and demonstrate tools for implementing many of the ideas discussed during these two days. And, finally, I thank the invited speakers and the many other people from NASA and UAH whose interest in Artificial Intelligence and space applications makes this conference both possible and meaningful.

Thomas Dollman

PRECEDING PAGE BLANK NOT FILMED

TABLE OF CONTENTS

VALIDATION AND VERIFICATION

Verification Issues for Rule-Based Expert Systems
Chris Culbert, Gary Riley, and Robert T. Savely 1

Using Output to Evaluate and Refine Rules in Rule-Based Expert Systems
D. C. St.Clair, W. E. Bond, and B. B. Flachsbart 9

Temporal and Contextual Knowledge in Model-Based Expert Systems: Ford Aerospace's Paragon Project
Tihamer Toth-Fejel and Dennis Heher 15

Methodology for Testing and Validating Knowledge Bases
C. Krishnamurthy, S. Padalkar, J. Sztipanovits and R. B. Purves 21

DEVELOPMENT TOOLS

CLIPS as a Knowledge Based Language
James B. Harrington 33

An Easy-to-Use Diagnostic System Development Shell
L. C. Tsai, J. B. Ross, C. Y. Han, and W. G. Wee 41

An Inference Engine for Embedded Diagnostic Systems
B. R. Fox and L. T. Brewster 47

CLIPS: An Expert System Tool for Delivery and Training
Gary Riley, Chris Culbert, Robert T. Savely and Frank Lopez 53

TELEOPERATOR/ROBOTICS

Issues Associated with Telerobotic Systems in Space
S. Hofacker and B. J. Schroer 59

PRECEDING PAGE BLANK NOT FILMED

Telerobotic Controller Development
W. S. Otaguro, L. O. Kesler, Ken Land
and Don Rhoades 65

Software Simulation of Time Delay in
Teleoperation
Wayne Goode 73

NEW TECHNOLOGIES FOR SPACE STATION AUTOMATION

MTK: An AI Tool for Model Based Reasoning
W. K. Erickson and M. R. Schwartz 79

Integration of Symbolic and Algorithmic
Hardware and Software for Automation
of Space Station Subsystems
H. R. Gregg, C. M. Wong, E. Hack
and K. J. Healey 81

Requirements and Options for Communications
Services in Support of the Systems Autonomy
Demonstration Project
R. M. Brown and Robert Yee 83

Knowledge Based System Verification and Validation
as Related to Automation of Space Station Subsystems:
Rationale for a Knowledge Based System Lifecycle
K. D. Richardson, P. Friedland and C.M. Wong 85

Monitoring of Space Station Life Support
Systems with Miniature Mass Spectrometry and
Artificial Intelligence
R. A. Yost, J. V. Johnson and C. M. Wong 87

DESIGN DATA CAPTURE

HSTDEK: Developing a Methodology for Construction
of Large-Scale, Multi-Use Knowledge Bases
Dr. Michael S. Freeman 89

Knowledge-Based Monitoring of the Pointing Control
System on the Hubble Space Telescope
Larry L. Dunham, Thomas J. Laffey, Simon A. Kao,
James L. Schmidt, and Jackson Y. Read 95

TALOS: A Distributed Architecture for
Intelligent Monitoring and Anomaly Diagnosis
of the Hubble Space Telescope
Bryant G. Cruse 101

A Knowledge-Based System for Monitoring the Electrical Power System of the Hubble Space Telescope
Pat Eddy 103

FAULT DIAGNOSTICS

Artificial Intelligence and Space Power Systems Automation
D. Weeks 109

Embedded Expert System for Space Shuttle Main Engine Maintenance
J. Pooley, W. Thompson, T. Homsley, W. Teoh
J. Jones, and P. Lewallen 115

Qualitative and Temporal Reasoning in Engine Behavior Analysis
W. E. Dietz, M. E. Stamps, and M. Ali 121

Exploring Hypotheses in Attitude Control Fault Diagnosis
Benjamin Bell 123

EUREX D: An Expert System for Failure Diagnosis and Recovery in the TCS of the European Retrievable Carrier EURECA
A. Kellner, W. Belau, and N. Schielow 131

TASK PLANNING FOR ROBOTIC APPLICATIONS

Task Path Planning, Scheduling, and Learning for Free-Ranging Robot Systems
G. Steve Wakefield 137

Development of a Task Level Robot Programming and Simulation System
H. Liu, K. Kawamura, S. Narayanan, G. Zhang,
H. Franke, M. Ozkan and H. Arima 143

Goal Driven Kinematic Simulation of Flexible Arm Robot for Space Station Missions
P. Janssen and A. Choudry 151

Heuristic Search in Robot Configuration Space Using Variable Metric
Ben J. H. Verwer 153

PARALLEL PROCESSING

A Multiprocessing Architecture for
Real-Time Monitoring
Thomas J. Laffey, James L. Schmidt,
Jackson Y. Read and Simon A. Kao 155

Expert Systems Relations in Space Applications
Michael Brady 161

Problem Solving as Intelligent Retrieval from
Distributed Knowledge Sources
Z. Chen 165

Application of Parallel Distributed Processing to
Space Based Systems
J. MacDonald and H. L. Heffelfinger 171

SAFETY, RELIABILITY, AND QUALITY ASSURANCE

Spacelab Data Processing Facility (SLDPF)
Quality Assurance Expert Systems Development
Lisa Basile, Angelita C. Kelly 181

FMEAssist: A Knowledge-Based Approach to
Failure Modes and Effects Analysis
James R. Carnes and Dannie E. Cutts 187

ESSAA: Embedded System Safety Analysis Assistant
Peter Wallace, Joseph Holzer, Sergio Guarro,
and Larry Hyatt 193

Intelligent Process Development of Foam Molding
for the Thermal Protection System (TPS)
of the Space Shuttle External Tank
S. S. Bharwani, J. T. Walls, and M. E. Jackson 195

KNOWLEDGE ACQUISITION

Space Shuttle Main Engine Anomaly Data
and Inductive Knowledge Based Systems:
Automated Corporate Expertise
Kenneth L. Modesitt 203

Expert System to Analyze High Frequency
Dependent Data for the Space Shuttle
Main Engine Turbopumps
Raul C. Garcia, Jr. 213

The Use and Generation of Examples in
Computer-Based Instructional Systems
W. J. Selig and J. D. Johannes 221

Interactive Knowledge Acquisition Tools
Martin J. Dudziak and Jerald L. Feinstein 227

SIMULATION

Automatic Mathematical Modeling for
Space Application
C. K. Wang 233

LISP Based Simulation Generators
for Modeling Complex Space Processes
F. T. Tseng, B. J. Schroer and W. Dwan 243

Knowledge Based Simulation
P. A. Newman 249

Rapid Prototyping and AI Programming Environments
Applied to Payload Modeling
Richard S. Carnahan, Jr. and Andrew P. Mendler 255

COMPUTER VISION

Orbital Navigation, Docking, and Obstacle Avoidance
as a Form of Three Dimensional Model-Based
Image Understanding
J. Beyer, C. Jacobus, and B. Mitchell 261

Computing 3-D Structure of Rigid Objects
Using Stereo and Motion
Thinh Nguyen 263

Real Time AI Expert System for Robotic
Applications
John F. Follin 269

Solid Modeling for the Manipulative Robot Arm
(POWER) and Adaptive Vision Control for
Space Station Missions
V. Harrand and A. Choudry 271

KNOWLEDGE REPRESENTATION I

Iterative-Deepening Heuristic Search
for Optimal and Semi-Optimal Resource Allocation
Susan M. Bridges and James D. Johannes 273

Similarity Networks as a Knowledge Representation
for Space Applications
D. Bailey, D. Thompson and J. L. Feinstein 279

Discovery and Problem Solving: Triangulation
as a Weak Heuristic
Daniel Rochowiak 285

Commonality Analysis as a Knowledge
Acquisition Problem
Dorian P. Yeager 291

PLANNING

Qualitative Models for Planning:
A Gentle Introduction
James D. Johannes and J. R. Carnes 297

Operational Aspects of a Spacecraft
Planning/Scheduling Expert System
David R. McLean, Ronald G. Littlefield,
and David S. Beyer 303

Planning and Scheduling for Robotic Assembly
B. R. Fox 309

Planning Activities in Space
Kai-Hsiung Chang 315

INTELLIGENT MAN/MACHINE INTERFACES

Intelligent Man/Machine Interfaces
on the Space Station
Rodney Daughtrey 321

Interfacing the Expert: Characteristics and
Requirements for the User Interface in
Expert Systems
Andrew Potter 327

Space Languages
Dan Hays 333

An Innovative Workstation
James A. Villarreal 339

KNOWLEDGE BASE/DATA BASE MANAGEMENT SYSTEMS INTEGRATION

Conceptual Information Processing -
A Robust Approach to KBS-DBMS Integration
Allen V. Lazzara, W. Tepfenhart,
R. White, and R. Liuzzi 345

Foundation: Transforming Data Bases into
Knowledge Bases
R. B. Purves, J. R. Carnes and D. E. Cutts 353

The Intelligent User Interface for NASA's
Advanced Information Management Systems
William J. Campbell, Nicholas Short Jr.,
Larry Rolofs, and S.L.Wattawa 359

SCHEDULING

An AI Approach for Scheduling Space-Station
Payloads at Kennedy Space Center
D. Castillo, D. Ihrie, M. McDaniel,
and R. Tilley 361

Scheduling Spacecraft Operations
Daniel L. Britt, Amy L. Geoffroy,
Philip R. Schaefer, and John R. Gohring 371

The Resource Envelope as a Basis for
Space Station Management System Scheduling
Joy Bush and Anna Critchfield 377

Prototype Resupply Scheduler
Steve Tanner, Angie Hughes, and Jim Byrd 383

NEURAL NETS

Neural Network Based Speech Synthesizer:
A Preliminary Report
James A. Villarreal and Gary McIntire..... 389

The Power of Neural Nets
J.P. Ryan and B. Shah 395

Neural Networks as a Possible Architecture
for the Distributed Control of Space Systems
E. Fiesler, A. Choudry, and J. VanderZijp 401

KNOWLEDGE REPRESENTATION II

A Data Structure and Algorithm for Fault Diagnosis
Edward L. Bosworth 403

Case-Based Reasoning for Space Applications: Utilization of Prior Experience in Knowledge-Based Systems
James A. King 409

Knowledge Representation by Connection Matrices: A Method for the On-Board Implementation of Large Expert Systems
A. Kellner 415

RB-ARD: A Proof of Concept Rule-Based Abort Region Determinator, IR & D 87267 26701
Richard Smith and John Marinuzzi 421

AUTOMATIC PROGRAMMING

A Learning Apprentice for Software Parts Composition
B. P. Allen and P. L. Holtzman 423

Automatic Program Generation from Specifications Using Prolog
Alex Pelin and Paul Morrow 425

On Acquisition of Programming Knowledge
Ashok T. Amin 427

Application of Artificial Intelligence to Impulsive Orbital Transfers
Rowland E. Burns 433

REAL-TIME APPLICATIONS

A Framework for Real-Time Distributed Expert Systems: On-Orbit Spacecraft Fault Diagnosis, Monitoring and Control
Richard L. Mullikin 439

TES - A Modular Systems Approach to Expert System Development for Real Time Space Application
Brenda England and Ralph Cacace 445

Prototype Space Station Automation System Delivered and Demonstrated at NASA
Roger F. Block 447

Applications of Expert Systems for Satellite Autonomy
A. Ciarlo and P. Donzelli 453

Index by Author 459

N88-16361

VERIFICATION ISSUES FOR 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 represent one important by-product of Artificial Intelligence research efforts. They have been under development for many years and have reached commercial viability in the last three to four years. 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. Expert system applications are computer programs and the same definitions for V&V apply to expert systems. Consequently, expert systems require the same assurance of correctness as conventional software.

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. This confusion must be resolved if expert systems are to succeed. As with conventional software, the key to effective V&V is through the proper use of a development methodology which both supports and encourages the development of verifiable software.

THE COMMON EXPERT SYSTEM DEVELOPMENT METHODOLOGY

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 most appropriate for solving a problem. Therefore, most expert system development is commonly done by rapid prototyping. The primary purpose of the 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 very difficult or 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 [1]. 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.

The goal of any software development methodology is to produce reliable code that is both maintainable and verifiable. A software development methodology for expert systems must serve a similar purpose as one for conventional software. However, there are some differences between expert systems and conventional software which will affect the development methodology. Development methodologies for expert systems are discussed in more detail in another paper by the authors [2]. Suffice to say here that some kind of development methodology must be chosen and applied to support effective V&V.

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. A very appropriate technique would be a direct derivative of the methods used to develop procedures, flight rules, and flight software for the Apollo and Shuttle programs. This technique consists 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 from past experience, the analysis efforts typically use high fidelity simulations based on system models to derive and evaluate control parameters. If expertise is available, the knowledge is reviewed by the panel and 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 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.

Programs of the complexity and size with which NASA regularly deals make this approach mandatory. Smaller programs generally will not require the resources or effort involved in verifying a system to this extent. 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.

The panel approach to verification discussed above is very effective at ensuring that the knowledge in the expert system is both correct and complete. Verification of conventional software also covers feasibility, maintainability, and testability. These verification efforts are generally done early in the design phase and should also be done for an expert system. The coded rules must also be examined to ensure that the consistency and completeness of the design is properly incorporated in the software.

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 rules by automated tools [3]. 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. They can be done by the whole panel, or more likely, small teams of people drawn from the whole panel.

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 also be verified. The use of rapid prototyping can be extended from testing representation to testing some of these areas as well. Iterative development can go a long way to ensuring that the final system truly meets the user needs in these kind of areas.

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. A test plan should be written which discusses how the final expert system will be tested.

OTHER ISSUES FOR EXPERT SYSTEM V&V

There are other differences between between conventional software and expert systems, and those differences will affect V&V efforts. Some of the differences are discussed in reference [4] and summarized below.

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. For a rule-based expert system, 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.

Verifying the Inference Engine

The inference engine in a rule-based expert systems is a completely separate piece of code anc can be fully verified independently from the rest of the expert system.

Verifying the Expert

This question is automatically resolved as long as the expert system is validated. The panel approach discussed in this paper provides continual feedback on the correctness of the experts knowledge.

Real-Time Performance

Most conventional programs provide performance "guarantees" through extensive simulation of the expected performance environment. Expert systems can provide the same kind of performance "guarantees". 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, but can be done for a given data set entered in a specific sequence.

Complex Problems with Multiple Experts

The panel review method already discussed here 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

Tracing requirements after they have been coded in rules may be more difficult than for conventional code, particularly when hybrid representation techniques are used, i.e. when both rules and objects are used to satisfy the program's requirements. This is an area that needs further consideration.

Verifying the Boundaries of the Expert System Domain

V&V of an expert system must be carefully aimed at identifying the boundaries of a problem since the experts sometimes can not readily do so. V&V must also ensure that the expert system fails gracefully in these circumstances.

There are additional issues not discussed in reference [4]. These are discussed more fully below.

Reasoning under Uncertainty

Some expert system applications deal with incomplete, inconsistent, or uncertain information. Humans do a very good job of reasoning under uncertainty, but it can be very difficult to develop consistent models which exactly duplicate this process. Numerous methods have been developed to allow expert systems to deal with this type of information, such as fuzzy logic, probability methods like Bayes theorem, Dempster-Schafer theory, certainty factors, etc. The nature of how humans use this type of information makes it very difficult to verify in an expert system. Different people

may give different answers when presented with the exact same information. V&V efforts must focus on two things; (1) verifying that the answers suggested in uncertain situations are 'acceptable' answers. The definition of 'acceptable' may be problem dependent, and (2) if uncertain information is combined, the method used to provide a certainty factor to the result must be consistent.

Maintaining a verifiable system

Long-term maintenance of an expert system is a poorly understood topic, primarily because there is little actual experience in this area. Soloway, et al. [5] discuss some of the difficulties in maintaining XCON, one of the largest and oldest expert systems in use today. They point out that XCON is a very dynamic system, with extensive changes occurring regularly. As with conventional software, most expert systems will change and V&V must be performed each time the modified system is released. The nature of almost all rule-based languages makes true modularization of code more difficult than with conventional software. Therefore, rule-based systems presently require complete retesting with every release, using a library of test cases. Good programming practices such as using explicit control features and simple rules are important aids, but may not be sufficient to prevent extensive retesting. This area will be better understood when more applications reach maintenance stages.

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 both users and 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 testable requirements. Without requirements, V&V are meaningless concepts. An extension of the flight technique panels used in previous NASA programs should provide both documented requirements and very high levels of verification for expert systems.

REFERENCES

[1] 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,

[2] Culbert, C.J., Riley, G., and Savely, R.T., "An Expert System Development Methodology Which Supports Verification and Validation", to be published.

[3] Stachowitz, R.A. and Combs, J.B., "Validation of Expert Systems", Proceedings Hawaii International Conference on Systems Sciences, Kona, Hawaii, January 6-9, 1987.

[4] Culbert, C.J., Riley, G., and Savely, R.T., "Approaches to the Verification of Rule-based Expert Systems", Proceedings of SOAR'87: Space Operations-Automation and Robotics Conference, Houston, TX., August 1987.

[5] 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 AAAI-87, Sixth National Conference on Artificial Intelligence, Seattle, WA., July 1987.

N88-16362

Using Output to Evaluate and Refine Rules in Rule-Based Expert Systems*

D.C. St. Clair**, W.E. Bond, and B.B. Flachsbart
McDonnell Douglas Research Laboratories
McDonnell Douglas Corporation
St. Louis, MO

ABSTRACT

As space systems become increasingly complex and ambitious, the need for reliable expert systems to perform monitoring and diagnostic functions becomes more critical. Rule-based expert systems typically require large knowledge bases which must be carefully evaluated before being used in space vehicle operations. In the evaluation/refinement process, the knowledge engineer and domain experts evaluate expert system output and refine the rule base. The rule base size, coupled with rule interdependencies, makes this a very difficult task.

The research described suggests a method to compare the output set (E) of a rule-based expert system with a known set of correct conclusions (C) for a given set of input data and make decisions on how to refine the rule base. Using the techniques presented, system developers can evaluate and refine rules more accurately.

INTRODUCTION

Expert system evaluation/refinement attempts to insure that the conclusions of an expert system match those of a human expert. Typically, this process is accomplished by having the knowledge engineer input cases where behavior is known by the domain expert. The predictions made by the expert system are then compared to the "correct" answers. Where the results differ, the knowledge engineer works with domain experts to:

1. Locate and refine rules whose performance is questionable,
2. Identify missing rules and add them to the knowledge base,
3. Resolve conflicting rules, and
4. Remove extraneous rules.

This process is time-consuming and difficult to apply when the knowledge base is large and has many rule interdependencies. Learning techniques suggest some methods which can be used to expedite the process [1,3]. In particular, these techniques can be used to help isolate questionable rules and suggest avenues which should be explored to correct the problems.

The task is to modify a set of rules of the form hypothesis implies conclusion, viz.

$$H \rightarrow K$$

where H and K contain one or more propositions or negated propositions connected with conjunctions. The components of each rule come from a description space which has been determined by the knowledge engineer and the domain expert. Identification of the description space is an important and difficult problem since it contains all propositions used to describe the environment. Existing rules are refined by altering the propositions within H and/or K. New rules are created by combining propositions from the description space. The process of evaluation may indicate the description space needs to be expanded.

---

* This work was supported by the McDonnell Douglas Independent Research and Development program.

** Dr. St. Clair is Professor of Computer Science at the University of MO-Rolla, Graduate Engineering Center in St. Louis. He is currently on leave at MDRL.

PRECEDING PAGE BLANK NOT FILMED

The use of output to evaluate and refine rules necessitates the collection of some additional information. A trace of each rule chain producing system output along with corresponding rule unifications must be maintained for each test scenario to allow individual rules to be evaluated. In addition, two experience indicators must be maintained for each rule. A rule's "times fired" statistic is incremented each time it is fired. A rule's "times correct" statistic is incremented each time it participates in a rule chain leading to a correct system response. These statistics are used in the evaluation/refinement process.

COMPARISON OF EXPERT SYSTEM OUTPUT WITH KNOWN RESULTS

For a specific test scenario, three different conditions can be identified by comparing the output set (E) of the expert system with the set of known correct results (C). The three basic set relationships are shown in Figure 1.

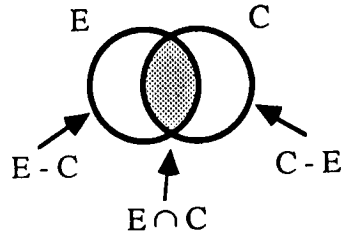


Figure 1. Comparison of Output with Known Results.

Each system output conclusion, $e_i \in E$, is the result of expert system input causing a chain of one or more rules to fire. Those $e_i \in E \cap C$ represent rule chains which terminate with a rule whose conclusion provides correct system response. Those $e_i \in E - C$ represent rule chains whose terminating rule conclusion produces an incorrect system response. Rule chains producing such output contain one or more incomplete or incorrect rules. Elements $c_i \in C - E$ indicate incorrect or missing rules.

The comparison of sets E and C for a specific test scenario will not identify cases where incorrect rule chains produce a correct conclusion. In many cases however, the identification of faulty rules in such chains will occur by applying the evaluation/refinement process to numerous test scenarios.

Some erroneous conditions can not be completely uncovered by comparing the contents of sets E and C. This includes cases where set E contains conflicting conclusions. In addition, each rule in an expert system which completely satisfies a suite of test scenarios will have identical "times fired" and "times correct" statistics. A rule whose "times fired" statistic is zero has not participated in the test suite.

EVALUATION/REFINEMENT OF RULES

The process of evaluating/refining rules consists not only of "fixing" incorrect or missing rules but of identifying rules which consistently work well and removing rules which are no longer needed by the system. The techniques described are iterative. To apply these techniques, each scenario in the test suite is processed by the expert system. Next, system evaluators use this output and the sets of known correct conclusions to perform the evaluation/refinement process. Then, the revised expert system reprocesses the test suite and the results are again evaluated/refined. Both the comprehensiveness of the test suite and the quality of expert system

design determine the number of times this cycle must be repeated. During each cycle, rule experience indicators must be updated so they accurately reflect rule performance. This section suggests ways of performing the evaluation/refinement process.

Case 1: $e_i \in E - C$

The rule chain leading to a conclusion $e_i \in E - C$ represents an error of commission. It contains one or more rules which should not have fired. The evaluation/refinement of such rule chains requires solution of what Minsky termed the credit/blame assignment problem [4]. The solution of this problem identifies rules responsible for incorrect system behavior. Bundy, et al.[3] describe two basic techniques utilized by rule learning programs for identifying the first faulty rule within a chain. The identification of multiple faults within a chain requires repeated application of the evaluation/refinement process.

The first technique compares the actual rule chain with the chain which should have fired. Some programs require this ideal chain as input [2] while others [5] attempt to derive it by analysis using problem-solving and inference techniques. The first difference between the chains indicates which rule is faulty. The necessity of identifying the ideal rule chain makes this technique difficult to apply.

The second technique for finding a faulty rule is called Contradiction Backtracking. This technique, developed by Shapiro [7] does not require the identification of an ideal chain. Assuming the actual rule chain concludes with $e_i \in E - C$, Shapiro's algorithm begins by examining the last resolution step which lead to e_i. If the propositions which were resolved to produce e_i are true, select the branch of the tree which contains these propositions as part of the rule hypothesis, else select the other branch. Backtracking up the resolution tree is continued in this manner until a rule from the rule base is reached. This is the faulty rule. Both Shapiro and Bundy, et al. give examples of Contradiction Backtracking.

Once a suspect rule is located, its evaluation can lead to one of several conclusions.

1. The hypothesis of the rule is correct but the conclusion is incorrect. This situation is resolved by correcting the rule's conclusion.
2. The hypothesis of the rule is incorrect. Offending propositions in the hypothesis are replaced with correct ones.
3. The hypothesis of the rule is incomplete. Additional propositions must be added to the hypothesis to restrict the firing of the rule. The process of restricting a rule's application in this way is called discrimination [1]. The need for adding additional propositions to a hypothesis may lead to the discovery that the description space for the problem is incomplete.

The alteration of a rule should be done carefully since such changes are likely to effect other chains in which the rule participates. If the values of one rule's experience indicators vary from the experience indicators of other rules in the chain, it is highly likely at least one rule in the chain participates in other chains.

Case 2: $e_i \in E \cap C$

Since those conclusions $e_i \in E \cap C$ are correct, the "times correct" statistic is incremented for each rule in the associated chain. These statistics are helpful when trying to correct faulty rules, since they provide a history of each rule's performance. Incrementing these statistics indicate the rule has participated in a chain which leads to a correct conclusion. They do not indicate that each rule in the chain is correct.

Case 3: $c_i \in C - E$

A known correct conclusion $c_i \in C - E$ represents an error of omission. This can happen for two basic reasons:

1. A chain exists for producing this conclusion but it contains one or more incorrect rules, or
2. No chain resulting in this conclusion exists.

Finding existing faulty chains in this case is extremely difficult unless the ideal trace is known. In simple cases, it may be possible to find a rule whose conclusion matches c_i but whose hypothesis is incorrect. If a suspect rule can be found, its hypothesis may contain incorrect or overrestrictive propositions. In the latter case, it may be possible to generalize the rule by removing the overrestrictive propositions [1]. In many cases, generalization results in combining several rules into one. Since rules being generalized may participate in other rule chains, these chains must be examined before performing generalizations.

When no chain exists for producing a missing conclusion, one of two types of refinements may be made. A new chain can be created which terminates with a rule whose conclusion is c_i. In many cases, the basic system architecture helps suggest how the rule chain should be created. This process actually expands the rule base of the expert system. St. Clair, et al. [6] used this discovery technique in an adaptive diagnostic expert system which automatically refines its knowledge base. Alternately, it may be possible to add the missing conclusion to a rule which has produced a correct conclusion $e_i \in E \cap C$. This option is viable only if e_i and c_i always occur together. If this is not the case, a new rule chain should be added which terminates with the conclusion c_i.

Case 4: Other Cases for Evaluation/Refinement

Rule evaluation/refinement is incomplete as long as the system contains rules whose "times correct" to "times fired" statistics are not equal. Such rules are members of incorrect rule chains. If a rule is incorrect, it should be evaluated/refined as indicated above while carefully noting the chains in which it participates. Such a rule may make a correct contribution to some chains and an incorrect contribution to others. It may be necessary to replace rules of this type by one or more new rules.

A rule having a small "times fired" statistic has contributed very little to the expert system's operation. This may be due to the fact that the rule has not applied to the scenarios tested or it is extraneous and makes little or no contribution to system performance. The former case can be resolved by utilizing test scenarios which fire the rule. Extraneous rules occur as a result of errors in system design or because the refinement of other rules has removed them from rule chains. Removal of extraneous rules from the knowledge base may be desirable.

Set E must be reviewed to determine if conflicting conclusions exist. Conflicting conclusions result when one rule chain produces a conclusion inconsistent with that of another, for example, when one conclusion requests replacement of unit A and another requests adjustment of unit A. The rule chain belonging to the incorrect conclusion must be refined.

Cases in which $e_i = e_j$ for $i \neq j$, indicate that two or more rule chains led to the same conclusion. This condition may be a result of the original system design or it may arise from subsumption caused by use of the generalization and discrimination mechanisms described above. Repetition of results is not always undesirable; however, it serves as an indicator that the participating rule chains should be evaluated and those rules which are redundant should be removed.

CONCLUSIONS

The techniques described provide an effective tool which knowledge engineers and domain experts can utilize to help in evaluating and refining rules. These techniques have been used successfully as learning mechanisms in a prototype adaptive diagnostic expert system [6] and are applicable to other types of expert systems. The degree to which they constitute complete evaluation/refinement of an expert system depends on the thoroughness of their use.

REFERENCES

1. Blaxton T. A. and Kushner, B. G., *An Organizational Framework for Comparing Adaptive Artificial Intelligence Systems*, **1986 Proceedings of the Fall Joint Computer Conference,** IEEE Computer Society, November 1986, pp. 190-199.

2. Brazdil, P., *A Model for Error Detection and Correction*, Ph.D. Dissertation, University of Edinburgh, 1981.

3. Bundy, A., Silver, B., and Plummer, D., *An Analytical Comparison of Some Rule-Learning Programs*, **Artificial Intelligence**, Vol. 27, 1985, pp. 137-181.

4. Minsky, M., *Steps Towards Artificial Intelligence*, **Computers and Thought**, E.A. Feigenbaum and J. Feldman (Ed.s), New York: McGraw-Hill, 1963, pp. 406-450.

5. Mitchell, T.M., Utgoff, P.E., and Banerji, R., *Learning by Experimentation: Acquiring and Modifying Problem-Solving Heuristics*, **Machine Learning**, R.S. Michalski, J.G. Carbonell, and T.M. Mitchell (Ed.s), Palo Alto, CA: Tioga, 1983, pp. 163-190.

6. St. Clair, D. C., Bond, W. E., Flachsbart, B. B., and Vigland, A. R., *An Architecture for Adaptive Learning in Rule-Based Diagnostic Expert Systems*, **1987 Proceedings of the Fall Joint Computer Conference**, IEEE Computer Society, October 1987.

7. Shapiro, E., *An Algorithm That Infers Theories From Facts*, **Proceedings of the Seventh International Joint Conference on Artificial Intelligence**, Los Altos, CA: William Kaufmann, Inc., 1981, pp. 446-451.

N88-16363

TEMPORAL AND CONTEXTUAL KNOWLEDGE IN MODEL-BASED EXPERT SYSTEMS: Ford Aerospace's Paragon Project

Tihamer Toth-Fejel and Dennis Heher
Ford Aerospace and Communications Corporation
Sunnyvale, California 94089-1198

ABSTRACT: This paper introduces Paragon, a general-purpose environment for building model-based expert systems. The focus is on contextual and temporal representation, with time considered a type of context.

Conceptually, a Paragon knowledge base is a highly constrained semantic network. Its interfaces enable the domain expert to make knowledge explicit, prevent the proliferation (and potential contradiction) of preconditions found in rule-based systems, and make the knowledge base highly partitionable for parallel processing. Paragon automatically generates LISP code from the knowledge base. Executing this code provides a simulation that allows the domain expert to observe the explicitly described behavior and verify the validity of the knowledge base.

Paragon has been demonstrated in domains that are understood well enough to be modeled, such as satellite diagnostics, ground station diagnostics, and satcom network monitoring.

INTRODUCTION: Temporal and contextual representation in expert systems is a difficult area in Artificial Intelligence (AI). Most expert systems consider time as a special type of context, as does Paragon. In rule-based systems, context is represented by the premise of each rule. Since the context in which a rule takes effect is globally referenced, the premises of rules become longer as the knowledge base gets larger. Partitioning rules into contextually similar sets only delays the inevitable.

At Ford Aerospace, when preliminary calculations showed that the domain of satellite diagnostics would require more than 100,000 rules, the researchers sought another technique [4][7]. The model-based semantic network approach, exemplified in the Calisto project [6], seemed promising for making large problems tractable.

GOALS: The satellite diagnostics domain is well understood; however, domain experts are still expensive and difficult to obtain. Therefore, it is desirable to train them in a minimal amount of time (one week) and immediately place them in front of a workstation running Paragon. Then the domain expert should describe a satellite's components, the causal and compositional relationships between them, and their behavior in terms of states, transitions, and events. The conceptual representation described should exhibit cognitive resonance. In other words, not only should Paragon be user-friendly, but it should also represent the high-level concepts and relations that the domain expert uses when thinking. Finally, Paragon should manipulate these concepts and relations in the same way as the expert does, thus accomplishing the intended tasks of fault diagnosis, analysis, correction, and planning in well-understood domains. As in most AI problem areas, everything hinges on knowledge representation.

PARAGON REPRESENTATION: The Paragon representation can be defined at five layers [2], as outlined by Ferguson [3].

At the **application layer**, Paragon has been demonstrated in many domains: satellite diagnostics, satellite network monitoring, and ground station diagnostics, with expected demonstration of planning and pattern recognition in the satellite domain. At this layer, Paragon enables the domain expert to specify (ie. create, name, and link) domain-specific knowledge about batteries, heaters, their thermal and electrical relationships, commands, procedures, and other physical and non-physical ideas. The domain expert never sees LISP code, and touches the keyboard only when assigning names.

At the **conceptual layer**, Paragon provides primitives such as Concepts (semantic net nodes), and Relations (semantic net links). Central to the issue of knowledge acquisition,

PRECEDING PAGE BLANK NOT FILMED

Paragon makes it possible for the domain expert to create user-defined, but Paragon-constrained, Concepts and Relations.

At the **epistemological layer**, Paragon provides six types of concepts that can be named and linked to other concepts (Figure 1 illustrates the examples):

1) Primitive Concepts (Primitives) represent instantiations of physical and nonphysical objects in the real world (ie. Battery1, Switch2). They gather Attribute Concepts together under a single name, and are the primary focus of behavior, causal relations, and compositional relations.

2) Definition Concepts (Classes) are generic covering sets of other Primitives or Classes, and allow the grouping of concepts by classification (ie. Battery, Heater).

3) Attribute Concepts (Attributes) describe properties of Primitives and Classes (ie. Voltage) and contain the name, type, and value of those properties.

4) Composite Concepts (Composites) are compositional groupings of concepts (ie. Composite Heater1 "has parts" HeatingElement and Switch2).

5) Event Concepts (Events) describe changes to Attribute values of a Primitive while in a particular state, and are equational in nature (for example, the Voltage of Battery1 is divided by the Resistance of Heater1 => the Current of Battery1.

6) State Concepts (States) collect the Events that occur while a Primitive is in a particular state (ie. Charge or Discharge).

Relations (the instances of which are also called links) contain a minimal amount of information and can be of four main types:

1) Specialization (or Classification) Relations describe the relations between classes and their subclasses or members. For example, there exists a Specialization Relation between Battery and Battery1. Inheritance of States, Events, and Attributes can occur in both directions along this link, saving considerable time in knowledge acquisition.

2) Compositional Relations describe the relations between concepts and their composites and/or components. One example was given in describing Composite Concepts.

3) Causal Relations describe how Primitives affect each other. For example, Battery1 POWERS HeatingElement1. Every Causal Relation is defined by the domain expert.

4) Transition Relations (Transitions) describe conditions under which "control" of a Primitive can switch between States, and contains a LISP expression which evaluates to true or false.

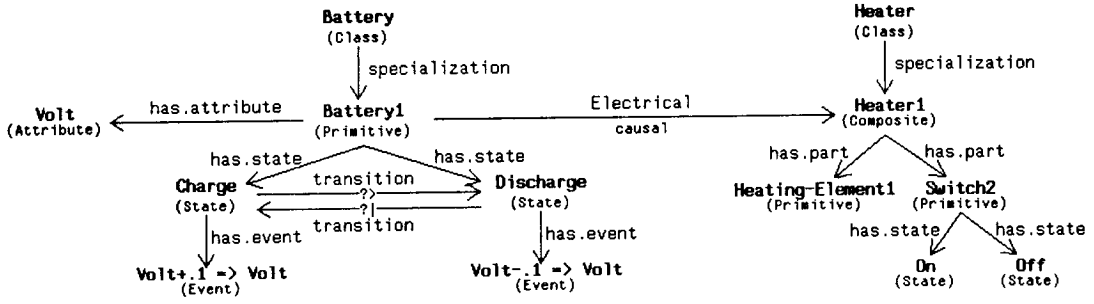


Figure 1. Paragon Knowledge Base

One of the most important aspects of Paragon with respect to contextual representation is that it enforces the principle of locality. This principle constrains access to information by requiring an explicit relation to be defined between concepts [3]. An Event can affect only one local Attribute, and while it can access all the other Attributes of the Event's Primitive, it can only access external attributes through explicit causal relations that are associated with the Event's Primitive. Transitions are constrained by the same principle.

The **logical layer** can be interpreted as describing how Paragon handles the ANDing and ORing of multiple relations and concepts. For example, if two concepts are electrically

connected by a number of digital electrical lines, then the domain expert can model it either with a single electrical relation, or many.

At the **implementation layer**, LISP hash arrays were used to represent both concepts and relations, while all Paragon interface and core functionality was implemented in LISP. For reasons of practicality, additional liberties were taken to compromise the "pure" Paragon representation with frame-based constructs.

After the domain expert finishes describing each piece of the domain, Paragon generates a simulation model in LISP source code. This code can then be executed, and its behavior observed via graphic active images and values. The model can be tested as if it was a piece of physical hardware, and if it demonstrates the same input/output functionality as a good device, then the domain expert knows that the knowledge base is correct [8]. This is a bold but unsupported claim. The developers of Paragon have not been able to mathematically prove truth preservation or logical correctness. However, humans seem to do intellectual tasks quite well without such rigorous proofs, and even computer programs have been quite useful without strict logical foundations. In the experience of Paragon users, a mistake in the description can appear either as wildly oscillating behavior or as no behavior at all. In other cases, the description error and resulting mis-modeling is more subtle, just as in physical hardware.

Faults are not modeled in Paragon, because such modeling would be self-defeating. This is because a device can fail in many more ways than it can work properly. The lack of fault models has not proven to be a handicap, because Paragon reasoning modules can pinpoint faults to whatever level the domain has been modeled. Current research on causal analysis and planning indicate that the Paragon representation is ideal for determining the cause of failures, and possibly even for finding recovery procedures, though some very high level of fault representation may turn out to be useful.

CONTEXTUAL REPRESENTATION: In a Paragon Knowledge Base, every object of the target domain is represented by a Primitive. This Primitive may have many States, with different Events changing their corresponding Attribute values differently in each particular State. In the example illustrated in figure 1, Battery1 has States Discharge and Charge, each containing an Event which modifies the value of Volt. The Process Definition Interface allows the domain expert to specify States and Transitions between States. There are two Transitions connecting Discharge and Charge, illustrated by ?> (conditionally true) and ?| (conditionally false).

To keep the simulator from endlessly looping through states via these transitions, the domain expert must specify the conditions (or context) under which the transition will occur, assuming that the Primitive is currently in the "from" state of that particular transition link. In many ways, this condition is similar to the premise of a rule, and caution must be exercised to prevent the proliferation of precondition clauses (that define a context) and the "ad-hoc-ness" that results from too much flexibility.

The purpose of the Context Specification Interface (CONTXSPEC) is to address the above problem by enforcing the principle of locality. At the implementation level, CONTXSPEC outputs a condition -- a piece of LISP code that evaluates to either true or false. On the input side, CONTXSPEC must satisfy the same four criterion that any representation of reality must provide [6]:

1) Completeness - represent all relevant and necessary knowledge in the domain.
2) Precision - provide appropriate granularity of knowledge.
3) Clarity - lack ambiguity in interpretation.
4) Cognitive Resonance - use the same concepts the domain expert does.

CONTXSPEC satisfies the criterion of completeness by providing the domain expert with relational operators (=, <, >, etc) and local Primitive Attributes (Voltage of Battery1, Temperature of HeaterA). Only Attributes of the States' Primitive and those passed in by

causal links can be accessed, in accordance with the principle of locality. This principle of locality limits communication between concepts by requiring all cause/effect relationships to be specified. While very frustrating at times, this strict limitation engenders a number of significant advantages:

1) When the domain expert wants to access a non-local Attribute, but CONTXSPEC doesn't provide that access, then he or she will realize that a causal relation has not yet been made explicit.

2) Prevents the proliferation of preconditions by tying the context to the location of the nodes and links within the semantic net.

3) Makes each transition self-contained with no side-effects, a property that makes the Paragon KB ideally suited for parallel processing.

4) Finally, the principle of locality restricts the impossibly large number of attributes CONTXSPEC must otherwise display to a manageable number.

By mousing four times, the domain expert can specify a simple condition such as "The Voltage of SolarArray1 is > 10", while CONTXSPEC prevents low-level errors such as mistyping or selecting the wrong menu. Most of the conditions in Transitions are this simple, though much more complicated ones are possible with logical operators AND, OR, and/or NOT. Most of the complexity ends up in the state graph, with its multiplicity of states and possible transitions. If the state diagram is too complicated, this is an indication that the concept should be broken down into its components, or that some states may be merged, unfortunately at a loss of modeling detail. Internally, this condition is represented by a case-grammar-like sentence, which is very easy to translate to LISP, establishing a condition in a Transition between two States. This condition can be true or false, depending on the overall state of the world, or in this example, the value of the Attribute Voltage of the Primitive SolarArray1. The evaluation of Events during the execution of simulation code causes Attributes' values to change.

Originally, Paragon gave the domain experts the capability to represent complex equations inside the Transition's conditions. Unfortunately, equations hide large amounts of implicit knowledge. Therefore Paragon was changed to prevent the domain experts from entering equations anywhere except in the Events. This increased difficulty in knowledge acquisition was caused by the necessity to break up complicated fomulas into representations more amenable to automated reasoning.

TEMPORAL REPRESENTATION: Reasoning about time is presently a very debated issue in AI. However, some ideas are generally agreed upon, for example Allen's relations on convex time intervals [1]. Early in Paragon's development, sets of algebraic mappings were found between any two intervals to produce a third interval (ie. Plus, Minus, Cross-Product, etc). With a set of relations and mappings, it was assumed that a useful algebra could be integrated into Paragon. The Temporal Specification Interface (TSI) was developed to quickly specify complicated temporal expressions. Unfortunately, a non-trivial algebraic group for time intervals was not found; fortunately, this lack of success didn't matter. During the development of TSI and the search for a temporal algebraic group, the domain experts used Paragon concepts to simulate clocks and timers whenever they needed them. Upon closer examination, it was found that Allen's relations could be implemented in Paragon at the Attribute-State-Event level, so TSI was never integrated into Paragon. In addition, Paragon has the ability to do multi-interval comparisons, which Ladkin showed to be infeasible when representing at the interval-relation level [5].

It turns out to be quite simple to set up very complicated timed, conditional or asynchronous cycles (or non-cyclic temporal state changes) in the state transition graph of a Primitive. For example, to simulate the sun-shade cycles that a solar array experiences in orbit, the Primitive SolarArray1 has an Attribute Clock, which is reset in States StartShade and EndShade by identical Events (0 => Clock) and set by (Clock+1 => Clock) in States Sun and Shade. The description of SolarArray1's behavior is completed by specifiying theTransitions to become true when Clock values are equal to 51 and 39 respectively.

As in most rule-based systems, temporal representation is a special case of context in Paragon. This is because at a certain level of granularity and ignorance, time (and context) can be considered a cause. In a rule, the premise can be thought of as causing the consequent. This paradigm may lead to the nonsensical belief that Monday causes Tuesday. Well, not exactly. The events that occur during Monday (like the passage of time) cause the Transition between Monday and Tuesday to become true, allowing the change of state. Given ignorance of the structure of the solar system and of the naming convention regarding 24-hour periods, it is perfectly acceptable for automated causal diagnosis to conclude that Monday causes Tuesday.

CONCLUSION: This paper presents a basic paradigm that allows representation of physical systems, with a focus on context and time. Paragon provides the capability to quickly capture an expert's knowledge and represent that knowledge in a cognitively resonant manner. From that description, Paragon creates a simulation model in LISP, which when executed, verifies that the domain expert did not make any mistakes. The Achilles heel of rule-based systems has been the lack of a systematic methodology for testing, and Paragon's developers are certain that the model-based approach overcomes that problem. The reason this testing is now possible is that software, which is very difficult to test, has in essence been transformed into hardware.

REFERENCES

[1] Allen, J., Maintaining Knowledge about Temporal Intervals, **Readings in Knowledge Representation**, edited by Brachman, R., and Levesque, H., Morgan Kaufmann, 1985, pp. 510-521.

[2] Brachman, R., On the epistemological Status of Semantic Networks, **Readings in Knowledge Representation**, edited by Brachman, R., and Levesque, H., Morgan Kaufmann, 1985, pp. 191-215.

[3] Ferguson, J. C., Beyond Rules: The Next Generation of Expert Systems, **Proceedings of the Air Force Workshop on AI Applications for Integrated Diagnostics**, May 1987.

[4] J. Ferguson, R. Siemens and R. Wagner, Starplan: A Satellite Anomaly Resolution and Planning System, from J. Kowalik, ed., **Coupling Symbolic and Numerical Computing in Expert Systems**, pp. 273-281.

[5] Ladkin, P., Time Representation: A Taxonomy of Interval Relations, **AAAI-86 Proceedings**, August 11-15, 1986, pp. 360-366.

[6] Sathi, A., Fox, M., and Greenberg, M., Representation of Activity Knowledge for Project Management, **IEEE Transactions on Pattern Analysis and Machine Intelligence**, Vol. PAMI-7, No. 5, September, pp. 531-552.

[7] Siemens,R., Golden, M., and Ferguson, J. C., "Starplan II: Evolution of an Expert System", **AAAI 86 Proceedings**, pp. 844-850.

[8] Toth-Fejel, T.T., **Self-Test: From Simple Circuits to Self-Replicating Automata**, Master's Thesis, University of Notre Dame, 1984, pp. 160-161.

N88-16364

METHODOLOGY FOR TESTING AND VALIDATING KNOWLEDGE BASES

C. Krishnamurthy, S. Padalkar, and J. Sztipanovits

Center for Intelligent Systems

Vanderbilt University, Nashville, Tennessee

B.R. Purves

Boeing Aerospace Company, Huntsville Alabama

ABSTRACT

The paper describes a test and validation toolset developed for artificial intelligence programs. The basic premises of this method are: (1) knowledge bases have a strongly declarative character and represent mostly structural information about different domains, (2) the conditions for integrity, consistency and correctness can be transformed to structural properties of knowledge bases and (3) structural information and structural properties can be uniformly represented by graphs and checked by graph algorithms. The interactive test and validation environment have been implemented on a SUN workstation.

PRECEDING PAGE BLANK NOT FILMED

INTRODUCTION

Testing and validation is the ultimate precondition for the application of artificial intelligence (AI) technology in space systems. In spite of its obvious significance, testing and validation have been a neglected topic in AI research. The results being reported are quite contradictory. Some authors have pointed out that certain knowledge-based systems, such as expert systems, are inherently untestable and unreliable, while others argue that software validation is easier for knowledge-based systems than for conventional programs.

The first section of this paper summarizes our results in the evaluation of AI technology from the aspect of software engineering. An important conclusion of this analysis is that clear separation between AI systems (expert systems, natural language systems, etc.) and AI techniques (declarative programming, symbolic programming, etc.) is necessary. It has been shown, that the well-known difficulties in testing and validation are inherent nature of the functionality of specific AI systems and do not stem from the implementation technology. Most importantly, the basic AI techniques offer new opportunities in software testing and validation, which can dramatically improve the test technology of complex software systems.

The second section of the paper describes a test and validation toolset developed for AI programming. The basic thrusts of the selected methodology are: (1) knowledge bases have strongly declarative character and represent mostly structural information about different domains, (2) the conditions for integrity, consistency and correctness can be transformed to structural properties of knowledge bases and (3) structural information and structural properties can be uniformly represented by graphs and checked by graph algorithms.

An interactive test and validation environment has been implemented on SUN workstation. The knowledge representation paradigms for which test and validation methods have been developed include: rule-based systems and object-oriented programming. The application of the methodology is presented for testing structural properties of object-oriented programs.

BACKGROUND

The problems of testing and validation can be examined only in the context of the system to be tested and validated. Therefore, clear distinction must be made between systems that are built using AI and the techniques developed and used in AI programming.

1. AI Systems and AI Techniques

One of the widely accepted, generic objectives of AI is to construct intelligent agents (Newell, 1982). Intelligent agents can operate autonomously in a task environment, are able to recognize their situation by means of the perceptual components, and are able to plan their actions according to a goal structure by means of their general knowledge. These capabilities are also manifestations of human intelligence, i.e., the primary objective of AI systems is to mimic human intelligence.

ORIGINAL PAGE IS
OF POOR QUALITY

The AI systems which have received the largest publicity in recent years are expert systems. Their primary purpose is to represent human knowledge symbolically and "operate" on the knowledge by using automated reasoning methods. Some of the most important aspects of expert systems that have attracted considerable attention are:

- ability to capture rare and expensive human expertise and make it available,
- ability to reliably operate in fuzzy, unexpected situations,
- ability to implement heuristics,
- ability to explain actions for users.

While seeking a better understanding of human intelligence and implementing systems that exhibit "intelligent" behavior, research in AI has discovered a number of novel software techniques and tools. These techniques and tools have proven to be extremely useful in a number of application domains struggling with construction of highly complex systems. More importantly, AI techniques have provided methods to use computers for symbolic, qualitative "computations," which have the immediate potential for building new generations of application systems in areas such as instrumentation and process control. The approach, which focuses primarily on AI techniques and not so much on the scientific objectives of AI, (i.e., understanding and imitation of human intelligence) is often referred to as AI engineering (Allmendinger, 1986).

It would be difficult to enumerate all of the new software techniques originated and elaborated by AI research. Here we discuss only declarative programming, which is widely used in the implementation of intelligent systems.

Conventional programming is essentially imperative, i.e., programs describe the sequence of steps that are necessary for solving a particular problem. We may state that imperative programs primarily represent "how to" knowledge. In imperative programming the programmer is responsible for transforming the problem definition ("what to") into its solution of imperative style.

Declarative programs describe the declarations of problems rather than their solution. The basic technique used in declarative programming is to build "smart" interpreters that can transform the declarations into "how to" knowledge. The key components of declarative programming are (1) the problem-specific representation language, which is used for describing the problem and (2) the corresponding interpreter.

Well-known programming paradigms that are strongly declarative are:

- logic programming, where programming occurs in the form of declaring objects and their relations, (a well known example of logic programming languages is, of course, Prolog),
- rule-based programming, where the knowledge is expressed primarily in rule format (e.g., ART, KEE, etc.),
- constraint-based programming, which includes the declaration of objects (e.g., variables) and the constraints (e.g., arithmetic

constraints) among them.

Declarative programming is widely used in constructing knowledge-based systems. The "knowledge base" is usually the declarative component while the interpreter is the procedural component of these systems (e.g., the rule base is the knowledge base, the inference engine is the interpreter in the case of rule-based expert systems).

2. Testability in AI Programming

Whether we approach AI programming from the side of specific AI systems (e.g., expert systems) or from the side of AI programming techniques (e.g., declarative programming), we can identify significantly different views concerning testing and validation.

From a functional point of view, expert systems try to mimic human expertise. The basic conceptual and practical problems stemming from this fact are clearly described by Lane, 1986.

a. Testing requires design specifications. Lane's observation is that specifications for expert systems, against which system performance can be evaluated "are almost universally lacking in current expert system developments." The probable reason is that though the concept of expertise is intuitively clear, it is impossible to give a unique specification for it (at least presently or in the immediate future). Obviously, the "rule-set" of rule-based expert systems can be considered only as a "model" of expertise, rather than its specification. He suggests the development of new methods for setting design requirements and system specifications that should be based on an improved understanding of the roles of expert systems in complex systems.

b. Performance is dependent on the scenario. A well-known problem of current and near-future expert systems is that their performance degrades dramatically at the "boundary of their knowledge base." Contrary to human experts, expert systems are unable to detect their limits so as to avoid catastrophic failures and to degrade gracefully in new or marginal conditions. Lane points out that except in the relatively simple cases, when the "expert system" is actually the implementation of a well-defined decision tree, the performance evaluation of expert systems has an "inherent dilemma." A possible method of testing is to sample the scenarios and conditions, and evaluate the system performance in specific situations. This method can fail to detect even potentially catastrophic outcomes. The other alternative is systematic enumeration of all possible input conditions, which is unrealistic in most cases due to time and cost.

Test approaches can help in the development of expert systems, but cannot resolve the problems mentioned above (Gashing et al., 1983).

The declarative character of the knowledge bases offers new opportunities for testing some of their structural and logical features. Validation methods are presented in Stachowitz et al., 1987; Nguyen, 1987; and Suwa et al., 1982; for checking inconsis-

tency, completeness, redundancy, etc., of rule bases. It should be mentioned that these tests cannot guarantee functional correctness but can offer significant help in detecting potential problems.

GENERIC TEST AND VALIDATION METHODOLOGY FOR KNOWLEDGE BASES

The basic thrust of our methodology is that the primary implementation technique for knowledge-based systems is declarative programming. As we have previously discussed, declarative programming includes three different program components, which are:

- interpreter,
- typically small imperative components, and
- declarations.

The interpreter and the imperative components are basically conventional programs that can be tested and evaluated by using well elaborated software engineering methods and techniques. In this sense, testing and evaluation of declarative programs does not differ from that of the conventional programs. The major difference is that the complexity of declarative programs is mostly concentrated in the declarations constituting the "knowledge base" of the system to be tested. Below we summarize some of the new opportunities emerging for testing and validation of declarative programs.

1. Automatic Proof of Correctness

Declarative programs are typically symbolic representations of structures. It is possible to implement automatic reasoning processes that can prove various properties of the structures represented. Requirements, such as:

"The fan-out must be less than or equal to 20," or
"Two active outputs cannot be connected"

can be easily checked on the declarative representation of a digital circuit simulator program. In other words, the functional correctness of the simulator can be tested by using automatic, high-level tools.

2. Mathematical Modelling

The structure of declarative programs can be mapped into graphs and different structural properties can be checked by using graph algorithms. E.g., causal networks which are used in failure mode and effect analysis can be tested for cycles; physical structures can be tested for connectivity; signal-flow structures can be tested for loops, etc. Graph algorithms can be used for testing the equivalence of different declarative programs, which is a unique possibility. (Proving the equivalence of imperative programs is an extremely complicated problem.)

3. Graphic Tools

Since declarative programs typically represent structures, they can be represented by graphic tools, and can be synthesized by inter-

active graphic editors.

Although, these opportunities have been recognized and exploited in some of the test and validation techniques mentioned before, their common feature is that the actual implementation is closely coupled to a particular knowledge-based system and knowledge representation language (Stachowitz, 1987).

Our goal was the development of a generic methodology and programming environment which effectively supports the testing and validation of different kinds of knowledge-based systems. The rationale behind this goal is the recognition that knowledge-based systems include multiple knowledge bases and are described in different representation languages.

The generic test and validation method can be summarized as follows.

Let us suppose, that L is the representation language and P is a set of declarations written in L. The general steps of validating the knowledge base are the following:

Specification of test criteria. By analyzing the specific nature of the knowledge base, a relevant set of test criteria [c(1), c(2),...,c(n)] has to be defined. The individual test criteria should be assertions on the structural properties of the knowledge base.

Specification of mapping rules. Depending on the semantics and syntactics of L, and the way the test criteria can be expressed as abstract graph properties, mapping rules (M) are defined. The rules maps P into a labelled, directed graph M(P)->G(V,E). The labels of the vertices and edges of the graph: v[a(1), a(2),...,a(n)] and e[a(1),a(2),...a(j)] are attributes that are extracted from P and associate the nodes and edges with its semantic entities.

Specification of user interface. The actual test proceeds by mapping the knowledge base (or certain sections of the knowledge base) into graphs and checking the test criteria by running graph algorithms. The results of the tests are presented by using a knowledge base specific graphic interface.

STRUCTURE OF THE TEST AND VALIDATION ENVIRONMENT

The methodology described above makes it possible for the design of a test and validation environment (TVE) where the common components are clearly separated from those which are unique to specific knowledge bases. The ultimate benefit of this separation is that the system can be easily adapted to different problems and representation languages and can provide a unified environment for testing and validating knowledge bases.

The structure of the TVE can be seen in Figure 1. The MAPPER accepts the knowledge base to be tested from the user and maps it into a graph. The ANALYZER runs a set of graph algorithms and outputs the results to the user. The analysis process is interactive and supported by graphics. The ANALYZER KERNEL constitutes the common part of TVE. It provides a set of

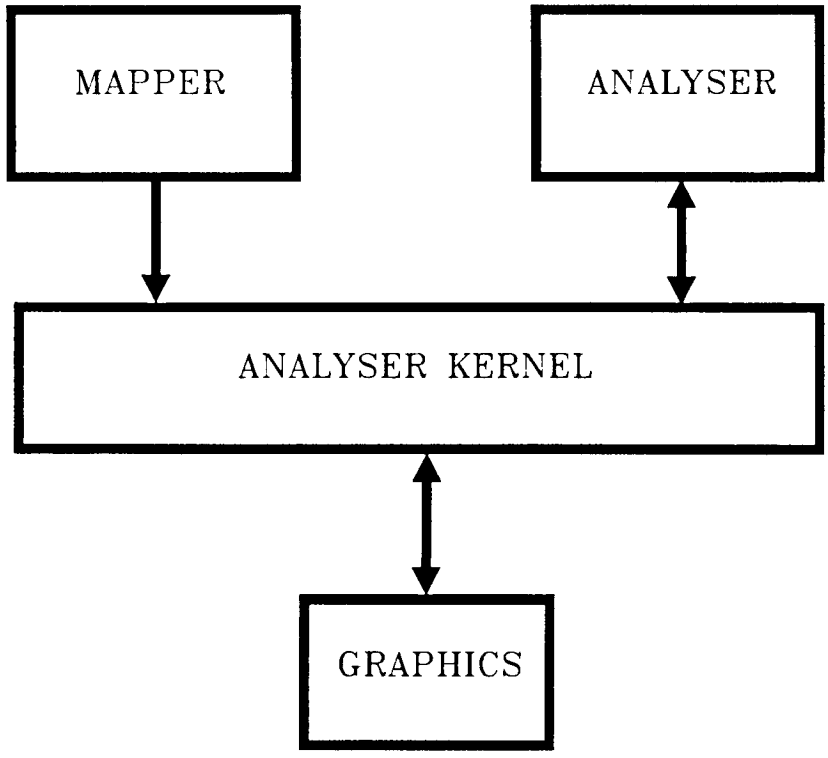


Figure 1: Functional Structure

services to build, represent and analyze graphs. The summary of the interfaces of the analyzer kernel can be seen in Tables 1, 2, and 3.

The Mapper Interface includes two sets of calls. One of them is used to parse the input source file which contains the knowledge base. The other set is used to create and modify graphs. The Analyzer Interface provides access to the library of graph algorithms which are the basic building blocks for implementing test and verification procedures.

The selection of graph algorithms is continuously expanded as new testing and validation methods are developed for different knowledge bases.

The third group of kernel calls facilitates generation of user interfaces. In order to help the user in navigating through complex structures and in analyzing structural properties, extensive color graphics are used with a sophisticated window system. The services provided by the graphics interface are summarized in Table 3. The interactive graphics interface makes it possible (1) to represent the entire graph, (2) to zoom into certain areas, (3) to select nodes and edges by using a pointing device and to display the corresponding semantic entity of the knowledge base in a text window, and (4) to start various analysis processes through a hierarchically organized menu interface.

IMPLEMENTATION

TVE has been implemented on a SUN 3/110 workstation by using the Sunview graphics package. The system is decomposed into two communicating processes (see Figure 2). The Analyzer and Mapper functions run as a LISP process. The appropriate kernel interface functions are written in C and are embedded in the LISP environment. The advantage of this solution is that the knowledge base specific components of the Analyzer and Mapper can be more conveniently implemented in LISP than in other available languages.

The graphics interface runs as a separate graphics process which communicates with the LISP process through UNIX pipes. After receiving a user command, it is decoded and the appropriate function call is sent to the LISP process to service the request.

Separation of the graphics interface from the other components of the system ensures the portability of TVE to other workstations, with different graphics capabilities.

APPLICATION EXAMPLE: TESTING AND VALIDATION OF OBJECT-ORIENTED SYSTEMS

Object-oriented programming has the virtue that hierarchical system declarations and properties, such as structural and functional inheritance, can map quite naturally into this programming methodology.

Typically, most of the useful object-oriented systems tend to become very large and, after a point, manual structural testing becomes extremely difficult, if not impossible. The TVE provides an automated, interactive test environment, with extensive graphics support, for the structural

Table 1. Mapper Interface

FUNCTION	PROCEDURE CALLS	DESCRIPTION
Parse input	[Internal set of macros specific to the representation language]	Builds symbol tables, stores text information
Create graph	create-node(attributes) create-edge(attributes)	creates a list of nodes edges, and graph adjency lists

Table 2. Analyzer Interface

FUNCTION	PROCEDURE CALLS	DESCRIPTION
Detect cycles	cycles (graph)	finds node-chains which form cycles in the graph
Find connected components	find-connected-components (graph)	finds a spanning forest for the graph
Find nodes matching certain attributes	find-group (graph attributes)	partitions the graph based on specific attributes of nodes or edges
Describe node	display-node (node)	displays all attributes of a node
Access nodes	gen-lower-tree(node) gen-upper-tree(node)	generates sub-tree rooted at this node

Table 3. Graphics Interface

FUNCTION	PROCEDURE CALLS	DESCRIPTION
Menu-based input	[executive calls]	converts analysis requests from graphics process into analyzer function calls
Graph layout generators	hierarchy (graph root), bipartite (graph) tree (graph root)	draws nodes and edges on screen
Highlight sections of graph	highlight (path graph) [executive calls]	highlights cycles, displays text, zooms on sections of the graph

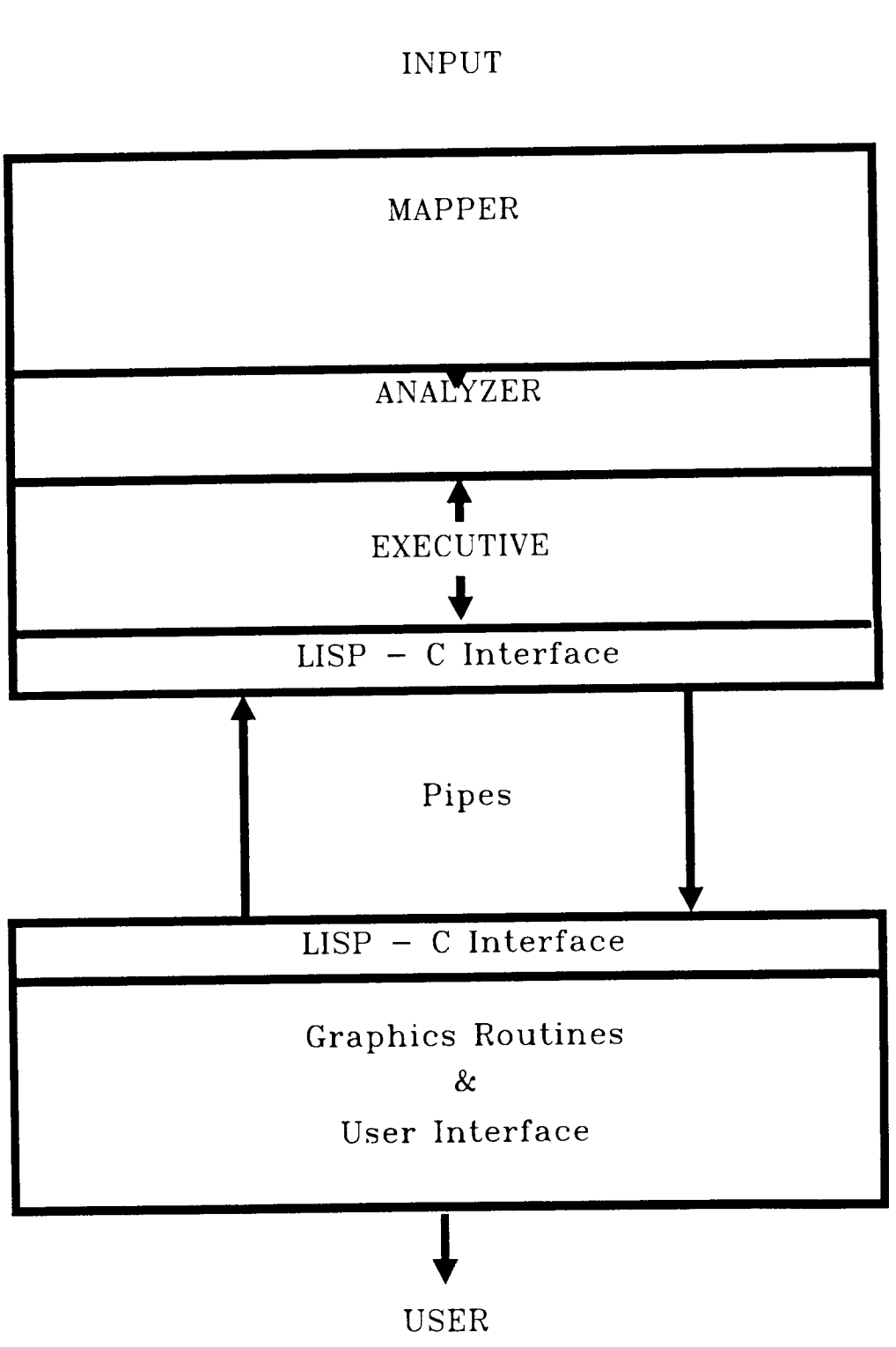


Figure 2 : Structure of Implementation

testing of large object-oriented systems.

Currently the facilities provided by the TVE are:

1. Generating the inheritance hierarchy for the entire system.

2. Generation of the inheritance tree for specific object classes in the system.

3. Detection and highlighting of cyclic inheritance of object classes.

4. Detection of missing class and method definitions.

5. Detection of conflicting method definitions (i.e., an object inherits methods of the same name from two different classes, but these two are in no way connected, i.e., they lie on two different paths in the inheritance tree.)

The sequence of actions performed is as follows:

The MAPPER accepts the object-oriented system written in a particular object-oriented programming language as input, and maps it into a graph.

Each object class in the system is mapped onto a node in the graph, and edges are defined as follows:

If an object class A inherits(includes) the class definition of object class B , then there is an edge from the node representing class A to the node representing class B. With this simple algorithm the entire graph is built. The current system implements a mapper for a Flavors-like object-oriented system, and uses the same algorithm as Flavors to determine method inheritance. The difference is that all information is explicitly displayed to the knowledge engineer, before expensive dynamic testing takes place.

For example, in Flavors, cyclic dependencies of objects are avoided, but the knowledge engineer is not notified. In the TVE all cycles are explicitly displayed.

On building the entire graph, the mapper terminates, and control passes to the executive which creates and communicates with the graphics server.

The graphics server, on creation, generates a window for the user interface. This window consists of a panel of test options, and a large canvas for displaying the graph generated for the system. An additional text sub-window is created for display of textual information about the system (e.g., object definition or list of inherited methods).

The user can now select any test option by simply selecting that option from the panel with a pointing device. This selection is communicated to the executive who, in turn, invokes analyzer routines to carry out the test. Information about any object in the system is obtained simply by pointing at the corresponding node in the graph.

TVE has also been used for supporting the static analysis of large

rule-based systems. Specifically, it has been successfully tried on a rule base containing approximately one hundred OPS5 rules.

CONCLUSIONS

The purpose of this paper was to discuss some of the software engineering aspects of AI programming and to describe a method and corresponding tools developed for testing and validating knowledge bases. The essence of the method is that the criteria for correctness is expressed in the form of structural properties and checked by using various graph algorithms.

The conclusion of our analysis was that the result of the evaluation depends on the approach to AI programming. Testing and validation of certain AI systems which try to mimic manifestations of human intelligence (e.g., expert systems) may be quite problematic because of the inherent difficulties in specification and performance evaluation. On the other side, programming techniques which are generally used in AI programming (e.g., declarative programming, symbolic programming, etc.) offer new opportunities for testing and validating the "knowledge base" of complex systems. These opportunities serve as one of the main incentives to use AI programming techniques in the design and implementation of complex systems.

This conclusion is quite contradictory to the often emphasized view, that AI techniques are "unsafe" compared to conventional programming techniques. The fundamental feature of knowledge-based systems is that most of the complexity is concentrated in their knowledge base. The dominantly declarative character of knowledge bases allows the application of automatic testing and validation techniques that can significantly improve the safety and reliability of large software systems.

REFERENCES

Allmendinger, G., "AI: Can Performance Match the Promise?," InTech, pp. 45-50, April, 1986.

Gashing, J., et al., "Evaluation of Expert Systems," in Building Expert Systems, F. Hayes-Roth, D.A. Waterman, and D.B. Lenat, eds., Addison Wesley, 1983.

Lane, N.E., "Global Issues in Evaluation of Expert Systems," Proc. 1986 International Conference SMC, pp. 121-125, 1986.

Newell, A., "The Knowledge Level," Artificial Intelligence 1:87-127, 1982.

Nguyen, T.A., "Verifying Consistency of Production Systems," Proc. of The Third Conference on AI Applications, pp. 4-8, 1987.

Stachowitz, R.A., et al., "Validation of Knowledge-Based Systems," Second AIAA/NASA/USAF Symposium on Automation, Robotics and Advanced Computing for the National Space Program, 1987.

Suwa, M., et al., "An Approach to Verifying Completeness and Consistency

N88-16365

CLIPS AS A KNOWLEDGE BASED LANGUAGE

JAMES B. HARRINGTON
HONEYWELL SPACE AND STRATEGIC AVIONICS DIVISION
CLEARWATER FL 34624-7920

ABSTRACT

CLIPS is a language developed by Johnson Space Center (JSC) for writing expert systems applications on a personal or small computer. The CLIPS language was written in the C programming language and JSC made provisions to call CLIPS from, or embed CLIPS within, a control or applications program. This paper will look at some of the salient characteristics of a knowledge based system (KBS). The capabilities of CLIPS will be discussed in light of these characteristics, and the KBS characteristics of CLIPS will be compared with those of LISP, Prolog, and OPS5.

INTRODUCTION

The intent of this paper is to describe the CLIPS programming language and compare it to three other artificial intelligence (AI) languages (LISP, Prolog, and OPS5) with regard to the processing they provide for the implementation of a KBS. The paper will conclude with a discretion of how CLIPS would be used in a control system. The definition of many of the commonly used terms in the field of AI languages will be found in this paper.

PROGRAMMING LANGUAGES

Several languages have been developed to enhance the building of KBS by providing a direct method of encoding both data and procedural knowledge (procedural knowledge is the knowledge of how to act on the data). The major requirement for a language to be used for developing a KBS is that it handle strings of characters or "symbols" as well as numbers. For the above reason Pascal and C are more favored, among the "standard" programming languages, for developing expert systems than is FORTRAN or assembly language. Several languages have been developed specifically to enhance the capability to deal with symbols; of these languages, this paper will deal with only LISP, Prolog, OPS5, and CLIPS.

The most common language for developing AI applications is LISP. LISP stands for *LIS*t *P*rocessing language. It was based on John McCarthy's work on nonnumeric computation published in 1960. LISP itself does not have any constructs that provide for explicit encoding of data and procedural knowledge, however, LISP is an excellent symbol processing language and provides a rich set of tools that can be used to develop the constructs desirable for a KBS.

Prolog is a relatively new language that has been developed for AI applications. Prolog stands for *Pro*gramming in *Log*ic. It was one of the first attempts to structure a language that would enable a programmer to specify his tasks in logic rather than in conventional programming methods. Prolog was created by Alain Colmerauer and his associates at around 1970.

The name OPS5 stands for *O*fficial *P*roduction *S*ystem, version 5. As one might expect, OPS5 grew out of set of OPS languages. The pilot system developed in OPS5 was "R1" for Digital Equipment Corporation (for the VAX Expert System (ES) configuration tool). C. Forgy and J. McDermott, of Carnegie-Mellon University, were responsible for the development of the OPS5 language.

CLIPS is the most recently developed language of the set to be discussed in this paper. CLIPS is a Forward Chaining rule based system. It is being developed by the Johnson Space Center's AI section, Mission Planning and Analysis Division, as a language suitable for ES development and delivery on conventional computers (ie. the IBM PC, VAX, etc.) and is intended for embedded applications. CLIPS was originally created by Frank Lopez around 1985 and reworked for release to the public by Gary Riley [Culbert 86]. CLIPS is an acronym for *C* *L*anguage *I*ntegrated *P*roduction *S*ystem.

KBS CONCEPTS

A KBS is a program or system that uses a base of knowledge to determine the program output. A KBS language is one that enhances the capabilities of combining data and production rules to obtain a meaningful output. The following sections describe some of the main concepts or characteristics of a KBS.

KBS vs Conventional Languages

A KBS language could be described as a language based on a set of rules that act like functions in a conventional language. These rules are triggered by data (or facts) rather than program flow. All of the facts are examined by the rules on a continuous basis. Hence the KBS code need not execute in the logical flow that it was written. There are often mechanisms for controlling the flow of rule activation (executing a given rule in a KBS) but in general, if the order that decisions are made can be predetermined, and remain constant regardless of the data, then a conventional programming languages would be a more appropriate selection.

Another important difference in the AI languages and conventional programming languages is the way variables a handled. In the AI languages the variable only has meaning within the particular rule in which it is located, there are no global variables. The only method available for "passing parameters" is by asserting a new fact on the fact list.

AI Facts

A fact can be a single element or a list of elements. Each indivisible element in a fact is called an "atom". One of the main reasons that LISP has became so popular for AI applications is because of its built in capability to work with lists.

Table 1 is a summary of the capabilities of each language to represent facts in the knowledge base. The "Argument Format" column specifies weather the element's value is based on its position in the fact list (positional) or is based on keyword recognition. The "predicate" column refers to the association of an atom within the fact to a header or a name; CLIPS is the only language that does not directly provide the capability of relating atoms to a name or function, however, the programmer can define a structure where certain positions within a fact are keywords and the other positions are variable values.

AI Rules

The executable "code" in a KBS are rules. A rule can be viewed as a special If/Then statement and can be partitioned into two parts. The if-part or logic section of the rule is the part of the rule that looks for matches and relationships among the data. The then-part or action part of the rule is activated only after the conditions in the if-part have been satisfied.

Table 1: Language Implementation of Facts*

Language	Argument Format	Predicate	Argument Name	Argument Value
LISP (list)	positional	first list element	n/a	rest of list
LISP (structure)	keyword	type name	slot name	slot value
Prolog	positional	function	n/a	argument
OPS5	keyword	class name	attribute	value†
CLIPS	positional	user-defined	user defined	value†

† must be atomic

LISP does not have any constructs that directly implement an If/Then type of statement. However, LISP does have the language constructs to build If/Then type rules that could allow multiple patterns to be matched as well as multiple actions to be performed. Prolog, OPS5, and CLIPS each provide for multiple pattern, pattern matching capabilities which are summarized in Table 2. In Table 2, "Conjunction" refers to the logical ANDing of facts. "Disjunction" is the logical ORing of facts. Table 3 is a table of commonly used knowledge base operations.

Table 2: If-part Pattern Matching**

Language Feature	Prolog	OPS5	CLIPS
=, ≠ (equal, not equal)	any term	number, symbol, predicate	number, symbol, function
<, >, . . .(less than, greater than . . .)	any term	number	number
Computed expressions	Yes (if-part only)	No (then-part only)	Yes (both parts)
Type test	atom, number, variable	number, symbol	atom, number
Negation, Conjunction	predicate, argument	predicate, argument	argument, function
Disjunction	predicate, argument	argument	argument, function
Nesting of Conditions	Yes	No	Yes

File I/O

A key part of the KB operations is the capability for interfacing with a "permanent" data base. A permanent data base is usually stored on magnetic disk, thus the capability to interface with a permanent data base relies on file I/O capabilities. Of the four languages, LISP has the most extensive set of I/O capabilities; OPS5 has the smallest set.

CLIPS has two sets of I/O file commands: the first is for saving and retrieving program files; the second is for saving and retrieving facts. The CLIPS facilities for saving rules from the CLIPS environment will save only the rules; there is no top

*Expanded from [Cugini 87], Table 3, p. 20.
**Expanded from [Cugini 87], Table 4, p. 23.

level command (like SAVE) to save the facts in the fact list or any "deffacts" statements (deffacts is a CLIPS construct that allows the user to develop a set of facts). CLIPS will load both rules and facts if the program file is created with an external text editor and the deffacts construct is used. During CLIPS program execution, CLIPS does support reading facts from, and writing facts to, disk files.

Table 3: Operations Of Rules On Facts*

Operation	Number of KB objects	Type of KB objects	Source	Language Statement	Rule-part containing the Statement
LISP:					
add	many	fact, rule	file	load	user-defined
add	one	fact, rule	program	make-x	user-defined
modify	one	fact, rule	program	setf	user-defined
delete	one	fact, rule	program	remove,	user-defined
		fact, rule		remhash,...	user-defined
Prolog:					
add	one	fact	program	implicit†	then
add	one	fact, rule	program	assert	if
delete	one	fact, rule	program	retract	if
delete	many	fact, rule	program	abolish	if
add	many	fact, rule	file	consult	if
replace	many	fact, rule	file	reconsult	if
OPS5					
add	one	fact	program	make	then
modify	one	fact	program	modify	then
delete	one	fact	program	remove	then
add	one	fact	program	build	then
CLIPS:					
add	many	fact	file	read <file>	then
add	many	fact	keyboard	read	then
add	many	fact	program	assert	then
delete	many	fact	program	retrace	then

† does not persist--derived and then discarded.

Table 4 is a summary of the I/O features of the four languages. "I/O language objects" refers to the ability to read objects as elements; each read associates a variable with an atom. "I/O characters" refers to the capability to read the external file a character at a time. "I/O binary" refers to the capability of treating an input as a set of bits where each bit may have a specific meaning. The user can modify the CLIPS source code to provide both character and binary input capabilities. "Line input" is the capability to input a line of data at a time regardless of the number of elements associated with the data line. "Pseudo-I/O" is the capability to treat internal memory as an I/O buffer and manipulate memory using I/O routines. "User control" of the input and output refers to the facilities the user has to control the format of the inputting and outputting of data. "Rename and delete files" refers to the capability to access system file commands form the language environment. The

*Expanded from [Cugini 87], Table 5, p. 25.

newer versions of CLIPS do provide access to the DOS commands but access is system dependent.

Table 4: Files and I/O Features*

Features	LISP	Prolog	OPS5	CLIPS
File types	Sequential and Random	Sequential	Sequential	Sequential
I/O language objects	Yes	Yes	Yes	Yes
I/O characters	Yes	Yes		User enhancable
I/O binary	Yes			User enhancable
Line input	as characters		as language objects	as language objects
Pseudo-I/O	Yes			
User control of input	Many Features	Some		Few
User control of output	Many Features	Some	Few	Many Features
Rename and delete files	Yes	Yes		New Versions

Inference Engines

The inference engine is the part of the language that derives the response to a set of facts. It is responsible for selecting which rules will be fired in which order.

The top level of the inference engine is how (or in what order) the facts are processed. Two of the most common terms used to describe the control strategy of rule firing are forward chaining and backward chaining. Forward chaining starts with a set of facts and processes facts with rules until it has reached some conclusion or until there are no more facts to process. Backward chaining starts with a conclusion or assertion of a condition and then checks the facts to determine if the condition can be supported. Backward chaining is extremely useful in any system where the program may be asked why it chose a specific course of action.

Another consideration for control strategy is the selection (or decision) of which rule to fire next. Rule activation relies on "control knowledge." The relationship between rules and control knowledge is that, "rules capture knowledge about *how to transform data*; control knowledge is about *when to transform data*" ([Cugini 87] p. 15, my italics). The commonly used terms for the decision making process are: Depth-first, Breadth-first, Recent-first, Best-first, and Heuristic [Cugini 87].

Prolog is a backward chaining system that processes facts in a depth-first fashion. A depth-first system tries to process as far down one decision path as possible until a block is encountered (in the form of an unsupported fact or improper conclusion). When a block is reached, a depth-first system will back up to the last successful node and proceed down the next alternate path. This process continues from left to right across the "decision tree" until the solution is found or all decision paths are exhausted.

Both OPS5 and CLIPS use forward chaining systems with recent-first fact processing. The recent-first technique does not necessarily progress toward an answer. In a recent-first system, the most recently asserted facts are given more weight so that rules using these facts would be fired first (unless there is some other weighting

*Expanded form [Cugini 87], Table 8, p. 52.

system which might override the rule firing order). Both OPS5 and CLIPS will continue to process facts until each rule has processed each applicable fact.

It is important to note that just because a language was designed around a specific type of inference engine, that language is not locked into that role. There are many cases where Prolog has been used to implement a forward chaining system. Likewise, OPS5 and CLIPS have been used to create backward chaining systems.

User Interface

Any good language will provide tools to aid in debugging of the source code. The most often used tools are: trace features (which will indicate which line of code is being executed), break points (where the number of lines to be executed is specified or an event is specified which will stop execution), and printing out changes in the values of variables. In some languages these tools must be written as part of the source code.

Table 5 is a summary of the user interface features that might be used for debugging a KBS. "Top-level control" refers to the process of invoking or running the KBS. The section on "watch derivation" refers to watching the "thinking" process of the KBS as it moves toward its end point. The "pause and step" features refer to the KBS executing a set number of cycles or instructions. Either before the KBS is executed or during a pause in the execution of a KBS a user may want to "inspect the KBS" (it's facts, it's rules, and the agenda--also known as the conflict list--for rules pending execution). "Manipulate KB" refers to the process of adding, or deleting, facts and rules during a pause in the KBS execution. "Manipulate derivation" refers to controlling the KBS during execution.

Embeddability

Of the four languages discussed, CLIPS is the only language that was designed to be embedded within another system. When CLIPS is purchased, the C source code is also supplied. The CLIPS User's Guide, Reference Manual, and Update notices supply information for customizing CLIPS and embedding CLIPS within other systems. The instructions are written around the Latice C compiler for the IBM PC however there is some information related to using the Lightspeed C compiler on the Macintosh.

Miscellaneous Language Features

CLIPS provides language constructs to perform algorithmic types of tasks. These constructs include If/Then/Else, and Do-While statements which can be executed in the then-part of the rule. Another feature that is useful in CLIPS is the ability to assign "weights" (call salience values) to rules. The rule with the highest salience value is the rule that will fire next. Once all of the criteria are met to satisfy the if-part of the rule the then part of the rule can then assert or retract facts required to control the flow to the next rule to be fired. The process of controlling some of the flow of rule firing, and the use of algorithmic constructs within a rule, greatly enhances CLIPS capability to perform systems simulations as well as making it easier for a conventional programmer to understand some of what is happening within the CLIPS program.

In addition to the language constructs provided, the user may also customize CLIPS for a particular task. The CLIPS User's Guide provides an example for adding a random number generator to CLIPS. The process shown in the User's Guide will work for any function associated with the then-part of the rule. The user could add

functions to convert an atom into a set of characters, or to read binary input from a data file, or developing drivers for special equipment (ie. software drivers for turning on and off solenoids). The capability of customizing CLIPS is an important strength to the language.

Table 5: User Interface Features*

Language Feature	LISP	Prolog	OPS5	CLIPS
Top-level control:				
invoke derivation	Yes	Yes	Yes	Yes
exit derivation		Yes		
exit system	undefined	Yes	Yes	Yes
Watch derivation:				
complete	Yes	Yes	Yes	
selective	Yes	Yes		Yes
Pause and step:				
pause from program	Yes	Yes	Yes	
pause at named entity		Fact or rule	Rule	
pause after n cycles			Yes	Yes
asynchronous interrupt		Yes		
step thru named entity	Yes, via statement	Yes		
step thru all	Yes	Yes	Yes, via run	Yes, via (run 1)
Inspect KBS				
named KB object	Yes	Yes	Yes	Yes, facts as a list rules by name.
matching KB object		Yes	Fact only	
derivation-state		Goal stack	Conflict set	Conflict set
Manipulate KB:				
add KB object		Yes	Facts only	Facts only
delete KB object		Yes	Yes	Yes
Manipulate derivation:				
abort		Yes		System dependent
backup program cycles		Yes	Yes	
continue	Yes	Yes	Yes	Yes (from program errors)
suspend derivation		Yes		Yes (on program errors)

A CLIPS APPLICATION

The CLIPS language is a good choice for writing a control system or simulation of a control system. The rule based nature of the CLIPS language provides an intuitive and quick medium for developing control rules.

As an example, a system designer required that a pressure of a vessel should never exceed 95 psi, and that at 75 psi a warning message should be sent to the operator. The types of rule that would be used to realize this control would be:

*Expanded from [Cugini 87], Table 6, p. 30.

(define rule: Warning-message
if (vessel-pressure => 75) => (then) (printout "Warning--vessel pressure has reached " (vessel-pressure/95)*100 " percent capacity))

(define rule: Activate-pressure-relief-valve
if (vessel-pressure => 95) => (then) (open (pressure-valve-3)) and (printout "Warning--vessel pressure critical. Relief valve has been activated"))

These rules are not in the CLIPS rule format because the language syntax would look confusing without sufficient explanation. The rule structure is similar though.

Though this in not AI in the strict sense, the rules are capturing the "rules of thumb" that the expert (the system designer) would use to control the system. The real strength of using CLIPS for the control language is that each rule stands on its own, any modification to the system would occur on a rule to rule basis with a minimal to other rules (ie. changing the name of a variable in one rule will have no effect on the function of another rule). For these reasons, Honeywell is reviewing CLIPS as a candidate language for demonstrating embedded ES capability in controllers for the Space Station.

CONCLUSIONS ABOUT CLIPS

Of the four languages, LISP is the most flexible but requires the most work to produce an ES. If the KBS requires high levels of flexibility or different types of inference operations during a single session then LISP would be the better choice of languages. Prolog and OPS5 provide a faster route for developing an ES, while also being easier to maintain, but at the expense of execution time and system memory.

Because of its embedability, its expandability, and its smaller size, CLIPS would be the better selection for embedding low-level ES capability within a control system. CLIPS is similar to OPS5 in its general operation. CLIPS is ment for use on personal computers or smaller computer systems and is the only language that was developed for embedded applications (putting ES capability into another system). Control systems inherently require forward chaining data processing to move from sensor inputs to a controlled output. The forward chaining rule base characteristics of CLIPS make it a good language for developing control systems.

N88-16366

An Easy-to-use Diagnostic System Development Shell

L. C. Tsai, J. B. Ross*, C. Y. Han**, W. G. Wee
Department of Electrical And Computer Engineering
University of Cincinnati
Cincinnati, Ohio 45221

ABSTRACT

This paper describes an expert system development shell for diagnostic systems, the Diagnostic System Development Shell (DSDS). The major objective of building the DSDS is to create a very easy-to-use and friendly environment for both types of users -- knowledge engineers and end-users. The DSDS is written in OPS5 and CommonLisp. It runs on a VAX/VMS system. A set of domain-independent, generalized rules is built in the DSDS, so the users need not be concerned about building the rules. The facts are explicitly represented in a unified format. These features make the DSDS very easy to use. A powerful check facility which helps the user to check the errors in the created knowledge bases is provided. A judgment facility and other useful facilities are also available. A diagnostic system (DS) developed based on the DSDS system is question driven and can call or be called by other knowledge-based systems written in OPS5 and CommonLisp. A prototype DS for diagnosing a Philips constant potential X-ray system has been built using the DSDS.

1. Introduction

Recently, expert systems have been applied to a variety of fields such as medicine, finance, engineering, and science [4,5]. The number of expert systems developed has doubled annually since the late seventies [2]. This rapid increment of the number of applications has led to the appearance of a wide range of commercially available expert system tools or so-called expert system development shells. Currently, these expert system tools are responsible for about 85% of fielded systems [3]. Most of the shells are powerful and fancy. But they are complex. Reference manuals with hundred of pages are not uncommon. It takes a relatively long time to learn them and to use them effectively. In addition, users are supposed to have some background in AI techniques and programming languages. In general, even with the shell available, developing an expert system is not a trivial task. For instance, in building of a medium size rule-based expert system hundreds of rules have to be created. Many issues such as the overall system structure and control schemes have to be considered by the user. Thus, it is desirable to create an environment that would take away the burden of learning a complicated development shell and creating rules that are related to the domain knowledge. In addition, this shell could be easily used by the system designer and end-users alike. With this objective in

* with the Aircraft Engine Business Group of General Electric Co., Evendale, Ohio.
** with Department of Computer Science, University of Cincinnati.

mind, the Diagnostic System Development Shell (DSDS) has been developed and is presented in the next sections.

Both the system architecture and its major components are presented in Section 2. A prototype diagnostic system (DS) developed by the DSDS is discussed in Section 3 and, finally, the concluding remarks are in Section 4.

2. The Diagnostic System Development Shell

The Diagnostic System Development Shell (DSDS) is a software utility that allows users, who know little about AI concepts, techniques, and any high level computer languages, to develop diagnostic systems and to use them.

Unlike the existing tools, there is a set of domain-independent, generalized rules in the DSDS. All the user needs to consider is the coding of domain knowledge (facts only) into knowledge bases. The facts, which are explicitly represented in lists with a unified format, are organized in nodes of decision tree structures. These features make both the DSDS easy to use and the knowledge base easy to build, modify, and expand. To monitor and to help users in building the DS a powerful check facility in the DSDS can pick up most of the possible mistakes the user might make. Since no AI techniques and programming skills are required, the domain experts can concentrate on coding the domain knowledge. A number of facilities, which include judgment, explanation, and help with graphics, will make the end-users feel the system is more friendly and convenient to run a DS developed by the DSDS.

2.1 Architecture of DSDS

The DSDS consists of eight major components. They include the generalized rule set (GRS), the fact knowledge base (FKB), and facilities such as the checker, the helper, the explainer, the judger, the DCL LISP interface and the graphic displayer. Among these components, the GRS, the FKB, and the checker are more important and unique. They will be discussed in the rest of this section.

The system architecture of the DSDS is shown in Figure 1. The built-in generalized rule set is in charge of reasoning. In the development mode of a DS, the user inputs domain knowledge into the FKB through an editor and invokes the checker to check the related LISP syntax and the node semantic rules.

During the execution of the DS, the system interacts with the user by generating questions to the user. The answers provided by the user will be checked by the judger which will identify and treat all the possible incorrect answers. The helper offers help with graphics when the user asks for. The DCL LISP interface makes it possible to use DCL under the OPS5 and COMMON-LISP environments. The graphic displayer are Fortran programs that displays graphics when help is provided. The explainer provides explanations of 'why' and 'how'.

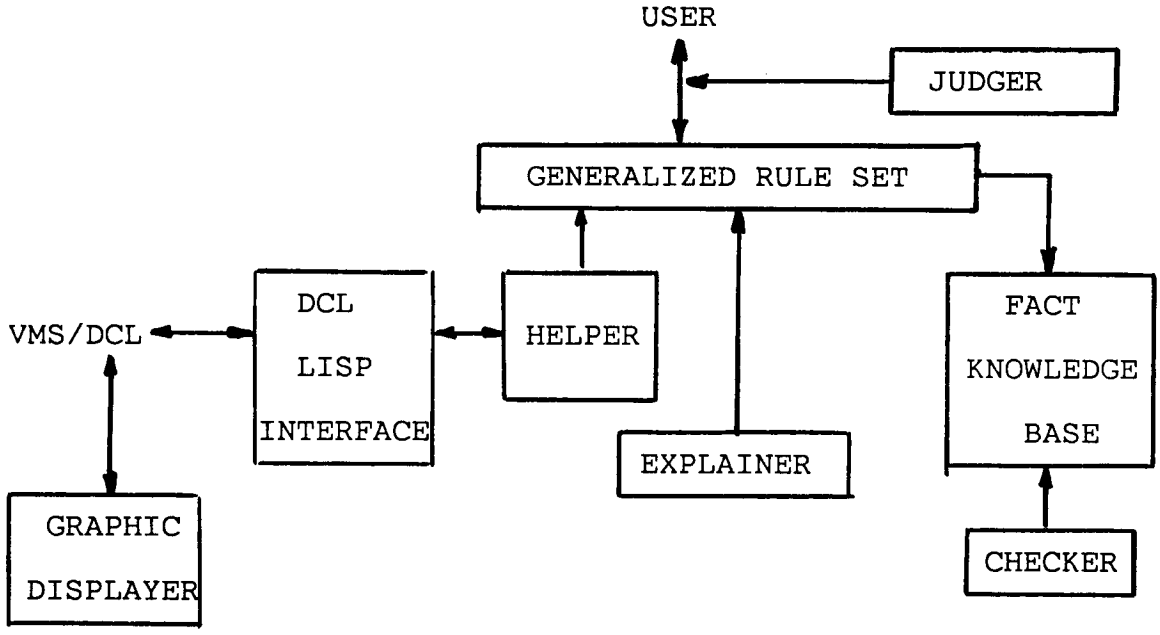


Figure 1. The Architecture of the DSDS system.

2.2 The Generalized Rule Set

The GRS contains the rules that will control the reasoning process while traversing down a decision tree during a diagnosis. The generalized rule set is based on two assumptions. One is that there is only a single source of error and the error is not transient. This assumption, as can be seen in most of the troubleshooting guides, is common in the field of diagnoses [1,7]. In fact, if there are more than one error source, the DS may be run several times to isolate all of them.

The other assumption is that the decision trees, on which the DS is based, are basically binary trees. The branches of every node of the decision trees represent two opposite propositions, yes and no, or positive and negative. This assumption fits diagnostic procedures well. The "car won't start" problem [6] is a typical example. Actually, the the binary tree structure may become a type of graph in which a node still has only two children, but different nodes could share a child, one node's children could be its parents or ancestors.

To illustrate the GRS, assume a simple, generic, binary decision tree of five nodes is given, in which the nodes N1 is the parent node of the nodes N2 and N3, and the node N3 is the parent node of the nodes N4 and N5. The nodes of the decision tree contain domain dependent facts, which will be described in the next section. Also, assume that in the node N1, a diagnostic test is being made and either answer yes or no is obtained. If yes, the node N2 will be pursued. In this case, N2 is a leaf that means a set of conclusions can be generated, problem related suggestions can be listed, and explanations be provided. If no, N3 is pursued and facts in node 3 will identify that this node is

not a leaf and a new series of actions such as generating intermediate status or results, continuing further testing, or other diagnostic actions are triggered. The process repeats for each node traversed until a leaf node of the decision tree is encountered.

2.3 The Fact Knowledge Base

Since the GRS is built in, the knowledge base includes facts only. So the knowledge bases in the DSDS are regarded as the fact knowledge bases. Facts, which are represented in lists, are organized in nodes of decision trees. Information related to a node is included only in the corresponding list, which consists of attribute-value pairs. All the lists are defined by a unified LISP format. There are three types of attributes: the necessary, the link, and the optional attributes. Each node list has all the necessary and link attributes and part of optional attributes. There are some syntax rules, which specify that the value types of certain attributes should be symbols or strings. There are also some semantic rules, which stipulate relations between attributes.

The FKBs are accessed by the generalized rules and transferred into OPS5 working memory. Unlike the generalized rules, the FKBs can be developed, modified and expanded by the user.

2.4 The Checker

Since the FKBs are defined by LISP functions and the users are supposed to know nothing about LISP. Some facilities should be available to avoid making mistakes. Two kinds of facilities could be helpful. The first is a special interactive interface which asks users questions, interpreters the answers and codes them into FKBs. But with this approach the users may easily get bored, especially all the facts are represented in the same way, the user will be asked the same questions repeatedly. The other way is to provide a check facility. Users can code the facts into FKBs directly through a text editor, then use the checker facility to check the developed FKBs. Since the facts are represented in a unified format, the second approach is more effective in the DSDS.

The checker is able to pick out related LISP syntax errors. It withdraws the decision trees from the built FKBs and displays them on the CRT. In addition, the checker can check the semantic rules. Once the checker picked out some error, it would print out error message, analyze the error and give the possible causes of the error. If a fatal error happens, the checker will stop working. Otherwise, it will continue to check until all the FKBs are checked.

3. An X-ray System Diagnostic System

A prototype -- an X-ray system DS was built by using the DSDS on VAX/VMS system. It is a DS for diagnosing a PHILIPS Constant Potential X-ray system. The X-ray system consists of two major units, the high voltage power supply and the computerized

controller. The controller contains nine printed circuit boards. The domain expertise came from the troubleshooting part of the X-ray system service manual and a domain expert. Based on this expertise twenty three decision trees corresponding to the same number of symptoms were built. The FKB includes 19 files with more than 5000 lines of code. The Checker was used to pick out all the syntax and semantic errors. To reduce the search space, this DS isolates the faults at four levels. The diagnostic procedure first finds out the fault is in the controller unit or in the high voltage power supply unit. Secondly, it isolates the faults in a certain board, then in a certain circuit and finally in a replaceable component.

4. Conclusion

An easy-to-use and friendly environment for designing diagnostic systems has been presented. The major features of the system are that the knowledge representation and input are facilitated by a unified list format and the reasoning of the expert system is done by the built-in generalized rule set. No previous knowledge of AI techniques and programming languages is required. The user needs only to concentrate on coding the domain knowledge into the so-called FKBs.

Overall the system is very straightforward to use. The prototype, explained in Section 3, has shown that the main objective of the DSDS system has been achieved. As a minor flaw of the implementation, the display of graphics is rather slow, since the process of generating graphics in VAX/VMS is complex. The OPS5 program has to call the display program written in Fortran via Lisp, DCL, and the display program has to get the data from disk.

5. References

[1] Davis, R. et al., "Diagnosis Based on Description of Structure and Function," Proc. of AAAI-82, 1982, pp.137-142.

[2] Gilmore, J. F. and Pulaski, K., "A Survey of Expert System Tools," Proc. of 2nd Conf. on AI Applications, 1985, pp.498-502

[3] Hamon, P. (ed.), "Inventory and Analysis of Existing Expert Systems," Expert System Strategies, Vol.2, No.8, Aug.,1986, pp.1-17.

[4] Hayes-Roth, F., Waterman, D. A., and Lenat, D. B. (eds.), Building Expert Systems, Addison-Wesley, Reading, MA, 1983.

[5] Waterman, D. A., A Guide to Expert Systems, Addison-Wesley, Reading, MA, 1985.

[6] Weiss, S. M. and Kulikowski, C. A., A Practical Guide of Designing Expert Systems, Rowman & Littlefield Pubs, 1984.

[7] A Constant Potential X-Ray System Model 161/321 Service Manual, Philips.

N88-16367

An Inference Engine for Embedded Diagnostic Systems†

Barry R. Fox
Artificial Intelligence Group
McDonnell Douglas Research Laboratories
PO Box 516, St. Louis, MO 63166

Larry T. Brewster
Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401

Abstract. This paper describes the implementation of an inference engine for embedded diagnostic systems. This system consists of two distinct parts. The first is an off-line compiler which accepts a propositional logical statement of the relationship between facts and conclusions and produces the data structures required by the on-line inference engine. The second part consists of the inference engine and interface routines which accept assertions of fact and return the conclusions which necessarily follow. Three design goals of this inference engine are emphasized. First, it is logically sound. Given a set of assertions it will generate exactly the conclusions which logically follow. At the same time, it will detect any inconsistencies which may propagate from an inconsistent set of assertions or a poorly formulated set of rules. Second, the memory requirements are fixed and the worst-case execution times are bounded at compile time. Third, the data structures and inference algorithms are very simple and well understood. This system has been implemented in Lisp, Pascal, and Modula-2. The data structures and algorithms are described in detail.

Introduction

Advanced aircraft and spacecraft are becoming increasingly reliant on onboard electronic systems. At the same time, onboard electronic systems are becoming increasingly complex, interrelated, and interdependent. Because of this reliance, it is necessary to constantly validate the behavior of onboard electronic systems, and when errors are detected, to quickly identify and isolate the faulty subsystems. Advances in artificial intelligence technology make it possible to construct embedded diagnostic systems which can monitor, validate, diagnose, and when necessary, deactivate critical onboard systems.

Embedded diagnostic systems impose implementation constraints which eliminate the use of most of the commercially available artificial intelligence tools. The implementation language and target processor are often dictated by contract. The memory requirements must be bounded and modest and the inference cycle must be bounded and predictable. The reliance on such systems for mission safety and success dictates that the behavior of the inference engine be demonstrably correct.

Many validation and diagnostic problems can be represented and solved entirely within the framework of zero-order (propositional) logic. For example, those problems which have traditionally been solved using decisions tables, debugging flowcharts, or decision trees involve a finite number of facts having true or false values which in turn are logically related to a finite number of intermediate and final conclusions. Generic facilities for the construction of embedded diagnostic systems are provided in the Zero-Order Environment for Test and Analysis (*Zeta*) described below.

† Research supported in part by the McDonnell Douglas Independent Research and Development Program.

PRECEDING PAGE BLANK NOT FILMED

Design Goals

The architecture and implementation of *Zeta* were guided by several design goals. Emphasis was placed on achieving generality and simplicity without sacrificing correctness nor capability.

The first goal was to make the system independent of any specific programming language or computer architecture. This would allow the system to be implemented with familiar programming languages on conventional architectures. At the same time, specific onboard computer architectures would not be eliminated, nor would special Lisp or Prolog architectures be excluded. This goal further requires that the data structures and algorithms be documented in sufficient detail that a new implementation in a new environment can be produced quickly.

The second goal was that the inference engine be generic. It should be possible to adapt a working inference engine to new diagnostic problems simply by creating and compiling the knowledge base for those new problems. A working implementation of the inference engine should be available as re-usable, off-the-shelf software.

The third goal was to make the inference engine demonstrably correct. Given a knowledge base which defines the logical relationship between observations and conclusions and given a set of assertions the inference engine should generate exactly the conclusions which logically follow. At the same time it should detect any inconsistencies which may propagate from an inconsistent set of assertions or a poorly formulated set of rules. Moreover, it should be possible to establish these properties from an analysis of the data structures and algorithms.

The fourth goal was to make the inference engine operate successfully and correctly with partial information. It should be possible to assert facts one at a time. With each assertion, the inference engine should be able to derive exactly those conclusions which follow from the aggregate of the present and preceding assertions.

The final goal was that, given a fixed knowledge base, the inference engine should have time and memory requirements which are both bounded and modest. This goal strongly influenced the choice of data structures and algorithms; it required the introduction of certain optimizations; and it also placed some limitations on the acceptable forms for a knowledge base.

External Knowledge Representation

The input to this system is a formula composed of the logical operators **and**, **or**, **xor**, **not**, and **implies**, parentheses for constructing sub-expressions, and symbols, which denote propositional parameters of the system under consideration. There is no explicit distinction between observations and conclusions. The input formula simply identifies the relevant boolean parameters of the system and the logical relationships between them.

The input formula can be a conjunction of rules of the form (*antecedent* **implies** *consequent*) but the input can be considerably more general than that allowed by most familiar rule-based systems. Most rule-based systems require statements in the form of an implication with additional restrictions that disjunctions (or more complicated expressions)

are not allowed in the consequent. For example, the phrase **(alpha xor beta xor gamma)** is very hard to formulate in a traditional rule-based system without enumerating one rule for each relevant combination of alpha, beta, and gamma. The input to this system can be an arbitrary boolean formula (with some limitations imposed only to eliminate syntactic ambiguity).

The only significant restriction on the input is semantic. It is assumed that the inference engine will only be used for mapping observations into conclusions and will not be used to derive new rules or prove theorems about the interrelationships between rules. For example, given the rule **(((alpha implies beta) and gamma) implies delta)** and a separate rule **(alpha implies beta)** it is certainly true that **(gamma implies delta)**. However, the detection and resolution of all such inter-rule relationships would be too time consuming to perform on-line. To guarantee bounded time and space requirements for the inference cycle, it is assumed that all such combinations of rules have been identified and resolved beforehand.

Internal Knowledge Representation

While the input to this system may be an arbitrary propositional formula, a much more regular structure is required for an efficient on-line inference cycle. For that reason, ZETA is composed of two parts. An off-line compiler which normalizes the given propositional formula and an on-line inference engine which performs the deductive processes. This normalization proceeds in four stages. First, expressions involving **xor** and **implies** are mapped into expressions involving only the operators **and**, **or**, and **not**. Second, all negated sub-expressions are recursively rewritten using deMorgan's law, resulting in a formula composed only of **and, or**, and positive or negative propositional variables. (Hereafter, positive or negative propositional variables will be referred to as *literals*. Third, the given formula is converted to conjunctive normal form through applications of the boolean distributive law, resulting in a conjunction of disjunctive clauses which in turn are composed only of literals. Finally, each disjunctive clause is mapped into a sequent of the form (*conjunction* **implies** *disjunction*). The natural interpretation of a sequent is that the truth of each literal in the antecedent implies the truth of at least one literal in the consequent. By convention, an empty antecedent is implicitly true, while an empty consequent denotes a contradiction. The sequent constructed from a disjunctive clause consists of an empty antecedent and a consequent identically equal to the given clause. Deductive processes which may be applied to sequents are discussed in the next section.

Inference Algorithm

Activities of the on-line inference engine are driven by a series of assertions and retractions of fact. It is useful to allow both assertion and retraction for very practical reasons. During development of a knowledge base, the knowledge engineer may wish to establish a particular state of the inference engine and then explore situations which can emanate from that state. It would be laborious to repeatedly reset the inference engine and then carry it to each situation through a series of assertions. Instead, it is better to be able to assert a fact, to determine its effect, and then to retract that fact in order to explore the immediate effect of other faults or conditions. On-line, this ability can be used to ensure the integrity of the diagnostic process in time critical situations. For instance, some physical conditions are intermittent. In a time-critical situation it would be risky to reset

the inference engine and repeat the entire history of observations and assertions simply because an intermittent physical condition no longer holds. In other situations it may be discovered that a sensor itself is faulty. Again, time may not permit a complete reset of the inference engine just to remove the conclusions derived from that erroneous sensor.

The inference cycle begins by placing an assertion on an agenda of activities to be performed. On each inference cycle one item is removed from the agenda and processed, but additional items may be placed on the agenda as a result.

The first step in processing an assertion is to determine whether it is consistent with the present state of the inference engine. Three conditions may hold. An assertion is an assignment of a boolean value to one of the propositional variables. If the variable is presently undefined, then any assertion for that proposition is considered to be consistent. If the variable presently has a value, and the value to be bound to that proposition is the same, then the assertion is considered to be redundant. However, if the variable presently has a value but it differs from the value to be bound to that proposition, then the assertion is considered to be inconsistent.

The second step in processing an assertion is to derive any conclusions which necessarily follow. This step is unnecessary for redundant assertions and must not be performed for inconsistent assertions. Given the semantic restrictions on the knowledge base discussed above, the derivation of necessary conclusions is both correct and efficient. This is accomplished by two techniques. First, the inference engine makes use of an index, constructed when the knowledge base was compiled, and inspects only those sequents which can potentially produce a conclusion from the given assertion. The index does increase the memory required to store the knowledge base by approximately a factor of two. However, the additional memory requirement can be more than offset by the following guarantee. The time required to process an assertion is independent of the size of the knowledge base; instead, it is related to the number of sequents which contain a given literal, and upon the number of conclusions which depend upon it. Second, instead of evaluating or analyzing a sequent, the inference engine applies a very simple rewriting rule based upon the natural interpretation of a sequent. Given an assertion that a literal **L** should be true, and given a sequent which contains **not-L** in its consequent, then some other literal in the consequent must be true. Hence, remove **not-L** from the consequent and include **L** in the antecedent. If only one literal remains in the consequent after this rewriting, then that literal must be true and an appropriate assertion is placed on the agenda.

A significant advantage of this sequent representation and method of inference is that assertions are reversible by retraction. The inference cycle begins by placing a retraction on the agenda. On each inference cycle one item is removed from the agenda and processed. As before, other retractions may be placed on the agenda as a result.

Like an assertion, a retraction is consistent if the value to be removed is equal to the value which a variable presently holds; it is inconsistent if the value to be removed is the opposite of the value which the variable presently holds; and it is redundant if the variable is presently undefined. There is an additional case. The value to be removed may match the value which the variable presently holds, but that value may be required or supported by the consequent of some sequent. Therefore that retraction would introduce an inconsistency if performed and is considered to be impossible. As with an assertion,

a retraction is performed only if it is consistent with the present state of the inference engine.

The process of rewriting sequents under retraction is the reverse of the assertion process described above. Given a retraction of some literal **L**, and given a sequent which contains **L** in its antecedent, first inspect the consequent of that rule. If only one literal remains in the consequent before this rewriting, then this retraction may necessitate the retraction of the consequent as well. If no other sequent supports this consequent, then place the appropriate retraction on the agenda and perform the rewriting: remove **L** from the antecedent and include **not-L** in the consequent. If the consequent has other support before the rewriting, or if the consequent contains more than one literal, then perform only the rewriting step.

Architecture

The architecture of this system can be sketched at two levels. At the highest level the system consists of two separate programs. The first accepts a propositional formula which defines the logical relationship between the boolean parameters of the system to be monitored. The output of the first program is the program fragments necessary to declare and initialize the data structures for the second program. The second program is produced by combining these program fragments with the off-the-shelf code for the inference engine. This two-stage process removes any parsing and normalization costs from the on-line system and it produces a diagnostic program of minimal size.

The lower level structure of the first program is not significant. The program can be viewed as a black box which performs the compilation function. The structure of the second program is significant. It provides the interface to the inference engine. This interface includes an initializing procedure which resets all propositional values to undefined and rewrites every sequent to the form **(() implies disjunction)**. There is a procedure for making an assertion which requires two parameters, a propositional identifier and a value to be bound to that variable, and which initiates the inference cycle described above. There is a procedure for making a retraction which requires only a propositional identifier, assuming that the user wishes to retract the present value of the given proposition. All deductive results produced during an inference cycle are placed on a stack; these results can be retrieved one at a time by simple procedure calls. Other procedures exist which will return the present value of a proposition, its symbolic name, etc.

Conclusion

The Zero-Order Environment for Test and Analysis system has characteristics which make it suitable for embedded diagnostic systems. Notably, the representation and methods of inference are independent of any specific programming language or computer architecture; the time and memory requirements are modest and the upper bounds may be determined prior to run time; the inference engine is generic and can be adapted to new applications by introducing a new knowledge base; the inference engine is demonstrably correct generating exactly the conclusions which follow from a given set of assertions while detecting any inconsistency; the inference engine complements the facility for incremental assertion with a facility for incremental retraction.

N88-16368

CLIPS: An Expert System Tool for Delivery and Training

Gary Riley, Chris Culbert, Robert T. Savely, and Frank Lopez
NASA/Johnson Space Center
Artificial Intelligence Section - FM7
Houston, TX 77058

Abstract

The 'C' Language Integrated Production System (CLIPS) is a forward chaining rule-based language developed by the Artificial Intelligence Section at NASA/Johnson Space Center. This paper examines the requirements necessary in an expert system tool which is to be used for development, delivery, and training. Because of its high portability, low cost, and ease of integration with external systems, CLIPS has great potential as an expert system tool for delivery and training. In addition, its representation flexibility, debugging aids, and performance, along with its other strengths make it a viable alternative for expert system development.

Introduction

Expert system technology is a major subset of Artificial Intelligence and has been aggressively pursued by researchers since the early 1970's. In the last few years, both government and commercial application developers have given expert systems considerable attention as well. An entire industry has grown to support the development of expert system tools and applications, with a wide variety of both hardware and software products now available. The availability of expert system tools has greatly reduced the effort and cost involved in developing an expert system.

Despite all this, expert systems have generally failed to make a major impact in application environments. This failure has stemmed from tool vendor's overemphasis on expert system development environments to the detriment of options for delivery of expert systems and training in expert system technology. Viable delivery options are necessary to field expert systems. Training options in expert system technology are necessary for the widest possible dissemination of this technology.

The 'C' Language Integrated Production System (CLIPS) is a forward chaining rule-based production system developed by the Artificial Intelligence Section at NASA/Johnson Space Center. Version 1.0 of CLIPS, developed in the spring of 1985 in a little over two months, accomplished two major goals. The first of these goals was to gain useful insight and knowledge about the construction of expert system tools and to lay the groundwork for future versions. The second of these goals was to address the delivery problems of integrating and embedding expert systems into conventional environments. Version 1.0 successfully demonstrated the feasibility of continuing the project.

Subsequent development of CLIPS greatly improved its portability, performance, and functionality. A reference manual[2] and training guide[5] were written. The first release of CLIPS, version 3.0, was in July of 1986. The latest version of CLIPS, version

PRECEDING PAGE BLANK NOT FILMED

4.1, was completed in September of 1987. CLIPS is currently available through COSMIC (see appendix).

The development of CLIPS, though not a significant advance in expert system representation capability, is a significant advance in the concept of providing a low cost tool that can be used for training, development, and delivery. The use of CLIPS in these areas will be examined in this paper with special emphasis on CLIPS's capabilities as a delivery and training tool.

Delivery

CLIPS addresses several issues key to the use of an expert system tool for delivery. Among these issues are: the ability to run on conventional hardware; the ability to run on a wide variety of hardware platforms; the ability to be integrated with and embedded within conventional software; low-cost delivery options; the ability to separate the development environment from the delivery environment (i.e. run-time modules); the ability to run efficiently (both speed and memory), and migration paths from development to delivery environments.

One major requirement for a delivery tool is the ability to run on conventional hardware. Current state-of-the-art expert system software tools are almost all based in LISP and run only on specialized LISP hardware, such as the Symbolics or TI Explorer, or on the new generation of workstations such as the SUN or Apollo. While these workstations do provide a conventional platform for delivery, investment in conventional hardware often precludes adding additional or specialized hardware to support expert systems. The question to ask when considering delivery is not "Is there a conventional machine that supports the expert system tool I want to use?", but rather "Does the conventional machine I have support the expert system tool I want to use?"

Portability of the expert system tool code insures the ability to deliver on a wide range of hardware from microcomputers to minicomputers to mainframes. Because CLIPS is written in C and special care was taken to preserve portability, CLIPS is able to provide expert system technology on a wide variety of conventional computers. CLIPS has been hosted on over a dozen brands of computer systems ranging from microcomputers to mainframes without code changes. To maintain portability, CLIPS utilizes the concept of a portable kernel. The kernel represents a section of code which utilizes no machine dependent features. To provide machine dependent features, such as windowed interfaces or graphics editors, CLIPS provides fully documented software hooks which allow machine dependent features to be integrated with the kernel.

The ability to integrate with and embed within existing code is an important feature for a delivery tool. Integration guarantees that an expert system does not have to be relegated to performing tasks better left to conventional procedural languages. It also allows existing conventional code to be utilized. The capability to be embedded allows an expert system to be called as a subroutine (representing perhaps only one small part of a much larger program). Many tools view themselves as the "master" program and only permit control to be passed to other programs through them. CLIPS allows integration with C programs as well as integration with other languages such as FORTRAN and ADA. In addition, many functions are provided which allow CLIPS to be

manipulated externally. Because the source code is available, CLIPS can be modified or tailored to meet a specific user's needs.

Applications should be delivered as economically as possible. Many tools require the entire development environment to run an application. This necessitates buying a new copy of the tool for every delivered application. Some tools provide the capability to generate run-time modules. These run-time modules are basically equivalent to the executable modules generated by compilers for procedural languages. Run-time modules allow the unneeded functionality and information associated with the development environment to be stripped away from the delivery environment. This is a desirable characteristic, but for many tools, each copy of a run-time module must be purchased.

CLIPS effectively addresses the problems of low cost delivery. The cost for CLIPS source code is $200. This initial cost provides unlimited copies of CLIPS for delivery, development, and training. In addition, CLIPS also provides the capability to generate run-time modules.

Another key feature for a delivery tool is efficiency. CLIPS is based on the Rete algorithm[3] which is an extremely efficient algorithm for pattern matching. CLIPS version 4.1 compares quite favorably to other commercially available expert system tools based on the Rete algorithm. CLIPS performs its own memory management, eliminating the problems associated with garbage collection that plague most LISP based expert system tools.

A good delivery tool should also provide a graceful migration from the development environment to the delivery environment. CLIPS was designed to be as compatible as possible with other forward chaining rule-based languages. Thus the capabilities of CLIPS closely mimic the forward chaining capabilities of tools such as ART[1] and OPS5[4]. Such similarity allows the migration of programs developed using the powerful environments provided by LISP machines and tools such as ART to conventional hardware. This approach allows problems to be prototyped using the most productive environment available and then delivered in the desired environment.

Training

CLIPS addresses several issues key to the use of an expert system tool for training. A good training tool should have knowledge paradigms of the appropriate complexity, be low cost, run on easily accessible hardware, and have easy to use interfaces and debugging tools.

A good training tool should meet the Goldilocks criteria. That is, its knowledge paradigms should be neither too complex nor too simple. Hybrid tools incorporating multiple paradigms such as ART can be extremely difficult to learn to use. In addition, training using hybrid tools can often require additional training on the use of the LISP language and environment. On the other hand, simple tools such as decision tree builders, often lack the necessary depth to teach a reasonable variety of expert system concepts. CLIPS's single paradigm, forward chaining rules, provides a reasonable compromise between complexity and simplicity. It also provides a training path to the more complex hybrid tools.

A low cost tool is necessary for the widespread use of expert systems technology. Groups interested in expert system technology may not be able to spend large sums of money to use tools which utilize some of the more powerful knowledge representation paradigms. Inexpensive tools that provide only limited paradigms may give a false impression of the types of problems that can be solved with expert system technology. CLIPS provides a reasonably powerful representation paradigm at an extremely low cost.

A tool for training should run on easily accessible computers. In other words, whatever computers are available. This means that the tool should be as portable as possible and run on microcomputers (which make very good target machines for training tools). Portability for the training tool is also essential because the development and delivery environments may differ from the training environment. It is not productive to train with a tool that cannot then be used to develop and deliver an expert system.

A training tool should be easy to use. Mouse/menu driven windowed interfaces with graphical knowledge browsers and integrated editors are highly desirable. Because of portability concerns, the standard version of CLIPS uses a command line interface. However, software hooks are available to allow more sophisticated interfaces to be built. Several such training and development interfaces have been developed. Version 4.1 of CLIPS has an integrated editor (designed to run on UNIX, VMS, and MS-DOS machines) which allows quick editing and recompilation of rules. CLIPS has a wide variety of debugging commands to assist in examining and debugging an expert system.

Development

Several features are key to the use of an expert system tool for development. Among these features are: multiple knowledge representation paradigms; sophisticated user interfaces, debugging aids, and browsers; integrated editors, compilers, and other system tools; and the ability to rapidly prototype and iteratively refine an expert system.

Access to multiple representation paradigms is quite useful during the development stage of an expert system since the representation paradigms needed to solve a problem are not always known in advance and can frequently change as the problem solution evolves. CLIPS utilizes only one major paradigm: forward chaining rules. The use of a single paradigm in a tool is a drawback for development, however, expert systems such as DEC's XCON[6] built using OPS5 demonstrate that single paradigm tools can be used to develop significant expert systems.

Most of the interface features described for training are also desirable for development. Sophisticated interfaces provide ease of use in creating, debugging, and examining an expert system. An integrated environment provides quick turn-around time during development which facilitates rapid prototyping and iterative refinement.

Clearly the development environment is the area in which CLIPS compares least favorably to other expert system tools. However, while CLIPS does not provide a development environment as powerful as the major state-of-the-art expert system tools, it does provide the basic features necessary for the development of expert systems.

Conclusion

CLIPS has several characteristics which make it highly advantageous to use as a tool for delivery and training. Among these are its portability, low-cost, and ease of integration with conventional environments. In addition, it provides an environment suitable for the development of expert systems.

References

[1] ART Reference Manual, Inference Corporation, Los Angeles, CA. 1986.

[2] Culbert, C. "CLIPS Reference Manual". NASA Technical Memo FM7(87-220), December, 1986.

[3] Forgy, C. "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem". Pages 17-37, Artificial Intelligence 19, (1982).

[4] Forgy, C. OPS5 User's Manual. Department of Computer Science document CMU-CS-81-135, Carnegie-Mellon University, Pittsburgh, PA. 1981.

[5] Giarratano, J. The CLIPS User's Guide. NASA Internal Note 86-FM-25, October, 1986.

[6] Harmon, P., and King, D. Expert Systems: Artificial Intelligence in Business. John Wiley & Sons Inc. New York, NY, 1985.

Appendix

NASA, DoD, and other government agencies may obtain CLIPS by contacting the CLIPS help desk 9:00 AM to 4:00 PM CST weekdays at (713) 280-2233. CLIPS is available outside the government through the COmputer Software Management and Information Center (COSMIC), the distribution point for NASA software. The program number is MSC-21208. Program price is $200.00 and documentation price is $17.00 (as of January 14, 1987). The program price is for the source code. Further information can be obtained from COSMIC:

COSMIC
Software Information Services
The University of Georgia
Computer Services Annex
Athens, Georgia 30602
(404) 542-3265

N88-16369

Issues Associated with Telerobotic Systems in Space

Scott A. Hofacker
Bernard J. Schroer
Johnson Research Center
The University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

This paper presents the research issues in using telerobotics in space. Included in this paper is a review of previous research in space telerobotics and the results of several telerobotics experiments.

INTRODUCTION

A number of studies have been conducted beginning in the 1970's concerning the issues associated with telerobotic systems. Some of the research issues in telerobotic systems for space application are: video viewing, scene lighting, feedback delays, and predictive displays.

Video Viewing

Three camera locations are commonly considered in most research studies. The first camera is mounted on the manipulator and gives a close up view directly over the robotic gripper. A second camera provides an overall view or scene of the task area. This camera provides depth perception to the operator. A third camera may also be necessary to provide an overhead view of the task area.

Several previous camera studies [Pennington 1983] have concluded that operators prefer two views. One view is positioning the camera for a side view, or orthogonal to the task board, and above the center of the board with a 60 degree field of view. The second view is positioning the camera above the task area and viewing down at a 70 degree angle.

A related issue in video viewing is the use of black and white versus color cameras. Most researchers have used a black and white camera on the manipulator and a color scene camera [Collins 1986]. The research has concluded that black and white cameras are adequate [Yorchak 1986].

Several studies have also been conducted concerning the use of stereo cameras. A dexterity test consisting of the peg-in-hole task with various size pegs concluded that the smaller pegs required more time than the larger pegs [Brye 1977]. Also, the response time was considerably less for a stereo camera system as compared to an orthogonal monoptic system.

In summary, an evaluation of a number of recent video viewing studies [Yorchak 1986] concluded that two cameras are better than one, two cameras positioned orthogonally are better than two cameras positioned to produce stereo, and a third camera for an overhead view does not seem necessary.

PRECEDING PAGE BLANK NOT FILMED

Scene Lighting

Since all space telerobotic tasks are performed in space, scene lighting is a critical factor. For example, a task can go from total darkness to total brightness by a mere change in orientation. Likewise, it is possible to obtain a variety of shadow conditions based on the location and position of the robot and the task in space. Scene lighting is relatively easy to simulate in a laboratory. For example, in the laboratory flat black drapes can completely surround the facility. Also, black cloths can be placed on the floor and around the task boards to eliminate any reflections. The overhead lights are then extinguished. The source lighting can then be directed at various intensities and focused on the task. In addition, a variety of shadows can be displayed on the task by positioning the light sources accordingly.

Feedback Delays

Time delays are inherent in any teleoperation system. Sending and receiving transmissions from space or space vehicles can result in time delays between 0.5 and 8.0 seconds. The length of delay depends on the number of switching satellites and the data processing times. A number of studies have been made of the effect of time delays on operator performances. In general, these studies have concluded that the task time increases with an increase in time delays [Ferrell 1965].

A related issue to time delays is the effect of limited camera bandwidths on operator performance. Bandwidths are generally limited because of the vast amount of data transmission necessary between the manipulator and the control station. Several studies [Ranadive 1979 and Deghuee 1980] have concluded that the operator can perform familiar simple tasks with considerably reduced bandwidths; however, these studies were done without any time delays.

Predictive Displays

Time delays cannot be completely eliminated in any teleoperation system. However, with predictive displays the operator is able to see, via a computer graphics representation of the robotic area, exactly where the robot will be after the commands are executed. As the operator moves the arm, the model will, in real time, update the graphics display to show the operator the effects of the command before the arm has actually received the command. This type of predictive feedback is useful to the operator by improving the low productivity of move and wait tactics. For example, a recent study [Sheridan 1984] found that predictive displays reduce task time between 50-150 percent. Also, in another study [Arnold 1963] predictive displays enabled the operator of a remote vehicle to drive at the same speed nearly as well with or without a time delay.

ROBOTICS LABORATORY

Figure 1 presents a system schematic of the space telerobotics laboratory at the University. The laboratory is configured around a Puma 562 6 DOF arm. Mounted on the arm is a high resolution black and white CCD camera (see Figure 2). The Puma is remotely controlled with two 3 DOF hand controllers at the control console.

Several scene cameras are located around the task board. One of the cameras is a color camera with auto white balance and a zoom lens that is mounted on a pan and tilt unit. Both the zoom lens and the pan and tilt unit are remotely controlled at the control console. The second scene camera is a black and white camera. All video output is fed back to monitors at the remote control console.

The inter-meshing gripper is a modification of a NASA design [NASA 1980]. The gripper is electrically operated and has two limit switches to indicate when the gripper is fully open or closed. The gripper is remotely controlled at the control console.

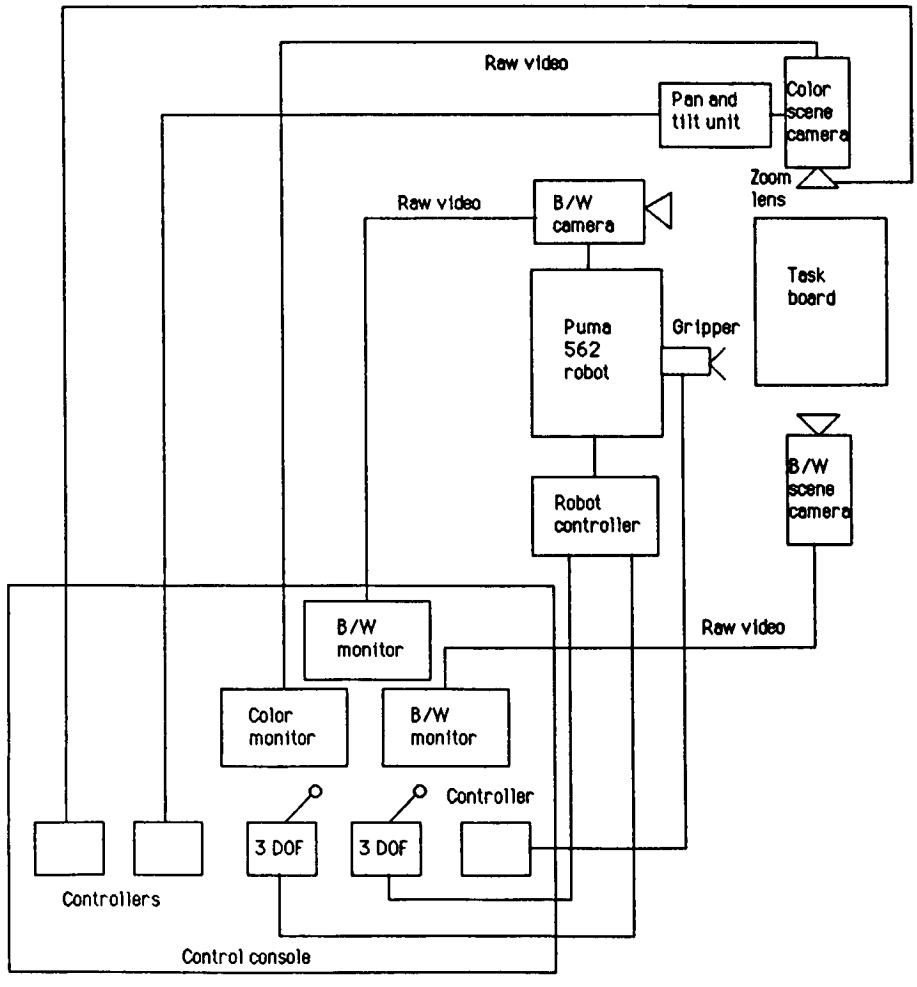


Figure 1. System schematic of robotics laboratory

TELEROBOTIC EXPERIMENTS

A simple peg-in-the-hole task was defined to evaluate the laboratory's hardware and software, especially the Puma control software. In addition, by selecting the peg-in-the-hole task, it was possible to compare and validate the results with previously published research. Figure 2 also includes a photograph of the peg-in-the-hole task board. Each board consists of three 1 1/8 inch holes. The peg diameter is one inch. Initially,

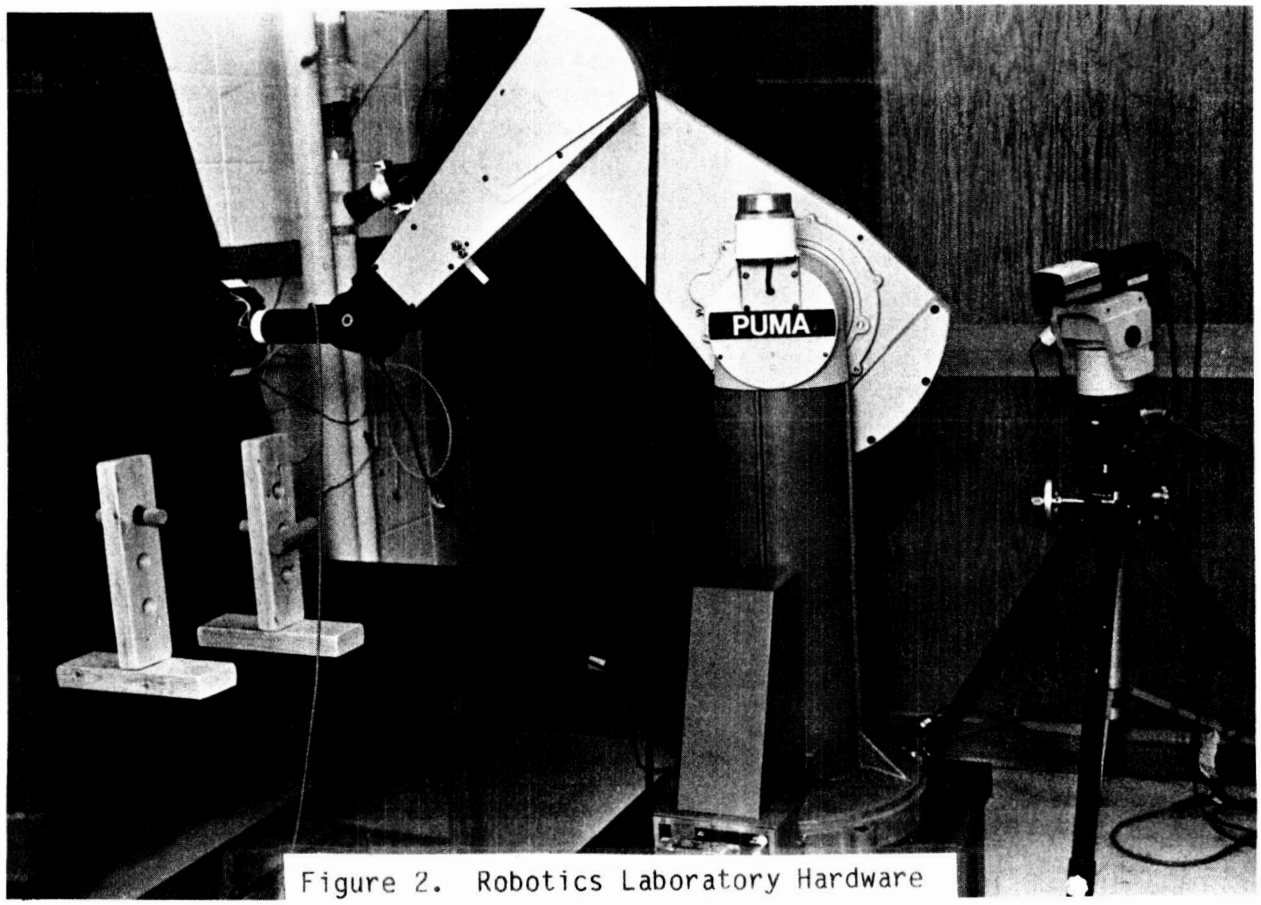


Figure 2. Robotics Laboratory Hardware

the peg is in the top hole on the right board. The task objective is to transfer the peg to the bottom hole on the left board. The response variable is the time to perform the task. This response variable is common to most telerobotic experiments. The scene lighting equipment did not arrive in time for the experiments. Therefore, all experiments were conducted with the normal laboratory lights turned on.

The following factors and levels within factors were considered in this experiment:

- Factor 1 - Time delay.
  Three levels of time delays were used (0, 1, and 2 seconds). These delays were changed through the robot control program.

- Factor 2 - Camera view.
  Three levels of camera views were used with each level consisting of two cameras: a side scene view and and an arm view where the camera was mounted on the Puma arm; an angle scene view with pan/tilt/zoom and an arm view; and an angle scene view with no pan/tilt/zoom and an arm view.

- Factor 3 - View color.
  Two levels were used: black and white and color. Only the side view camera with the pan/tilt/zoom was color.

ORIGINAL PAGE IS
OF POOR QUALITY

Each level of each factor is combined with all levels of every other factor in the experimental design. Therefore, the experimental design consists of a 3x3x2 completely randomized factorial experiment. This design results in 18 cell. Each cell is also replicated four times, two each by two student subjects, for a total of 72 runs.

The ANOVA results show a significant effect of time delay on task time ($F = 33.18$, $p<0.05$). A number of other research studies have been made on the effect of time delays on operator performance. In general, these studies have concluded that the task time increases with an increase in time delays [Ferrell 1965 and Yorchak 1986].

The ANOVA results did not show a significant effect of camera view on task time ($F = 1.80$). While the camera view did not show a significance, the subjects stated preference for the side, or orthogonal, view as opposed to the angle view. This preference agrees with findings by Kirkpatrick [Kirkpatrick 1973] that orthogonal views are more effective.

The ANOVA results did not show a significant effect of view color on task time ($F = 2.31$). This result agrees with other researchers who have concluded that black and white cameras are adequate [Yorchak 1986]. The ANOVA results also did not show any second or third order interaction effects. These four interactions were delay and view; delay and color; view and color; and delay, view and color.

Figure 3 is a plot of the total task time for the 0, 1, and 2 second time delays. The task times were averaged for each time delay and represent 24 values. As can be seen, the total task time increased from 2.99 seconds with 0 second delay to 4.64 seconds with a two second delay.

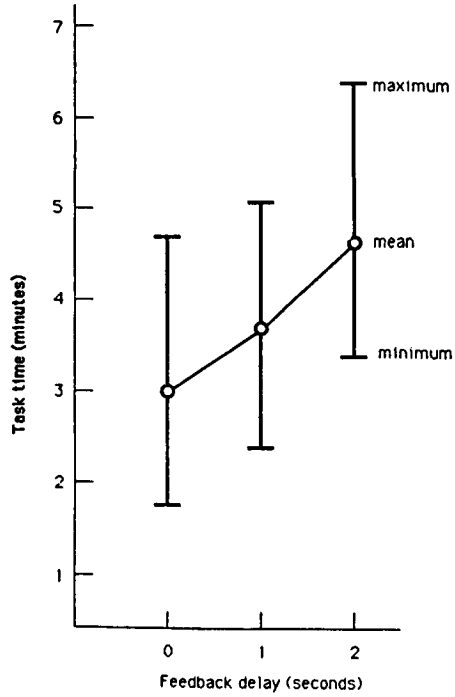


Figure 3 . Total task time with feedback delays

CONCLUSIONS

Future enhancements to the laboratory include: placing the pan/tilt/zoom under voice control, predictive displays and artificial intelligence as an operator assistant. Additional experimentation will include increased sample size, increased task difficulty, and improved task boards and lighting. This experimentation will produce data to answer questions proposed by previously research and will provide information on telerobotics for space applications.

ACKNOWLEDGMENTS

This research is being supported in part by an equipment grant from the Boeing Aerospace Company and grants from the Alabama Research Institute.

REFERENCES

1. Arnold, J. E. and P. W. Braisted, Design and Evaluation of a Predictor for Remote Control Systems Operating with Signal Transmission Delays, NASA Tech Note D-2229, December 1963.

2. Brye, R. G., P. N. Frederick, M. Kirkpatrick, III, and N. L. Shields, Jr., Earth Orbital Teleoperator Manipulator System Evaluation Program, Report H-77-2, Essex Corp., Huntsville, AL 35803.

3. Collins, S. L. and R. B. Purves, "Remote Servicing of Space Systems," Proceedings of the Conference of Artificial Intelligence for Space Applications, NASA, Huntsville, AL, Nov. 1986.

4. Deghuee, B. J., Operated-Adjustable Frame Rate, Resolution, and Grey-Scale Tradeoff in Fixed Bandwidth Remote Manipulator Control, Man-Machine Systems Laboratory, Report MIT, September 1980.

5. Ferrel, W. R., "Delayed Force Feedback," Human Factors, Oct. 1986.

6. Kirkpatrick, M., T. B. Malone, and N. L. Shields, Jr., Earth Orbital Teleoperator Visual Systems Evaluation Program, Essex Corp., Alexandria, VA, March 1973.

7. NASA, Mechanical Hand for Gripping Objects, NASA Tech Brief MFS-23692, Summer 1980.

8. Pennington, J. E., A Rate-Controlled Teleoperator Task with Simulated Transport Delays, NASA Technical Memo 85653, Sept. 1983.

9. Pepper, R. L. and J. D. Hightower, "Research Issues in Teleoperator Systems," Proceedings of the Human Factors Society, 1984.

10. Ranadive, V., Video Resolution, Frame Rate, and Grey-Scale Tradeoffs Under Limited Bandwidth for Undersea Teleoperation, Report TR-NR196-152, MIT, 1979.

11. Sheridan, T. B., Review of Teleoperator Research, Man-Machine Systems Laboratory, MIT, 1984.

12. Yorchak, J. P., Teleoperator Human Factors Study, MCR 86-558, Martin Marietta, Denver, CO, May 1986.

N88-16370

TELEROBOTIC CONTROLLER DEVELOPMENT

W. S. Otaguro, L. O. Kesler
McDonnell Douglas Astronautics Company
Huntington Beach, California 92647

Ken Land, Don Rhoades
NASA - Johnson Space Center
Houston, Texas 77058

ABSTRACT

Telerobotic experiments can be performed with existing technology on the orbiter to demonstrate the feasibility to perform supervised robotic material handling and positioning functions in space. To meet NASA's Space Station's needs and growth, MDAC has developed a modular and generic approach to robotic control which provides near term implementation with low development cost and capability for growth to more autonomous systems. This effort uses the MDAC developed, vision based, robotic controller and compliant hand integrated with the RMS arm on orbiter. A description of the hardware and its system integration will be presented.

A ground demonstration of this system will be performed at the Manipulator Development Facility at NASA-JSC using a full size, 1 G version of the orbiter RMS arm. Details of this program will be presented.

INTRODUCTION

The objective of the MDAC/NASA Robotic Tracker demonstration using the Manipulator Development Facility (MDF) arm at the Johnson Space Center is to functionally demonstrate the technology readiness of telerobotics (supervised autonomy) to perform approach, positioning, engagement, and assembly under man supervised but autonomous robotic operations. The MDF arm was chosen for this demonstration because it represents the implementation of teleoperation in space and can be readily used to demonstrate telerobotics as well. The MDF arm is a 1 G version of the shuttle RMS arm. Currently, the MDF arm is teleoperated with direct man or computer preprogrammed control and guidance.

TELEROBOTIC ARM DEMONSTRATIONS

This demonstration will use the existing MDF capability with a robotic tracker developed by McDonnell Douglas. This tracker will be used to process video camera inputs to determine target position, bearing, and attitude and to guide the MDF arm under operator supervision. The basic hardware elements and interface are shown in Figure 1. The robotic tracker will interface to the

MDF system through an RS232 link to the SEL 32/77 computer. The robotic tracker will be considered as a remote terminal and will input and display the following:

1. Input pitch, yaw, and roll and X, Y, Z data for operator command mode for MDF arm positioning.
2. Display data as the arm moves.

The tracker will also have the capability to perform command check and obtain response from the SEL 32/77 computer.

The crew station will initiate and stop operational commands as usual. The R12 panel will remain fully operational by being able to input pitch, yaw, roll and X, Y, Z data and perform command check.

This functional demonstration will perform manual target lock-on with autonomous target tracking, MDF arm guidance, approach, and positioning. The target range will be within 20 feet at acquisition. The target will be a standard grapple fixture target and will initially be stationary. The initial demonstration with a fixed lens camera will allow autonomous approach to about 3 feet.

TEST PROCEDURE

The MDF arm operator will position the camera to have the grapple fixture target in its field of view. The tracker operator will then place the acquisition gate about the target for lock-on by the tracker. After lock-on, the tracker will calculate the target position and input to the SEL 32/77 the new position (X, Y, Z) and attitude (pitch,, roll, yaw) to which the end of the MDF arm is to be moved. Initially, the tracker will estimate a straight line motion of the end of the MDF arm to the target with discrete movement increments of several feet. As confidence in the overall system operation is gained, larger increments will be allowed. For any movement to be executed by the MDF arm, the MDF operator must activate an execute control button on the R12 panel. After each arm movement, the tracker will calculate a new position and send it to the SEL 32/77 for execution until the camera is approximately three feet from the target.

DEMONSTRATION OPTIONS

After these static tests are completed, a series of moving target tests may be performed where the target will be moved to a new position after each arm movement. Also close in positioning (1 ft) with respect to the target will be performed as allowed by the availability of camera lenses and targets. Target engagement by telerobotic control is also being investigated using a compliant hand, which will incorporate vision based

tracker guidance for approach to the target with force and position from the elements (fingers) of the hand for final target engagement.

MDAC TRACKING SENSOR

The architecture of the multimode sensor tracker is shown in Figure 2. The multiprocessor tracker is composed of three functional parts:

1) A Fairchild CCD 3000 camera 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 required for these demonstration 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 the MDF arm. 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 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 theprocessor 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 capability 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.

COMPLIANT HAND

The MDAC-Compliant hand will consist of a drive unit, a pullmember, a strain sensor and segmented elements (Figure 3). The drive unit will be a brushless d.c. motor and a harmonic drive. The pullmember will be a metal cable within a cable guide. The sensor will be mounted between the motor and the end of the segmented elements to sense the pulling force of the pullmember.

The segmented element will be constructed of a number of individual links and compression springs. The pullmember will run from the pulley through the springs and through all links to the end of the element.

The brushless d.c. motor will rotate a pulley on its shaft to wind up the pullmember. The sensor will initially sense the resisting force created by the compression springs within the links and will feel a rapidly increasing force after contact with the target.

The MDAC segmented elements, however, have a number of individual links, which are separated by springs to average forces between links. It is this principle that gives our intelligent, sensor-controlled segmented elements the superior capability to accept an objects random curvature where each element develops its own shape. The number of links is not limited and all link dimensions are adjustable to create the best suited combination. The MDAC-Compliant hand (Figure 4) has four segmented elements and a palm camera for object identification and alignment control.

CONCLUSION

The capability of existing, packaged tracker and compliant hand hardware to perform telerobotic control functions with minor upgrades are functionally being demonstrated. Upgrades in hardware and software will be required to address the

requirements of space operations. However, a great deal of the basic development have already been and are being performed and funded by other government agencies. This demonstration, with the 1 G RMS arm, will provide additional information on the integration of this technology with existing systems for near term robotic space operations.

ACKNOWLEDGEMENTS

The support of Harry Erwin, Bob Lacy, Lynn Harvey, and Ann Marie Ching of NASA, John O'Keefe of Bendix Corp., Klaus Gruenbeck and Eric Griesheimer of MDAC-HB, and Al Eisenberg of MDAC-Houston in the planning and execution of this effort is gratefully acknowledged.

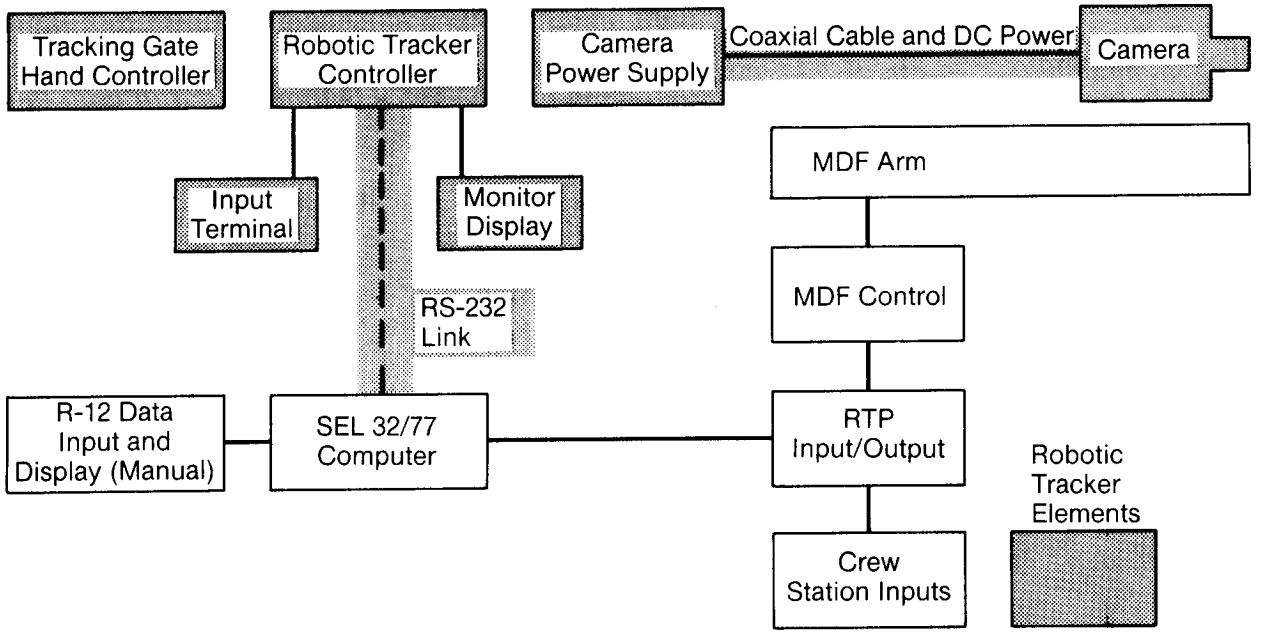


Figure 1. MDF/Robotic Tracker Hardware Elements

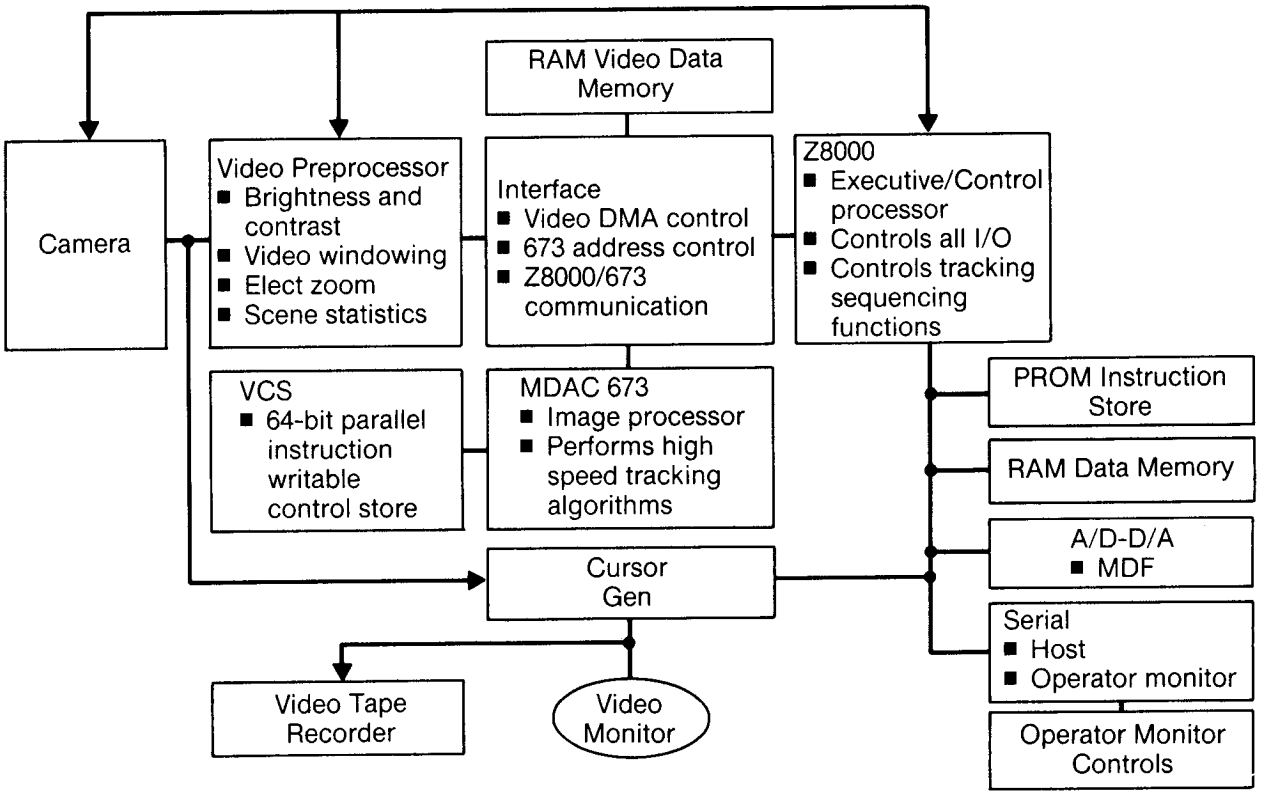
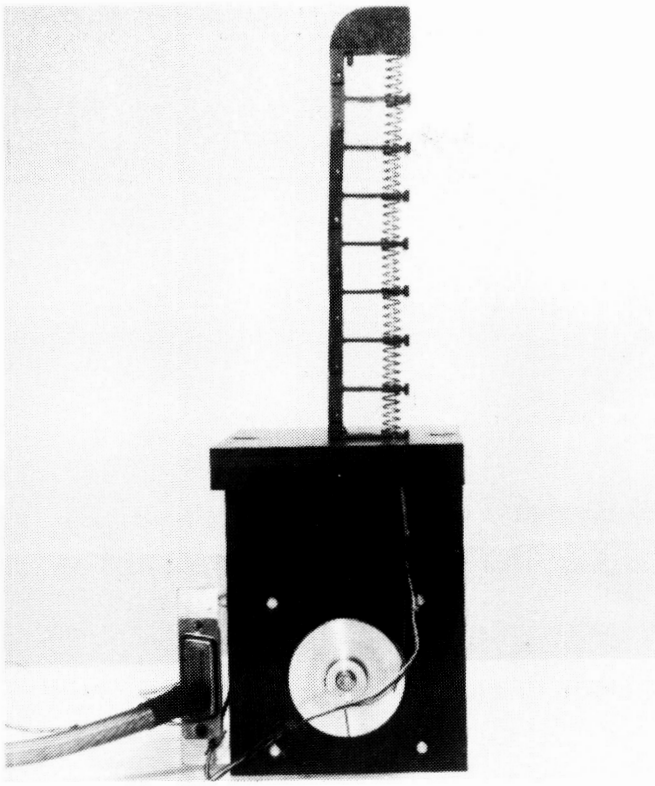


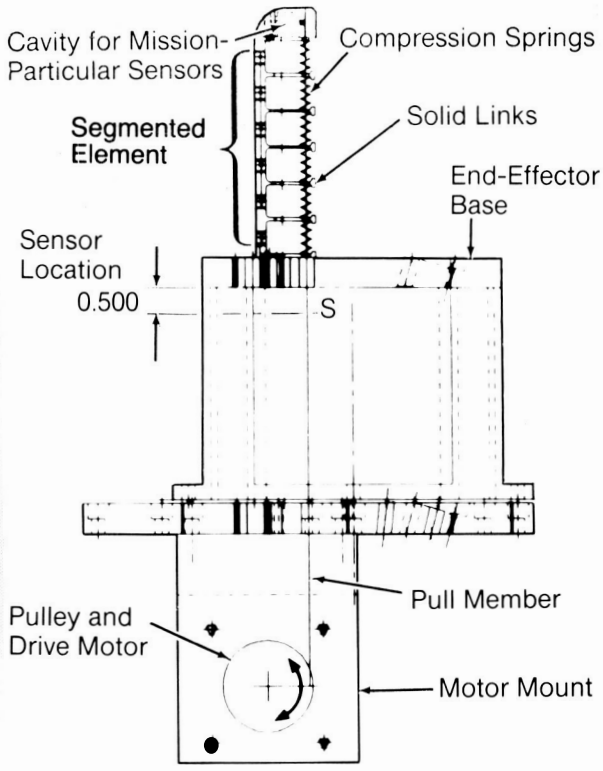
Figure 2. Architecture of the Multimode Tracker

ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY



Test Model of Single-Axis End-Effector (Extended)



Drive Schematic

Figure 3. Segmented Element Drive

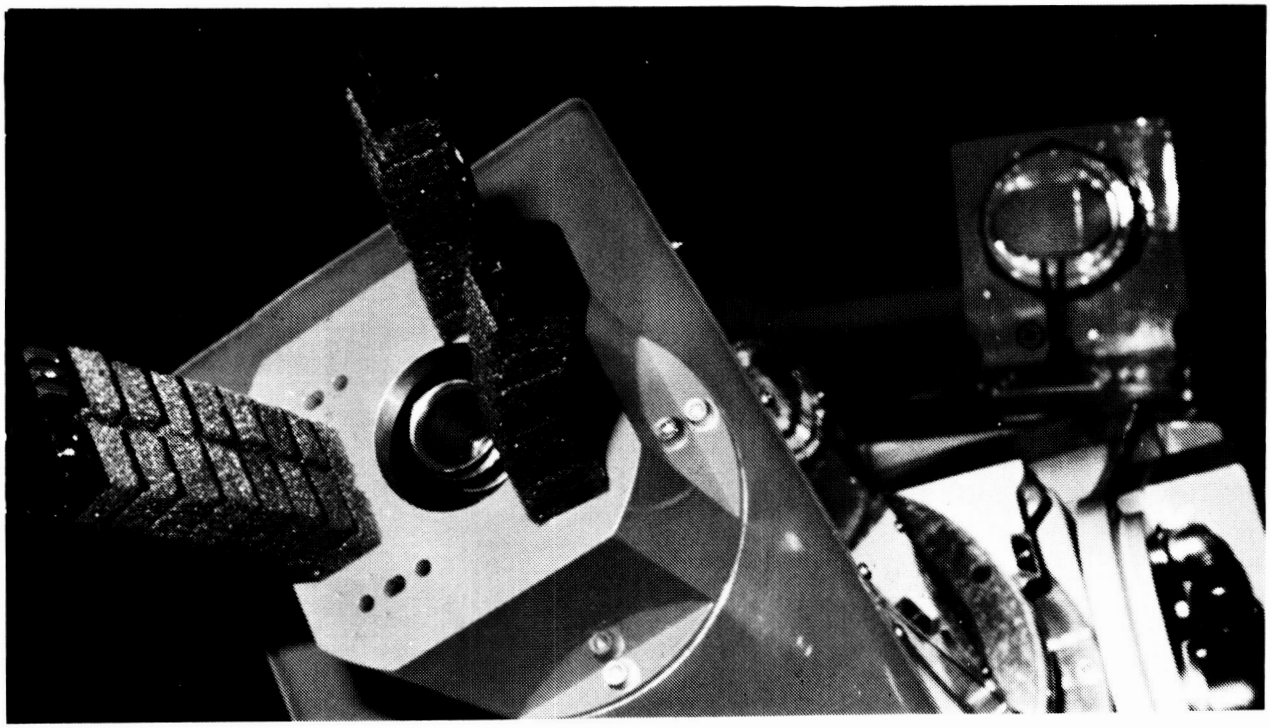


Figure 4. MDAC Compliant Hand With Four Segmented Elements and Palm Camera

N88-16371

Software Simulation of Time Delay in Teleoperation

K. Wayne Goode
Kenneth E. Johnson Research Center
University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

This paper describes research done in the Space Robotics Laboratory at UAH studying the effects of time delay on teleoperation.

INTRODUCTION

The long range goal for the NASA space station is to establish a permanent presence of man and machines in space. Because of cost and safety factors, teleoperation will be important to the fulfillment of this goal.

Teleoperation means remote operation. That is, the robot receives instructions from a human operator and then performs the task. The task to be performed is at a remote distance from the operator.

In space applications there is often a delay between the time the human operator gives a command and time the command is executed in space. There is a further delay before pictures from cameras located at the remote site are relayed to the operator. As the time delay increases, so does the time required to complete the task, thus making the operation of the remote device more difficult.

The Johnson Research Center at the University of Alabama in Huntsville has established a space robotics laboratory to study time delay and other issues of teleoperation.

This paper will describe this lab and deal specifically with the software written to control the robot and simulate time delays.

HARDWARE

Figure 1 presents a system schematic of the space robotics laboratory. The laboratory is configured around a Puma 562 robot with 6 degrees of freedom (see Figure 2) which is on loan from Boeing Aerospace Company in Huntsville.

A custom designed joystick controller with two joysticks, each with three degrees of freedom, is used to control the robot. These joysticks are connected to the robot controller through an analog to digital interface.

PRECEDING PAGE BLANK NOT FILMED

SOFTWARE

The software to control the robot is written in VAL II, the control language for Puma robots.

Joystick Calibration

The joystick outputs a voltage in the range of 0 to 10 volts for each axis. The minimum, maximum, and center are different for each axis. The calibration program asks the operator to move each of the joysticks to its extreme positions. The program reads the minimum, maximum, and center voltages for each axis and stores the information for reference when reading the joystick.

The interface is subject to electrical noise. The voltages will vary by as much as 5%. However, since the joystick does not need to be very precise, this is an acceptable error. To reduce the error, the joystick is read several times and the values averaged.

Reading the Joystick

The subroutine JOYSTICK reads the joystick controller using the analog to digital interface. The reading are normalized in the range -1 to 1 based on the calibration information recorded by the calibration program. This number is squared to give more precise control around the center point.

Any reading less than .05 is considered to be 0. This dead zone around the center keeps the robot from drifting slightly due to small errors in the joystick reading caused by electrical noise.

Time Delays

The program DELAY (see listing 1) is the move program with a time delay added. The program prompts the user for the length of the time delay at the start of execution of the program.

When using a time delay, the program must read the joystick, store the information, recall other information and move the robot accordingly. The time required to execute each loop must be the same to keep the time delay accurate. The program executes this loop seven times a second. The steps it takes are:

1. Read the joystick.

2. Calculate the new position based on the readings.

3. If the position is out of range use the last valid position.

4. Store position on the top of stack.

5. Pull next position off the bottom of stack.

6. If the robot will complete the current motion by the end of the time allowed for the execution of the loop (1/7 of second), command the robot to move to the position just pulled off the stack.

ORIGINAL PAGE IS
OF POOR QUALITY

7. Wait until the time allowed for execution of this loop has elapsed.

The robot can buffer one movement at time. That is, once a command is given to start a motion the program proceeds without waiting for the motion's completion. However, if another motion command is given before the first motion is finished, the program will wait for the first motion to finish and start the second motion before continuing with the program.

This can be a problem when the joystick must be continuously read. The solution is to skip the movement to the new location when the current movement will not be completed in the amount of time allowed. The robot will catch up during the next motion command.

The amount of time for each loop must be held constant so that the delay will be correct. An entry is read off the stack and a new one put on each time a loop is executed. This way, the time delay can be changed by varying the size of the stack. The stack should have seven entries for each second of delay because each step is 1/7 of a step long.

The computer's clock is incremented 35 times a second and the loop time must be a multiple of that time. 1/7 of a second was selected because it was the shortest time in which the steps necessary to each loop could be executed.

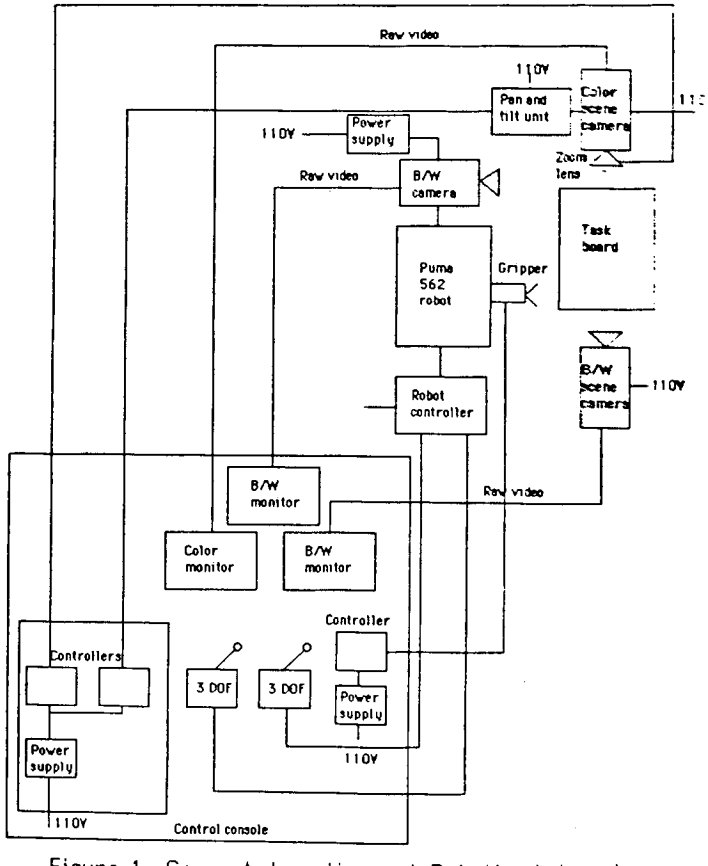


Figure 1 Space Automation and Robotics Laboratory

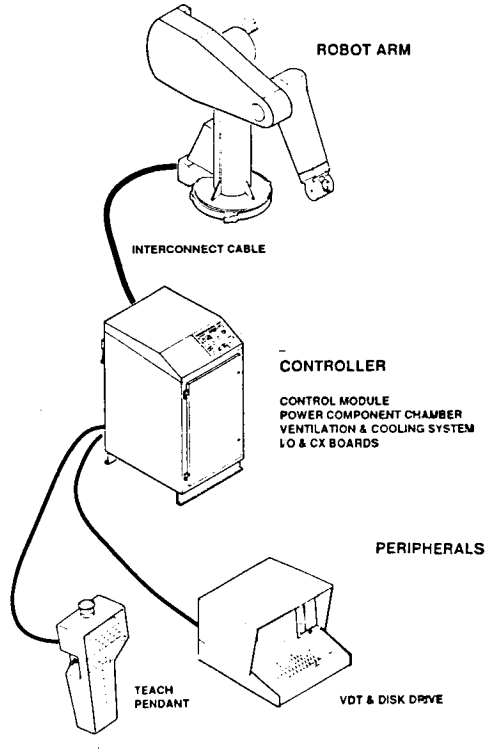


Figure 2 Puma 562 robot

The function of many of the command of the VAL II language are obvious. Some of the non-obvious commands and functions in the programs listings in Appendix B are explained below.

ADC — This function returns a value in the range -1023 to 1024 that represents a voltage in the range -10 to +10 for the analog to digital interface channel indicated.

BREAK — This command suspends program operation until the current robot motion has finished. Normally, the program starts a robot motion and continues with the program execution.

DECOMPOSE — This function returns the six components of a robot location. The components are x, y, z, orientation, altitude and rotation.

HERE — This command assigns the current location to the specified location variable.

INRANGE — This function returns a value indicating whether the specified location is in the robot's work envelope.

MOVE — This command moves the robot to the specified location.

RIGHTY — This commands sets the robot in a right handed configuration.

TIMER(-1) — This function returns the amount of time left before the current robot motion is finished.

TRANS — This function returns a location variable described by the six components given. This is the opposite of the function DECOMPOSE.

SET — This command is used to assign a value to a location variable.

Listing 1

DELAY PROGRAM

```
1       ;Delay
2
3       ;KW Goode 8JUL87
4
5       TYPE /C25, /U22, "Time delayed robot controlled program"
6       TYPE
7
8       CALL joystick
9       IF er THEN
10          TYPE
11          TYPE "The joystick power is turned off."
12          RETURN
13      END
14
15      scale 1 = 20
```

```
16        scale a = 5
17        st = 0
18        count = 0
19        rate = 5
20        tics = INT(tps + 0.5)
21
22        RIGHTY
23        MOVE TRANS (-800,0,-90,0,-90)
24        TYPE "Moving the robot to the starting position"
25        BREAK
26        HERE rpos
27        TYPE /U20,"
28        TYPE /U20, /S
29        PROMPT "Enter time delay in seconds ==>", delay
30        TYPE
31        steps = delay * rate
32        IF steps < 1 THEN
33        steps = 1
34        END
35
36        DECOMPOSE p[] = rpos
37        FOR i = 0 to steps
38            SET stack [i] = rpos
39        END
40        TIMER 1 = 0
41
42    10  SET dpos = stack [st]
43        IF TIMER (-1) <tics/rate/TPS THEN
44             MOVE dpos
45        END
46        CALL joystick
47        p[0] = p[0] + x1*scale1
48        p[1] = p[1] + y1*scale1
49        p[2] = p[2] + z1*scale1
50        p[3] = p[3] + x2*scalea
51        p[4] = p[4] + y2*scalea
52        p[5] = p[5] + z2*scalea
53        SET rpos 1 = TRANS(p[0], p[1], p[2], p[3], p[4], p[5])
54        IF INRANGE (rpos1) == 0 THEN
55        SET rpos = rpos 1
56        END
57        DECOMPOSE p[] = rpos
58        SET stack [st] = rpos
59        st = st + 1
60        IF st == steps THEN
61           st = 0
62        END
63        count = count + 1
64        WAIT TIMER (1) >= (count*tics/rate -0.5)/TPS
65        GOTO 10
```

N88-16372

MTK: An AI Tool For Model-Based Reasoning

William K. Erickson and Mary R. Schwartz
Systems Autonomy Demonstration Project Office
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

A 1988 goal for the Systems Autonomy Demonstration Project Office of the NASA Ames Research Center is to apply model-based representation and reasoning techniques in a knowledge-based system that will provide monitoring, fault diagnosis, control and trend analysis of the Space Station Thermal Management System (TMS). A number of issues raised during the development of the first prototype system inspired the design and construction of a model-based reasoning tool call MTK, which was used in the building of the second prototype. This paper outlines these issues, with examples from the thermal system to highlight the motivating factors behind them, followed by an overview of the capabilities of MTK, which was developed to address these issues in a generic fashion.

PRECEDING PAGE BLANK NOT FILMED

N88-16373

Integration of Symbolic and Algorithmic Hardware and Software for the Automation of Space Station Subsystems

Hugh Gregg, Lawrence Livermore National Laboratory
P.O. Box 808, L-310, Livermore, CA 94550
Kathleen Healey, Johnson Space Center
Houston, TX
Edmund Hack, Lockheed EMSC
Houston, TX
Carla Wong, Systems Autonomy Demonstration Project Office
NASA Ames Research Center
M/S 244-18, Moffett Field, CA 94035

Abstract

Traditional expert systems, such as diagnostic and training systems, interact with users only through a keyboard and screen, and are usually symbolic in nature. Expert systems that require access to data bases, complex simulations and real-time instrumentation have both symbolic as well as algorithmic computing needs. These needs could both be met using a general purpose workstation running both symbolic and algorithmic code, or separate, specialized computers networked together. The latter approach was chosen to implement TEXSYS, the thermal expert system, developed by NASA Ames Research Center in conjunction with Johnson Space Center to demonstrate the ability of an expert system to autonomously control the thermal control system of the space station. TEXSYS has been implemented on a Symbolics workstation, and will be linked to a microVAX computer that will control a thermal test bed. This paper will explore the integration options, and present several possible solutions.

PRECEDING PAGE BLANK NOT FILMED

C-2

N88-16374

Requirements and Options for Communications Services in support of the Systems Autonomy Demonstration Project

Richard M. Brown and Robert Yee
Systems Autonomy Demonstration Project Office
NASA Ames Research Center
Moffett Field, CA

Abstract

Work will begin this year on the development of the second of four demonstrations of automation technology under the Systems Autonomy Demonstration Project. This demonstration will involve elements of four NASA Centers: ARC, JSC, LeRC and MSFC. Unlike the first demonstration, intercenter digital data communications will be a vital element of this demo.

This paper presents a study of the requirements and options for interconnecting the development systems and demonstration testbeds at NASA centers supporting the Systems Autonomy Demonstration Project. The communications requirements of the SADP development and demonstration environments are described, potential communications protocols are examined and compared, the options for network topologies are examined, and the expected communications error rates and system availability are described.

PRECEDING PAGE BLANK NOT FILMED

N88-16375

KBS V&V as Related to Automation of Space Station Subsystems: Rationale for a KBS Lifecycle

K. Richardson and C. Wong
Systems Autonomy Demonstration Project Office
NASA/Ames Research Center
Moffett Field, CA

Abstract

The role of V&V in software has been to support and strengthen the software lifecycle and to ensure that the resultant code meets the standards of the requirements document. KBS V&V should serve the same role, but the KBS lifecycle is ill-defined. This paper explains the rationale of the simple form of KBS lifecycle - a development process with certain critical differences from the traditional lifecycle.

Among differences is the requirements specification, which is in some respects more flexible than in traditional software development, but which nonetheless assumes a more significant portion of the KBS development effort.

Special KBS development requirements are accommodated where possible by modifications to the traditional software lifecycle. Research areas are suggested for those aspects which present new or unusual difficulties for V&V.

PRECEDING PAGE BLANK NOT FILMED

N88-16376

"Monitoring of Space Station Life Support Systems with Miniature Mass Spectrometry and Artificial Intelligence"

Richard A. Yost and Jodie V. Johnson
Department of Chemistry
University of Florida
Gainesville, FL 32611

and

Carla M. Wong
Systems Autonomy Demonstration Project Office
M/S 244-18
NASA Ames Research Center
Moffett Field, CA 94035

Abstract

The combination of quadrupole ion trap tandem mass spectrometry with artificial intelligence is a promising approach for monitoring the performance of the life support systems in the space station. Such an analytical system can provide the selectivity, sensitivity, speed, small size, and decision-making intelligence to detect, identify, and quantitate trace toxic compounds which may accumulate in the space station habitat.

PRECEDING PAGE BLANK NOT FILMED

N88-16377

HSTDEK: Developing a Methodology for Construction of Large-Scale, Multi-Use Knowledge Bases

Dr. Michael S. Freeman
NASA/Marshall Space Flight Center

Abstract

Marshall Space Flight Center is the lead center for the Hubble Space Telescope Design/Engineering Knowledgebase (HSTDEK) Project which is being funded by NASA's Office Of Aeronautics and Space Technology (OAST) as an element in the Core Technology Research effort of the Systems Autonomy Technology Program (SATP). The primary research objectives of HSTDEK are to develop a methodology for constructing and maintaining large scale knowledge bases which can be used to support multiple applications. To insure the validity of its results this research is being pursued in the context of a real-world system, the Hubble Space Telescope. This paper will describe the HSTDEK objectives in more detail, briefly discuss the history and motivation behind the project, outline the technical challenges faced by the project, and present the approach which is being taken to achieve its goals.

Introduction

The capture of design data and, to a lesser extent, design knowledge is already an integral part of large-scale development programs within NASA. However, it is primarily a manual and paper-based process. Even where design data is developed or stored in an electronic medium, there is no common framework or representation which permits the results of the traditional engineering activities comprising the development effort to support an integrated application. With regard to design data, for example, design of the structural components of a system may be done using a Computer Aided Design tool but the weight and center of gravity of the components are not directly available to the computer based tools used by the mass properties analyst to calculate integrated system values. Nor is the expertise of the thermal expert directly available to the electrical engineer designing some power consuming (and thus heat generating) component of the system.

At present, design data is shared through the development of an enormous set of paper documents on any major project, and expertise is shared by means of design reviews based on these documents. These reviews are very costly, involving dozens of experts, and typically identify hundreds of discrepancies which must then be corrected in the design. Most of these discrepancies could be avoided if each expert had the benefit of the other experts' related expertise throughout their design instead of only at checkpoints such as Preliminary and Critical Design Reviews. If that expertise exists only in the heads of our engineers, we will not be able to effectively share it even within a project, much less between projects. But it is now possible to capture that design knowledge in a form which will make much of it constantly available. The capture of such knowledge and the design data to which it refers in a particular domain, however, will result in a knowledge base whose scale greatly exceeds that of current knowledge engineering efforts. It also implies an ability, not currently available, to utilize knowledge about the design of a system to support multiple engineering activities such as evolutionary design, fault diagnosis, planning for operations and maintenance, training, etc.

PRECEDING PAGE BLANK NOT FILMED

The primary objective of the HSTDEK project is to develop a methodology for constructing knowledge bases on the scale required to support NASA projects and for making the diverse types of design data and design knowledge found in these projects available for use by multiple knowledge based systems performing different functions. That NASA projects in particular are excellent domains for the use of knowledge based systems technology has been known for some time [1]. This fact is now being recognized by NASA management as critical to achieving the goals we have set for the 1990's and beyond, especially in the Space Station.

The Advanced Technology Assessment Committee (ATAC), which Congress mandated in Public Law 98-371 to identify specific Space Station systems which advance automation and robotics, has formed a subcommittee to assess the state of the art in design knowledge capture. Its recently released report concluded with the following recommendation.

> "NASA has an exceptional opportunity for both technology benefit and technology transfer, which can succeed if NASA:
>
> - o Determines a firm course of action, and assigns responsibility for execution of each step
> - o Provides the supportive environment needed for community-oriented development
> - o Establishes and enforces community guidelines for and standards for information exchange
> - o Encourages the effective development of emerging DKC technology to a stage of readiness, while adopting a discerning posture towards its use." [2]

An Automation and Robotics program has been established in OAST to achieve the goals set for NASA by ATAC in these areas. The Systems Autonomy Technology Program (SATP), of which HSTDEK is an element, is the primary vehicle for pursuing the ATAC automation goals.

The Systems Autonomy Technology Program

The overall program goal of the SATP is to develop, integrate, and demonstrate the technology required to enable intelligent autonomous systems for future NASA missions [3]. It is managed by the Chief of the Information Sciences Division at NASA/Ames Research Center, with the advice of representatives from each NASA center who comprise the Systems Autonomy Intercenter Working Group (SAIWG), and has a planning horizon of about ten years. Within the Core Technology Research effort of the SATP, the HSTDEK Project is an element of the Knowledge Acquisition task in the Planning and Reasoning area. The goals of this task are defined as follows.

> "The objectives are to develop the ability to preserve the 'corporate memory', i.e., to ensure that all the facts, heuristics, and other information gained during the design, construction, and testing of a device are available in a practically usable form during the operational lifetime of the device." [4]

The HSTDEK project has been established in direct response to these objectives.

The HSTDEK Project

The HSTDEK Project is managed out of the Space Systems Division in the Systems Analysis and Integration Laboratory of Marshall Space Flight Center (MSFC). This Division has Systems Engineering responsibility for the Hubble Space Telescope (HST) at MSFC, which is the NASA lead center for HST development. The project has four main tasks, to be accomplished over a five year period, each of which involves collaborative efforts with other organizations.

- Development of a methodology for construction of large-scale knowledge bases which can each be used to support multiple knowledge based applications.
- Assessment of the data products of the traditional engineering activities which composed the HST development process as sources of design data and design knowledge, as well as insertion points for knowledge engineering technology.
- Construction of the HST Design/Engineering Knowledgebase itself.
- Construction of two knowledge based systems which validate the methodology used to construct HSTDEK by performing multiple functions in support of HST verification and operations.

This approach is based on the principle that research into the use of knowledge engineering technology will be most effective if it is done in the context of a "real world" application such as support of the Hubble Space Telescope. Each of these tasks will be discussed in more detail below, but it is worthwhile at this point to discuss the characteristics of the HST which led to its use as the domain for this project.

The Hubble Space Telescope Domain

It may seem odd, at first, to select a mature project such as HST as a domain in which to develop methods for design data and design knowledge capture. The fact that the design activities are essentially complete, however, offers a number of advantages to this project. On a new program, it might be necessary to wait several years for access to some design products and activities. Using a mature project, these inputs are all available immediately and can be analyzed as a whole system of activities. The refinement of the design products over the development cycle could also result in a large amount of wasted effort in construction of a knowledge base. This would not be a consideration if the goal was to use knowledge based systems to support the development process. The goal here, though, is to develop a methodology which will be useful in capturing the design data and knowledge rather than a knowledge based system to assist in the design process. It is therefore preferable to work from a set of design data and knowledge which is truly representative of that produced in NASA development projects. In addition to the data products of HST development, many of the experts involved in its design will still be available to contribute their expertise to the project through HST launch and checkout. Finally, NASA is about to initiate a number of major development projects; Space Station, Advanced X-Ray Astronomical Facility (AXAF), the Orbital Maneuvering Vehicle (OMV), etc. To select any of these projects, as the domain in which to pursue this research would mean that the technology developed would not be available for use during those projects. As described above, technology transfer into the Space Station project is a major driver for this research and such delays are not acceptable.

In addition to the benefits offered by the timing of the HST project, there are technical characteristics of the HST which recommend it as a framework for research into design knowledge capture. It is a major NASA project, whose design, construction, and test have taken many years to complete. It is typical of large-scale NASA development projects in that it involves many technical disciplines, and the integration of a design developed by a number of different contractors. As the Project Management center, MSFC has responsibility for meeting cost, schedule, and technical performance goals. Goddard Space Flight Center is responsible for the HST Science Instruments, the Data Management system, the ground system, and the Science Institute. The Lockheed Missiles and Space Company is the prime contractor for the Support Systems Module (SSM), including its design, development, fabrication, assembly and verification; integration of systems engineering and analysis for the overall HST; and support to NASA for planning and implementing ground, flight, and orbital operations support. Perkin-Elmer Corporation is responsible for the Optical Telescope Assembly (OTA). Finally, the European Space Agency (ESA) is responsible for the HST Solar Arrays. [5] This distribution of HST expertise across a large number of experts in different organizations and at different geographical locations, makes it a very difficult and unusual problem from the knowledge engineering perspective, but also is typical of the NASA design domain. A methodology which can handle the complexity of the HST domain is likely to be useful in most NASA projects. Finally, the knowledge base developed in this project will benefit the HST project itself throughout its fifteen year operational lifetime. This will be a significant "spin-off" of what is, in itself, valuable research.

Project Planning

The funding of the HSTDEK Project as an element of the Systems Autonomy Technology Program implies a collaboration between MSFC and ARC. As the NASA lead center for automation, ARC acts as a consultant to MSFC in planning the knowledge engineering activities in HSTDEK including the selection of appropriate development and delivery systems for knowledge based systems, as well as knowledge engineering training for MSFC participants in the project. The four tasks identified earlier as comprising the HSTDEK Project also involve collaborations among several organizations, as discussed below.

The most fundamental task to be performed in the HSTDEK project is the development of a methodology for construction of large-scale knowledge bases which can each be used to support multiple knowledge based applications. These are areas of basic research in knowledge engineering. The scope and complexity of the design data and design knowledge to be captured in this project far exceeds that of current knowledge bases. Current knowledge bases also are typically focused on a single aspect of a system, with little knowledge of other subsystems or technical disciplines. A necessary component of this task is the establishment of a framework within which different types of knowledge about the HST can be accommodated. This will allow knowledge based systems utilizing HSTDEK to reason from a deeper and more basic understanding of the system. The use of a comprehensive knowledge base like HSTDEK to support several applications will raise another major research issue; the design of multi-user knowledge bases. This is an extremely difficult technical problem [6,7], and will probably have to be addressed as a separate research project. It should be noted that this goal of developing a methodology for constructing a knowledge base which can accommodate the variety and quantity of design knowledge required by NASA projects is not the same as developing a methodology for capturing all the types of design knowledge typically generated in such projects, but complements such efforts.

The research required to accomplish this task will be performed at the Knowledge Systems Laboratory of Stanford University. The first year's effort will have three major results; a small prototype of a multi-use knowledge base in the HST domain, a set of knowledge representations to support the construction of a larger multi-use knowledge base, and the preliminary specification of a methodology for constructing large-scale, multi-use knowledge bases. The prototype will cover two subsystems of the HST (the Pointing and Control System and the Electrical Power System), one constructed at Stanford and one constructed at MSFC using the knowledge representations developed at Stanford. It will support two applications, probably fault diagnosis and a scheduling function. The knowledge acquisition effort at Stanford will draw primarily on the expertise available in the HST designers at the LMSC facility in Sunnyvale California, as well as work at the Lockheed Artificial Intelligence (AI) Center. Knowledge base construction at MSFC is discussed below.

The second main task in HSTDEK is to assess the data products of the traditional engineering activities which make up a NASA development project as sources of design data and expertise, as well as processes which could directly benefit from use of knowledge based systems technology. This is planned as joint research between MSFC and the Computer Science Department of the University of Alabama in Huntsville. The first product of this activity will be a prioritized list of HST data products as knowledge sources which will assist MSFC knowledge engineers in attempting to acquire as much design data and expertise as possible from them. This will be refined and generalized based on experience with other development projects at MSFC. The other main product of this activity will be a recommended approach for including knowledge engineering in NASA's traditional engineering activities.

The third main task in HSTDEK is actual construction of a large-scale knowledge base which can support multiple applications. Initially, traditional knowledge engineering methods will be utilized. As the Stanford research develops improved approaches to the problem, they will be incorporated in this effort. Several organizations will be involved in construction of HSTDEK, in addition to the Stanford researchers. It is expected that the bulk of the actual knowledge engineering will be done by MSFC personnel. To enable that effort, a six month knowledge engineer training program at the Lockheed AI Center is planned for as many as ten MSFC participants over the five year duration of the HSTDEK Project. This will have the highly beneficial side effect of creating a significant knowledge engineering capability at MSFC. The participation of Lockheed personnel as knowledge engineers as well as domain experts is also being investigated, both through the Lockheed AI Center and LMSC/Sunnyvale. The Lockheed AI Center has already been pursuing several projects in the HST domain, and it is highly desirable to find a means of incorporating these efforts into HSTDEK. Similarly, agreements with GSFC and the HST Science Institute will be sought to enable their participation in HSTDEK. Coordination of these efforts will be the responsibility of MSFC.

In order to confirm that knowledge bases constructed using the methodology developed in this project can actually be used in multiple real-world applications, two knowledge based systems will be built and demonstrated by MSFC in operational environments. First, the HST Operational Readiness Expert (HSTORE) system will utilize an early version of HSTDEK to support checkout of the HST during the Orbital Verification phase immediately following HST launch. HSTORE is planned to provide a fault diagnosis capability based on HSTDEK to MSFC engineers in the Huntsville Operational Support Center. A second knowledge based system called GESST (Ground-based Expert System for Space Telescope) will be made available for use in the Space Telescope Operational Control Center at GSFC in 1992 using a more complete version of HSTDEK. It is intended to fully demonstrate a large-scale, multi-use knowledge base.

Conclusions

In its third progress report, the ATAC voiced concerns that NASA was not pursuing the development of automation technology rapidly enough to adequately support Space Station design, and urged that NASA include a requirement for design knowledge capture specifically in its detailed design and development phase proposal requests for Space Station [8]. An earlier staff study by the Office of Technology Assessment stated similar concerns and concluded that aggressive research should be initiated without delay into advanced automation and robotics technology [9]. The HSTDEK Project directly addresses these concerns by establishing collaborations with members of the AI community, both in industry and academia, to effectively pursue the research necessary to enable the capture of design data and knowledge in major NASA projects, demonstrating this technology in the real-world domain of the Hubble Space Telescope, and developing NASA's in-house ability to utilize such technology.

References

1. Freeman, M. S., and Hooper, J. W., "Factors Affecting the Development of Expert Systems in NASA", First Conference on Artificial Intelligence for Space Applications, NASA/Marshall Space Flight Center and the University of Alabama in Huntsville, October 1985.

2. "Design Knowledge Capture: State-of-the-art Technology and Application Assessment", Report of the Design Knowledge Capture Subcommittee of the Advanced Technology Advisory Committee, September 1987.

3. "Systems Autonomy Technology Program Plan Executive Summary" (Final Draft), NASA/Ames Research Center, July 1987, page 1.

4. "Systems Autonomy Technology Program (SATP) Plan" (Final Draft), NASA/Ames Research Center, July 1987, page 29.

5. Space Telescope Systems Description Handbook, LMSC/D974197B, 31 May 1985, Section 2.5.

6. Freeman, M. S., "The Emergence of Multi-User Expert Systems", Second Conference on Artificial Intelligence for Space Applications, NASA/Marshall Space Flight Center and the University of Alabama in Huntsville, November 1986.

7. Freeman, M. S., "An Investigation of Multi-User Expert Systems", Dissertation, University of Alabama in Huntsville, May 1987.

8. "Advancing Automation and Robotics Technology for the Space Station and for the U. S. Economy: Progress Report 3", NASA-TM89190, October 1986.

9. "Automation and Robotics for the Space Station: Phase B Considerations", an Office of Technology Assessment Staff Paper, 1985.

N88-16378

Knowledge-Based Monitoring of the Pointing Control System on the Hubble Space Telescope

Larry L. Dunham
Thomas J. Laffey
Simon M. Kao
James L. Schmidt
Jackson Y. Read

Lockheed Missiles and Space Co., Inc
O/62-85 B/579,
P.O. Box 3504
Sunnyvale, CA 94088-3504
(408) 742-3415

Abstract

This paper describes a knowledge-based system for real-time monitoring of telemetry data from the Pointing and Control System (PCS) of the NASA Hubble Space Telescope (HST) that enables retention of design expertise throughout the three decade project lifespan by a means other than personnel or documentation. The system will monitor performance, vehicle status, success or failure of various maneuvers, and in some cases diagnose problems and recommend corrective actions using a knowledge base built using nominal mission scenarios and the over 4,500 telemetry monitors from the HST. The real-time system consists of a data management task, an inferencing task, and an I/O task that run concurrently in multiple CPUs and communicate via a message passing scheme. Real-time graphical displays can be selected by the user on the multi-level block diagram of the HST control system displayed by the I/O task. This paper describes the application of L*STAR to analysis of monitors from the PCS. A detailed description of the multiprocessing architecture will be described in another paper in the conference. L*STAR is undergoing continued development and is being used to monitor test cases produced by the Bass Telemetry System in the Hardware/Software Integration Facility at Lockheed Missile and Space Co. in Sunnyvale, California. LMSC is assembling the vehicle under the direction of NASA/Marshall with a 1989 launch planned.

Introduction

Lockheed Missiles and Space Company (LMSC) is the prime contractor for the Support Systems Module (SSM) and Integration Systems Engineering for NASA's Edwin P. Hubble Space Telescope (HST). The HST is considered one of the greatest scientific experiments in history of mankind. The field of astronomy will be revolutionized by the opportunity to see to the edges of the universe (14 billion light years away), seven times further than we can now observe. Unimpeded by the Earth's atmosphere, scientists will be capable of seeing objects fifty times fainter than those visible today with a stability equivalent to focusing a laser beam in Washington, D.C. on a dime in Boston!

The launch of the HST by the Space Shuttle has been delayed due to the Challenger space shuttle accident. The current launch date is late 1989 and the lifetime of the spacecraft after launch is expected to be a minimum of fifteen years. This gives the total project, from design to end of mission, a lifetime of over a quarter of a century. Capturing knowledge of engineering experts to ensure continued expertise over the project's lifetime is one of the goals of the application described in this paper.

A second factor driving this application is the complexity of the ground operations task. The Space Telescope Operations and Control Center (STOCC) at the NASA/Goddard Space Flight Center in Greenbelt, Maryland, will monitor the vehicle's health and safety 24 hours a day using three shifts of operators. There are almost 5,000 different telemetry monitors in 11 different possible formats. Each format has a subset of monitors available in it, and the rate at which a specific monitor is reported varies from 40 hertz to 0.025 hertz. Execution of on-board stored program commands (SPC) are handled by a 40 hertz processing rate making it impossible for the operator to watch individual command executions.

Lockheed's Artificial Intelligence Center in Menlo Park, California, has been working on developing a real-time monitoring tool called *L*STAR* (Lockheed Satellite Telemetry Analysis in Real-Time). This knowledge-based monitoring system is still under development with initial rule bases being centered on the Pointing and Control System (PCS), the Data Management System (DMS), and the Electrical Power System (EPS). This paper will address the PCS application of the *L*STAR* system. The PCS application, while far from completion, has already led to many valuable insights.

Requirements

The real-time requirements of the PCS include the following: command verification, safemode prevention warnings, configuration validation, performance assessment, and configuration monitoring.

Command verification is the process of checking a new command against the configuration of the spacecraft prior to the sending of that command. This is to ensure that the command will not endanger the vehicle or interrupt the current mission. Verification would include checks to prevent commands that would expose a scientific instrument aperture to a bright object such as the Earth or the Sun, for example.

Safemode prevention warnings are messages to the operator that indicate a dangerous condition developing and show the autonomous safety system checks done by the HST flight computer that will initiate a response if ground action isn't taken. The response by the flight computer's safemode system can range from shutting down subsystems to shutting the whole flight computer down and passing control of the vehicle over to the PSEA (Pointing/Safing Electronics Assembly). Almost all responses by the safemode system abort the current command list and require recovery by the ground system. The safemode prevention warnings are intended to warn the ground so that either the safemode response can be avoided or anticipated.

Configuration validation checks compare the modes of various subsystems and ensure that they are compatible. Certain states should never occur simultaneously. For example, the attitude of the vehicle is not available for computations when the vehicle is in Drift mode (used in deployment and recovery from safemode) or trying to align with the Sun (Sun Point Control). Computations such as the Momentum Management's Minimum Energy Law which predicts the momentum profile for the next half orbit using the current attitude should not be active during either of those two states.

Performance assessment allows the operator to know in real-time how successful the planned mission is. For example, at each point in the mission, there is an anticipated inaccuracy in the vehicle attitude. After an attitude update by the Fixed Head Star Trackers (FHST), the error should be less than ten arcseconds, and after establishing lock on stars with the Fine Guidance Sensors (FGS), the error should not exceed the radius of the search used to find the stars. The operator would be warned if reported attitude errors did not meet the expectations for any given time.

Configuration monitoring simply reports, to the operator, changes in the mode of any subsystem. This includes unplanned changes such as those resulting from unexpected loss of lock on the stars being used for guidance of the vehicle.

Organization of System

L*STAR uses rules from its knowledge base to intelligently monitor the HST telemetry stream. So that the system does not have to examine the entire ruleset, each rule has certain contexts in which it is valid. The rule will not be examined or triggered if its context does not match the current context of the inferencing system. For instance, a rule to check the performance of a vehicle maneuver does not need to be tested during Drift mode. Rules may be applicable in a single context or multiple contexts. The context of the inference engine is at all times identical to the mode of the HST. The mode of the HST is an enumerated attribute with currently ten legal values: *Drift, Inertial Hold, Science, Sun Pointing Control, Loss of Lock, Attitude Hold, Maneuver, FGS Acquisition, FHST Acquisition, Mechanism Motion*, and *PSEA*.

Each flight computer software subsystem (total of 13) is a different class (i.e., schemata or frame type) with unique attributes. They all have at least two attributes called Status and Mode. Status is typically either **normal** or **abnormal** and Mode has values that are specific to the subsystem.

A natural way to organize the rules that will potentially number in the thousands was to put each subsystem's rules in separate files. In each file the rules are titled with a number assigned (similar to the Dewey Decimal System) as shown in Table 1. In some cases the rules may fit two categories, in which case the lower number is used.

```
1.x - determines subsystem mode
2.x - determines subsystem variable attributes
3.x - context dependent limit checking
4.x - checks for invalid mode switch or illegal
      variable attributes
5.x - outputs messages to operator
```

Table 1 - Rule Number Convention

A sample rule for testing to ensure that speed of reaction wheel 1 is normal for certain contexts would look as follows:

```
RULE      : "[3.1] Reaction Wheel 1 Check";
CONTEXT   : { Inertial_Hold, FGS_Acq, FHST_Acq, Science,
              Mechanism_Motion };
PRIORITY  : 100;
AUTHOR    : "Simon Kao/Larry Dunham";

IF   ([value\monitor\QDVOMEVO] > 12.0) (*radians/sec*)
THEN [status\momentum_management\MOMAN] := abnormal;
     send( IO, ALERT, "QDVOMEVO", "Wheel speed 1 high at
                                   %[time\satellite\HST]");!
```

An identical rule is needed for each of the other three reaction wheels. This need for vector notation is common in satellite telemetry systems. Many monitors are position vectors in the vehicle frame, or are sets of values for identical hardware (6 gyros, 4 wheels, etc.). Facilities for processing and manipulating vectors is generally not available in commercial AI tools. This capability is being added to *L*STAR* and should lead to a significant decrease in the total number of rules.

Temporal Reasoning

Commercial AI tools have few, if any, capabilities for reasoning about past, present, and future events. For satellite telemetry monitoring it is a necessity to have such capabilities. *L*STAR* has implemented them as built-in functions of the inference engine. Temporal reasoning in the simplest form is the ability to use trends and statistics in rules via functions

such as rate-of-change and average value over a time period, for example. This makes it straightforward to write rules based on the vehicle error decreasing or the commanded body rate being steady.

The more complex part of temporal reasoning involves the difficulties of handling data from telemetry with various time tags associated with them. The monitors are reported at various rates and at various times. For example, if telemetry flags A and B cannot be true at the same time, the fact that the last reported values are both true should not fire a rule for illegal configuration. A or B may have just been reported true and the other value has not been reported since that time, and the system needs to wait for the next sample of the second monitor and discover if it is reported as still being true. If it is, then the assumption (which in some cases is not valid) is that if one of the monitors was reported true, both before and after the time that the other one was reported true, then at some point in time they must have both been true simultaneously. However, for fast changing monitors with slow reporting rates, even this logic is faulty.

An example rule is that the Minimum Energy Law should be off when the vehicle mode is in Sun Point control. The vehicle mode flag changes very slowly, so that the user is assured that two identical samples ensure a constant mode over that time period. If the vehicle mode were reported twice as frequently as the Minimum Energy Law flag, then the Table 2 shows the valid and invalid telemetry patterns.

	valid sequences					invalid sequences		
t = 1.0 ME sample	on	on	on	off	off	on	on	off
t = 1.01 SP sample	off	off	on	on	on	off	off	on
t = 1.06 SP sample	off	on	on	on	off	off	on	on
t = 1.1 ME sample	off	off	off	on	on	on	on	on
t = 1.11 SP sample	on	on	on	off	off	on	on	on

Table 2 - Valid/Invalid Sequences for Temporal Reasoning

A rule prohibiting ((ME on) AND (SP on)) would have prohibited three of the valid states. The *L*STAR* Inference Engine has an AND function being added to it to handle rules of this type properly. This need is common in all telemetry sampled systems. Tools which process rules based only on change data (e.g., an OPS5 based on the Rete network) cannot handle these types of Relationships

*L*STAR* User Interface

The I/O Process, which can run on its own processor, provides sophisticated displays that help both the console operators and the analysts. The operator is provided with messages

that are either information, alerts, or warnings. These messages are stored and recallable using the mouse.

For the PCS, a set of hierarchical displays of the flight computer software is done in great detail. The diagram shows the relationships of the various monitors. Each monitor can be plotted on the screen in real-time by simply mousing the monitor name on the diagram. The analyst can then track problems backward using the diagram to determine the initial monitor that indicates abnormal behavior.

Conclusions

The *L*STAR* system has been proven to be able to handle real data from HST tests and perform the monitoring in real-time. The multi-processor design allows for multiple inference processes to be distributed to additional processors if the rule-base becomes unmanageable in real-time for one processor.

The insights into the problems of satellite telemetry systems with regard to easy vector notation and temporal reasoning have shown commercial tools severely lacking. *L*STAR* is an attempt to fill that niche. Both insights and answers have been gained by having a team consisting of personnel from the LMSC AI Center, the HST flight software development group, and the ground system operations group.

This system is currently under development and is being used to monitor test cases produced by the Bass Telemetry System in the Hardware/Software Integration Facility at Lockheed Missile and Space Co. in Sunnyvale, California.

Acknowledgements

The authors wish to acknowledge the encouragement and support received from Wally Whittier, LMSC HST S/W Program Manager in Sunnyvale, California.

N88-16379

TALOS: A DISTRIBUTED ARCHITECTURE FOR INTELLIGENT MONITORING AND ANOMALY DIAGNOSIS OF THE HUBBLE SPACE TELESCOPE

Bryant G. Cruse
Goddard Space Flight Center
Greenbelt, ML 20771

ABSTRACT

Lockheed, the Hubble Space Telescope Mission Operations Contractor, is currently engaged in a project to develop a distributed architecture of communicating expert systems to support vehicle operations. This architecture, called TALOS for Telemetry Analysis for Lockheed Operated Spacecraft, has potentially wide applicability to spacecraft operations. The architecture mirrors the organization of the human experts within an operations control center. Initial development consisted of building a successful prototype that functions within the existing HAT ground system environment at Goddard Space Flight Center. The prototype analyzes telemetry from a history tape to determine the state of the vehicle with respect to on-board safemode events. The prototype is currently being expanded along two fronts. One consists of expert system modules which perform deeper-level diagnostics and which operate in an off-line mode. The second is a high-speed front-end expert systems which will monitor real-time telemetry for anomalies and spacecraft status and send activation and initialization messages to the off-line modules when an anomaly is detected.

Design and implementation of practical expert system applications to support a complex spacecraft like the HST has posed a number of challenges. Choice of expert systems tools to support the project has projected a problem. Selection is constrained in the first place by the requirements of hardware and software compatiblity with the existing HST ground system. Further, no off-the-shelf expert system shell was found to have the necessary performance to support real-time analysis of the HST's engineering telemetry stream. These factors have led to the selection of two separate tools currently in development at the Lockheed Artificial Intelligence Center. One has the necessary speed to support real-time analysis and the other has the flexibility required to diagnose a wide range of anomalies. The requirements of TALOS have been a significant driver for the development and refinement of these two tools. The real-time monitoring tool will be discussed in another paper at this conference.

The TALOS architecture will be described and the unique aspects of the project will be discussed. Current status of the project will be reviewed.

N88-16380

A KNOWLEDGE-BASED SYSTEM FOR MONITORING THE ELECTRICAL POWER SYSTEM OF THE HUBBLE SPACE TELESCOPE

BY

PAT EDDY

LOCKHEED ARTIFICIAL INTELLIGENCE CENTER
2710 SAND HILL ROAD, MENLO PARK, CA. 94025

ABSTRACT

Lockheed is currently in the process of developing expert systems that perform on-line monitoring, statusing and trend analysis of major Hubble Space Telescope (HST) systems. The three major systems under development are the Pointing Control System (PCS) [KLSRD87], the Data Management System (DMS) and the Electrical Power System (EPS). These expert systems are part of the TALOS [CW87] system for assisting in HST ground operations which is being developed at Goddard Space Flight Center. This paper will treat the EPS expert system (a part of the TALOS distributed architecture).

This paper will describe the design and the prototype for this system as demonstrated on 9/23/87 at the LMSC Artificial Intelligence Center, Menlo Park, California. This prototype demonstrated the capability to use real time data from a 32k telemetry stream and perform operational health and safety status monitoring, detect trends such as battery degradation and detect anomalies such as solar array failures. This prototype along with the PCS and DMS prototype expert systems form the initial TALOS capability.

INTRODUCTION

Lockheed Missiles and Space Company (LMSC) is the prime contractor for the Support Systems Module (SSM) and Integration Systems Engineer for the Edwin P Hubble Space Telescope (HST). Additionally, LMSC is the HST Mission Operations Contractor, responsible for the ground operations that ensure the health and safety of the HST.

The HST is a complex, state-of-the-art satellite with unique ground operations requirements. Accordingly the ability to reliably monitor telemetry in real-time is highly desirable. To this end the Telemetry Analysis Logic for Operating Spacecraft (TALOS) system is being developed.

The Talos system consists of real-time and off-line deep analysis knowledge-based modules. This paper will concentrate on the real-time monitoring and analysis EPS monitoring system. The basis for this system is L*STAR, a Lockheed proprietary tool. This shell [SRLK87] consists of a multiprocessing architecture for performing real-time monitoring and analysis using knowledge-based problem solving techniques.

In order to handle asynchronous inputs and perform in real-time, the system consists of three separate processes which run concurrently and communicate via a message passing scheme. The Data Management Process gathers, compresses, and scales the incoming telemetry data before sending it to the other tasks. The Inferencing Process consists of a proprietary high performance inference engine, written in the C programming language, which can run at a rate of close to 1000 rules per second. It uses compressed telemetry data to perform a real-time analysis on the state and health of the Space Telescope. The I/O Process receives telemetry monitors from the Data Management Process and status/health messages from the Inference Process, updates its graphical displays in real-time, and acts as the interface to the console operator. The I/O Process resides on DEC VAXStation II/GPX high resolution color graphics workstation. It consists of a hierarchy of displays which the user may traverse using a mouse. The three tasks run concurrently and may reside on the same or different processors. Furthermore, the multitasking architecture has been designed in such a way that multiple inference processes, multiple data management processes, and multiple I/O processes can run concurrently and communicate with each other. It processes several hundred telemetry monitors per second.

The population of L*STAR with HST EPS knowledge has produced a very fast and reliable operator assistant to be described in this paper.

EPS REAL-TIME TELEMETRY MONITOR

The TALOS EPS real-time telemetry monitor will provide operators with the following real-time telemetry capability:

- HST Mode/Configuration Monitoring.
- Real-time EPS Statusing.
- Command Verification.
- Multi Orbit Health History Tracking.
- Ability to Handle Unexpected Telemetry Loss.
- Correlation of Temperature, Power Error Histories.
- Real-time warning of approaching intelligent thresholds.
- Real-time comparison of an EPS model generated power profile with telemetry based EPS status.
- Projection of telemetry based EPS status in accordance with load and time parameters defined by a model generated power profile.
- Suggested load changes to prevent exceeding limits.

One of the key elements required for this system to be able to reason about the health and safety of the HST is knowledge about the present and projected future operational mode of the HST. Accordingly, this system establishes the following contexts for the reasoning process:

- Battery Reconditioning
- Off Normal Roll
- Stationary HST
- Maneuvering HST
- Orbit Day
- Eclipse
- Hardware/Software Sunpoint
- Solar Array Repositioning
- Seasons Of The Year
- Orbital Decay

This system provides the operator with a top level functional diagram of the end to end power flow of the EPS starting with the Solar Arrays and ending with the Power Distribution Units. Lower level displays show more detailed diagrams of EPS components until all telemetry values are accounted for. The EPS statusing capability consists of the following:

- EPS Configuration
- SA power output
- Status of K relays
- Structure Current
- Battery voltage/temperature
- Battery Recharge Ratio
- Diode Bus Current/Voltage
- Sun to Orbit Ratio
- Power Distribution Unit Current
- Battery Shunt CKT Status
- Solar Panel Assembly Connect/Disconnect
- Distribution Bus On/Off
- Heater Status
- Voltage VS Temperature Change Current Controller Activation Curves

The following capabilities will provide an EPS real-time telemetry monitoring assistant that uses temporal reasoning and communicates with other expert systems:

- Command verification will allow the automatic indication of command group execution on a display of the mission timeline. In addition, the EPS expert system will receive indication of a command reject from the DMS real-time monitor and such information as maneuver status from the PCS monitor.

- Multi-orbit health, history tracking will display and compare a 15 orbit history set of events such as the following:

  - Trickle Charge Time
  - Maximum battery Voltage at Eclipse
  - Minimum Battery Voltage at Eclipse

o Management of unexpected telemetry loss is necessary to support multi-orbit event tracking above. It will be accomplished by designating a beginning/end of orbit event that occurs at a predictable time such as K-relay closure. If the required data has not been collected by the end of orbit then that orbit will be shown as missing data and the data collection process will be reinitialized for the beginning of the next orbit.

o Correlation of temperature, power and error histories will allow the operator to compare any multi-orbit health histories for the purpose of identifying failure trends.

o Intelligent threshold warnings consits of monitoring real-time EPS telemetry to provide operator warnings of approaching anomalies as a result of reasoning about the current operational mode on future loads.

o EPS status and power profile comparison compare an EPS model generated power profile with the actual EPS to show a projected status based on the load as defined by the EPS model. The operator may project the power status to a number of orbits or hours in the future.

o Projection of load and time changes will allow the system to obtain the results of proposed load and time changes by extrapolating the telemetry based EPS status into the future. In addition, the expert system will have access to the projected electrical activity as defined by the power profile and will be able to warn the operator of HST components or science instruments that are scheduled to come on and approach load limits.

o Suggested load changes will give the operator access to the projected loads created by HST components in their operational configurations. When the power profile deviates from the current EPS status the system will be able to access a load shedding component list and make recommendations to the operator.

ACKNOWLEDGEMENT

This work was done under the sponsorship of Tom Laffey of the Lockheed Artificial Intelligence Center.

CONCLUSIONS

The work accomplished to date provides the operator with an intelligent assistant that reasons about current system status such as the following:

- Battery Voltage
- Charge Current
- Bus Voltages and Current
- Component Temperatures
- Solar Array Output

It then advises the operator whether the above parameters are normal in accordance within the following contexts:

- HST maneuvering
- HST Stationary
- Solar Arrays repositioning
- Orbit Day
- Eclipse

Thus, this system advises the operator that a given current is normal, not just within limits, while taking into account maneuvering status and time of day. Future work will add safemode, seasons, and orbital decay to this reasoning process.

The above information is presented in meaningfull displays showing the end to end energy production and usage along with lower level detailed displays. Warnings and adviseries are provided. This work is on-going and future work includes completing the requirements listed in this paper.

REFERENCES

[CW87] B. Cruse and E. Wende. Expert system support for HST operations. In Proceedings of the 1987 Goddard Conference on Space Applications of Artificial Intelligence and Robotics, NASA Goddard Space Flight Center, May 1987.

[KLSRD87) S. Kao, T. Laffey, J. Schmidt, J. Read, L. Dunham. Monitoring the PCS of the Hubble Space Telescope. Third Conference On Artificial Intelligence for Space Applications, George C. Marshall Space Flight Center, November 1987.

[SRLK87) J. Schmidt, J. Read, T. Laffey, S. Kao. A Multiprocessing architecture for real-time monitoring. Third Conference On Artificial Intelligence for Space Applications, George C. Marshall Space Flight Center, November 1987.

N88-16381

ARTIFICIAL INTELLIGENCE AND SPACE POWER SYSTEMS AUTOMATION

David J. Weeks
NASA/MSFC
EB12
Marshall Space Flight Center, AL 35812

Abstract

This paper will discuss various applications of artificial intelligence to space electrical power systems. Completed, on-going, and planned knowledge-based system activities will be overviewed. These applications include NICBES (the expert system interfaced with the Hubble Space Telescope electrical power system test bed and one of the few NASA expert systems in daily operational use; the early work with SSES; the three expert systems under development in the Space Station advanced development effort in the core module power management and distribution system test bed; planned cooperation of expert systems in the CM/PMAD breadboard at MSFC with expert systems for Space Station at JSC and LeRC; and the intelligent data reduction expert system under development.

Background

The size and complexity of spacecraft power systems are increasing dramatically. America's first space station, Skylab, employed an eight kilowatt power bus. From fifteen to twenty ground support personnel were required to monitor and control the electrical power system on this early space station. At times, extensive crew involvement was necessary to correct system faults. [3] [4]

The Electrical Power Branch at Marshall Space Flight Center has been involved since 1984 with the development of expert or knowledge-based systems to facilitate the automation of electrical power systems. The expert systems developed thus far are focused on fault diagnosis and contingency payload scheduling. Systems now under development address comprehensive fault management, automatic rescheduling, and intelligent data reduction. Future plans involve the development of expert systems for battery management, trends analysis, and component failure forecasting. [5]

Several expert system prototypes have been developed including: the Hubble Space Telescope electrical power system test bed diagnoser /analyzer named NICBES (NIckel-Cadmium Battery Expert System), the Space Station Experiment Scheduler (SSES), and the Loads Priority List Management System (LPLMS). Other current and proposed automation activities will also be briefly discussed.

Hubble Space Telescope Expert System

The Nickel-Cadmium Battery Expert System (NICBES) is interfaced with the Hubble Space Telescope electrical power system test bed at the Marshall Space Flight Center. A functional block diagram is shown in Figure 1.

As presently configured, this breadboard is operated continuously and automatically telephones the test bed personnel at work at home in the event of a test bed anomaly. When these personnel arrive at the test bed site, they troubleshoot the system and take any steps necessary to restore the system to full operational status while protecting critical flight-type components in the test bed.

NICBES has three major functions in addition to the collection and storage of telemetry data from the test bed. The first function or mode is fault diagnosis. NICBES will independently verify the occurrence of an anomaly and recommend appropriate corrective actions.
The second mode is status and advice. NICBES will evaluate the status of each battery in the test bed (there are six, 23-cell flight-type batteries) and give advice concerning each battery.

The third mode is the decision support graphics which offers 12 plots for any of the six batteries in the system. These plots display summary data for the 12 previous simulated orbits.

PRECEDING PAGE BLANK NOT FILMED

ORIGINAL PAGE IS
OE POOR QUALITY

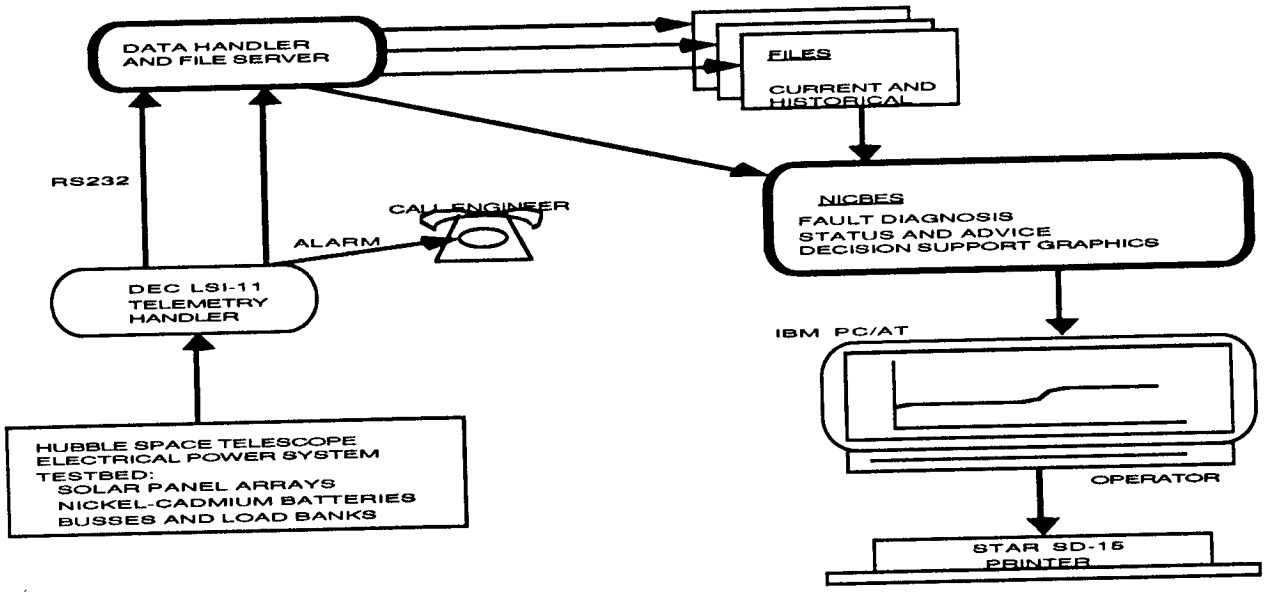


FIGURE 1. NICBES Functional Block Diagram

NICBES was developed by the Martin Marietta Denver Aerospace Corporation in Denver, Colorado under contract to NASA/Marshall Space Flight Center. [1] The expert system is implemented on an IBM PC/AT in Prolog.

Contingency Payload Scheduler

The Space Station Experiment Scheduler (SSES) is a proof-of-concept demonstration prototype expert system which schedules/reschedules payloads very quickly. Although SSES only employes about a few of the scheduling constraints that the Marshall Space Flight Center Experiment Planning System considers in Spacelab mission planning, it can reschedule 50 payloads for a full two week period in less than three minutes. It considers power consumption, payload duration, intermittent usage, crew attendance required, and priority class. Though a relatively simple model, this expert system demonstrates that a dynamic rescheduler embedded in a spacecraft power management system can help handle perturbations to the available power, assuring that power is utilized as it becomes available as well as avoiding the unnecessary load shedding of critical payloads in the event of a reduction of the available power. [2]

Space power has historically cost about $1000 per kilowatt hour versus $0.05 per terrestrial kilowatt hour. On Space Station maximum utilization of available power will be necessary to accommodate all the experiments and other payloads on board. In the event of reduction of available power, it is imperative that a critical load is never shed unless absolutely necessary. A Department of Defense or European Space Agency payload might be critical for national defense or due to international agreements; a science experiment may have a critical window for operation; or a materials processing payload may increase in importance as an expensive crystal nears completion and cannot be interrupted without incurring flaws.

The SSES was developed in 1985 by Technology Applications Inc. of Jacksonville, Florida as part of the Space Station Advanced Development program. It is implemented in Large Memory GC LISP on an IBM PC/AT with 3.5 megabytes. The graphics were coded in Turbo Pascal.

ORIGINAL PAGE IS
OF POOR QUALITY

Space Station Core Module Power Management and Distribution System Automation Project

One of the most ambitious automation projects at the Marshall Space Flight Center is the Space Station Advanced Development effort for automating the Core Module Power Management and Distribution (CM/PMAD) system. The CM/PMAD breadboard will employ three expert systems in addition to extensive conventional automation software to control the power system breadboard as shown in Figure 2. The systems autonomy is pushed down as far as possible in the system such that in the event of an automation breakdown at the lower levels, the next higher levels will assume responsibility of the components below them in the hierarchy.

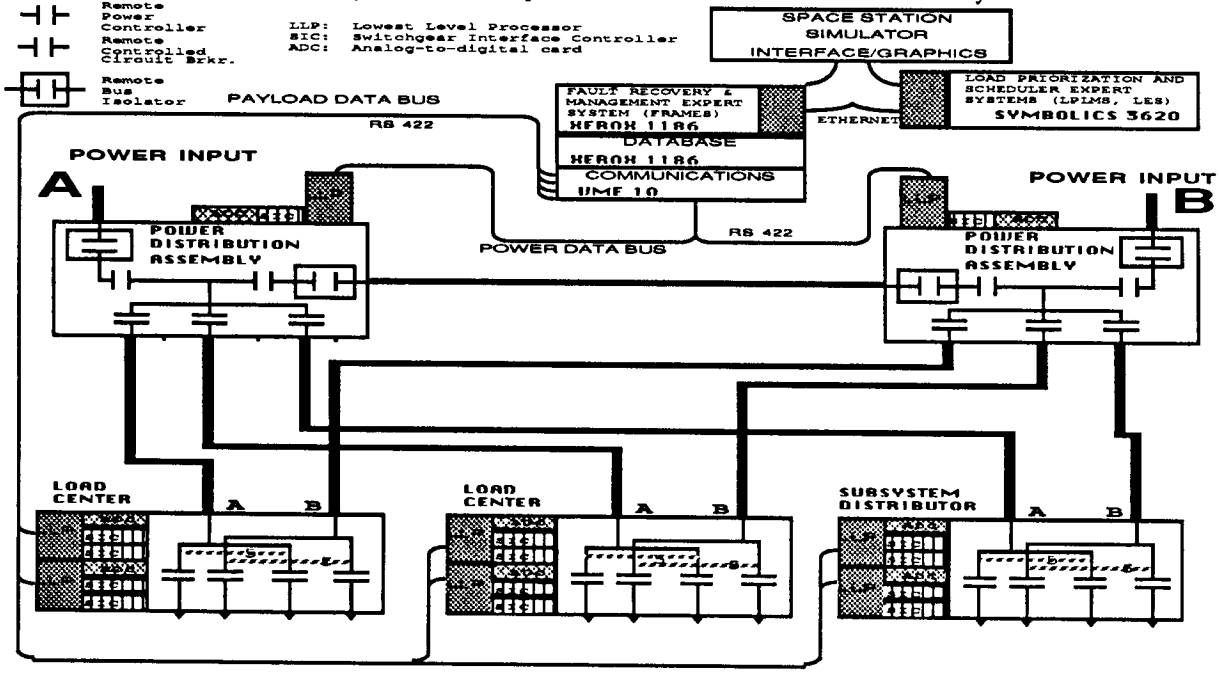


FIGURE 2. MSFC SPACE STATION MODULE ELECTRICAL POWER SYSTEM BREADBOARD

The first of the expert systems is the Load Priority List Management System (LPLMS). The LPLMS maintains a real time dynamic representation of all the module loads and relevant facts so applicable rules can fire to reorder portions of the list as situations change.

The loads in a laboratory module may have dynamic priorities. A critical noninterruptible materials processing experiment involving crystal growth may have a different priority as its nears completion. Other factors may change priorities such as equipment malfunctions. An expert system such as LPLMS is critical in order to determine which loads must be shed in the event of perturbations to available power for the module. It is imperative that the 'critical' loads are not shed unnecessarily.

The LPLMS is currently implemented in the production language HAPS on the Symbolics 3600 series workstation. Load priority lists are sent down from the expert system to the conventional processors in the breadboard every 15 minutes.

LES, the Load Enable Scheduler, can schedule and reschedule a number of payloads with various scheduling constraints. This expert system will generate the baseline schedules for the breadboard as well as accept information from the other processors on when and how to reschedule the power system payloads. LES refines the ground load enable requests and builds a detailed mission activities list through simulation performance and list passing to the Supervisor Subsystem Simulator (SSS).

FRAMES is the Fault Recovery And Management Expert System. This expert system watches over the entire breadboard operation looking for anomalies and impending failures. FRAMES functionality actually extends to the lowest level processors in the breadboard for comprehensive fault management of the entire breadboard.

FRAMES is responsible for detecting faults, advising the operator of appropriate corrective actions, and in many cases, autonomous corrective action implementation through power system reconfiguration. The expert system will carry out trends analysis seeking incipient failures and soft shorts as well as open circuits.

The more conventional automation software resides in the power control unit (PCU) and in the lowest level processors (LLP). The PCU resides on a VME/10 system and performs process database updates on system data such as the load time enable, load time disable, primary remote power controller (RPC) number, secondary RPC number, switch permission number, nominal maximum power value, upper voltage limit value, lower voltage limit value, upper current limit value, lower current limit value, actual snapshot voltage value, actual snapshot current value, actual snapshot RPC total power value, and actual snapshot power value. The PCU also generates the process command list, allocates individual LLP command lists from the command lists generated by the Supervisor Subsystem Simulator (SSS) to the lowest level processors, allocates individual LLP priority lists generated from the overall priority list developed by the LPLMS to the LLPs, maintain the state of the actual load system within the present configuration list, graph the total power output as a function of time per bus, maintain a failed components list, maintain a major events list, provide a user interface for switching loads in and out (for emergency use where the user assumes full responsibility for the breadboard operation), function as the central communications device, allocate database change information to the SSS, and execute emergency shutdown procedures.

The lowest level processors maintain their individual command and priority lists, execute command control as directed by the individual command list, execute local load sheds as directed by their individual priority lists to save the bus from bad scheduling or overloading, execute data compression, execute data reporting, handle condition exceptions, execute automatic switch control, execute list directed switch control, verify configuration limits and allocations, and execute immediate commanded switch control. These processors are 68010 microprocessor-based controllers.

The SSS simulates various module interfaces such as to other subsystems, other elements of the overall Space Station electrical power system, the crew, and ground support elements. It resides on a Xerox artificial workstation though it is not itself an expert system.

Together these various elements comprise a fairly elaborate approach to power system automation. It is anticipated that this advanced development effort will contribute to the actual core module power management and distribution system automation on-board the Space Station. Martin Marietta is providing the contractor support for this system development. [6]

Marshall Space Flight Center is also cooperating with the Lewis Research Center in a 1990 Power Systems Autonomy Demonstration. This project involves the entire Space Station electrical power system test bed at two NASA Centers cooperating with the Space Station thermal control system test bed at NASA's Johnson Space Flight Center. Ames Research Center has overall responsibility for this project under the Systems Autonomy Demonstration Program.

Intelligent Data Reduction

The IDARE (Intelligent DAta REduction) project is a research effort involving the capture of the facts and heuristics that battery system specialists employ in determining the significant components of battery telemetry data. The research will be directed toward the Hubble Space Telescope power system test bed telemetry data and is expected to result in a knowledge-based system which autonomously reduces this telemetry data to its significant components for further trends analyses, improved system state-of-health monitoring, and fault prediction.

It is estimated that at any given time, perhaps 80 to 95 per cent of battery telemetry data is insignificant. It is hoped that the telemetry data can be reduced by an order of magnitude. As such data reduction is extremely labor intensive, an expert system would greatly reduce testing analysis and operational support.

The IDARE project is being conducted in cooperation with the University of Alabama in Huntsville under an university grant. If successful, applications are expected for complete elec-

trical power systems as well as other subsystems with propulsion personnel showing a special interest.

Conclusions

Knowledge-based or expert systems are being demonstrated for electrical power system applications involving proof-of-concept prototype intelligent systems. Artificial intelligence approaches should not replace conventional computer programs that work well. Instead, these knowledge-based systems should be employed to fill the gaps where traditional approaches either perform poorly or cannot be employed at all.

If autonomous electrical power systems are to be incorporated on future spacecraft, knowledge-based system prototypes must continue to be developed and demonstrated in fairly realistic electrical power system breadboards and test beds. Program managers must be convinced that these systems are safe, reliable, and can be developed within cost in a timely fashion.

References

1. Kirkwood, N. and D.J. Weeks, "Diagnosing Battery Behavior with an Expert System in PROLOG," Proceedings of the 21st IECEC, San Diego, CA, paper 869410, August 1986, pp. 1801-1807.

2. Touchton, R.A., "Common Module Dynamic Payload Scheduler Expert System," Proceedings of the 21st IECEC, San Diego, CA, paper 869406, August 1986, pp. 1785-1790.

3. Weeks, D.J. "Application of Expert Systems in the Common Module Electrical Power System," SPIE Vol. 580 Space Station Automation, Cambridge, MA, September 1985, pp. 35-39.

4. Weeks, D.J. and R. T. Bechtel, "Autonomously Managed High Power Systems," Proceedings of the 20th IECEC, Miami FL, paper 859157, August 1985, pp. 1.132-1.138.

5. Weeks, D.J. "Expert Systems in Space," IEEE Potentials, Vol. 6, No. 2, 1987, pp. 18-21.

6. Weeks, D.J. "Space Power System Automation Approaches at the George C. Marshall Space Flight Center," Proceedings of the 22nd IECEC, Philadelphia, PA, paper 879104, August 1987, pp. 538-543.

N88-16382

EMBEDDED EXPERT SYSTEM FOR SPACE SHUTTLE MAIN ENGINE MAINTENANCE

BY:

J. Pooley, W. Thompson, T. Homsley, and W. Teoh, PhD

SPARTA, Inc.

4901 Corporate Drive

Huntsville, Alabama 35805

J. Jones and P. Lewallen

NASA Marshall Space Flight Center

Huntsville, Alabama

ABSTRACT

Space Shuttle Main Engine (SSME) maintenance, whether preventive, scheduled, or unscheduled, is a major escalating cost item. Significant progress has been made in the NASA and Air Force communities toward performance of the health monitoring function in instrumentation, analysis techniques, and envelope (trends and rate of change) monitoring. Current techniques require that domain experts be integrally involved in the analysis session and make on-line decisions to direct analysis. The SPARTA Embedded Expert System (SEES) is an intelligent health monitoring system that directs the analysis by placing confidence factors on possible engine status, then recommends a course of action to an engineer or the engine controller. This technique can prevent catastrophic failures or costly rocket engine down time because of false alarms. Further, the SEES has potential as an on-board flight monitor for reusable rocket engine systems. The SEES methodology synergistically integrates vibration analysis, pattern recognition, and communications theory techniques with an artificial intelligence technique - the Embedded Expert System (EES). This integration affords a robustness via the analysis techniques with an ability to resolve conflicts by the expert system techniques.

INTRODUCTION

A critical element of the Space Shuttle Main Engine (SSME) program is the development of a turbo-pump health monitoring system (HMS). A HMS that could predict incipient failures and permit routine maintenance to be scheduled based on performance indicators would dramatically reduce the need for refurbishment, improve equipment availability, and make maintenance more cost-effective. The key functions of an effective HMS are shown in Figure 1.

- **RECOGNIZE AND CATEGORIZE PERFORMANCE**
  (Baselining Of Performance Standards)
- **RECOGNIZE AND CORRELATE INDICATORS OF IMPENDING FAILURE**
  (Incipient Failure Prediction)
- **RECOGNIZE AND CORRELATE INDICATORS OF NEED FOR REMEDIAL ACTION**
  (Scheduling Of Routine Maintenance In A Cost-Effective Manner)

Figure 1. HMS ESSENTIAL FUNCTIONS

PRECEDING PAGE BLANK NOT FILMED

Significant progress has been made in the NASA community toward performance of the HMS functions. There have been relevant advances in instrumentation [4,1], analysis techniques [2,5], and in detection of anomalies and failures [3]. Each of these advances has demonstrated individual attributes useful for an HMS to correlate failure modes with turbo-pump components at risk. However, an integrated HMS that uses and updates the SSME data base is possible through the use of emerging AI techniques. AI techniques, specifically a rule-based expert system, can enhance the functions of an HMS. SPARTA has developed and adapted a set of algorithms to produce an innovative application of Artificial Intelligence techniques. The keystone of this application is an expert system that uses confidence levels to resolve conflicts among compound data, and that heuristically trains on each data set to derive (or modify) classification rules. This expert system has been named SEES, an acronym for SPARTA Embedded Expert System.

SEES ARCHITECTURE

The SEES architecture is shown in Figure 2. In SEES, conventional computation methods are used to reduce the raw data to a manageable "derived" data set, and to extract pertinent information (signatures) from the derived data set. This information is then used by the SEES to derive rules, with the help of domain experts/knowledge engineers, to establish a knowledge base. In future phases, SEES will use this set of rules to determine engine conditions during SSME testing.

MAJOR COMPONENTS

As can be seen from the architecture in Figure 2, there are three major subsystems to the SEES HMS: The SEES front end (SFE), the embedded expert system (EES), and the supportfunction library (SFL). The SFE processes the raw data to screen obvious anomalies and to derive the reduced data set, then generates from it an appropriate signature. The process of data screening, reduction and signature generation is the unique and proprietary innovation of SEES. The embedded expert system (EES) uses this signature and the reduced data, with the help of the SFL and the rule set in its knowledge base, to infer the operating conditions at a given instant, deduce the mean time to failure and recommend maintenance schedules. The SFL, as its name implies, is a set of supporting functions for the rest of the HMS.

SEES FRONT END (SFE)

The SFE is comprised of signal analysis techniques that convert raw count accelerometer data to Engineering Units and transform the data to the frequency domain using Fast Fourier Transforms (FFT) to derive a power spectral density (PSD) for input to a Data Conditioning Module. The Data Conditioning Module processes the PSD signal to remove the extraneous components. Finally, the conditioned PSD is evaluated as a candidate for signatures derived during this processing (by the Pattern Matcher) or binned to be considered for establishment of another signature.

SEES SUPPORT FUNCTION LIBRARY (SFL)

The SFL consists of the algorithms that transform reduced data into symbol structures for use by the Development Engine and/or the Inference Engine to accomplish inference and control. This transformation is accomplished by applying communications theory and image processing methods to the SFE conditioned data.

SEES EMBEDDED EXPERT SYSTEM

The Embedded Expert System (EES) is an integral part of the HMS. The EES is a rule based knowledge system that uses forward chaining strategy, and has the ability to recognize and categorize performance, incipient failure and the need for remedial action. It consists of a development engine, a knowledge base, an inference engine, and a user interface.

The Development Engine

The development engine is a subsystem of EES which is intimately related to the knowledge acquisition process; it allows a knowledge engineer to transcribe the knowledge gained from the domain expert into a set of rules that make up the knowledge base. A basic characteristic of the SEES problem in analyzing SSME vibration data is the volatility of signatures and the importance of high rate vibration data. While some rules can be developed, the evaluation of data in real-time leads to the requirement to merge information from multiple sources. This leads to the use of the blackboard architecture for storing intermediate hypotheses, the use of a certainty factor merging heuristic and rule-use counting as a "rule-critic".

The SEES Knowledge Base

Based on SPARTA's study of the training sets, we expect the knowledge base to be quite large. Our investigation indicated that signatures may be extracted and meaningfully classified. Thus, the rule set may be ordered in an appropriate manner (e.g., a rule tree) to reduce search space. The nature of the data is such that one can seldom specify a diagnosis with absolute certainty. Thus in SEES, certainty factors will be used to reflect uncertain information. These certainty factors can be either computed algorithmically by the development engine based on derived or existing knowledge, or estimated by domain experts or knowledge engineers.

The SEES Inference Engine

The SEES HMS is basically data driven. Thus, a forward chaining strategy is appropriate. The incoming data, although reduced by the SFE, is still quite complex, and entering the HMS at a high rate. The EES inference engine must and will have the capability to invoke functions in the SFL for further data reduction. Perhaps one of the more important tasks of the inference engine is to determine when an unknown situation (i.e., not in the rule base) occurs. It should be able to coordinate with the domain expert or knowledge engineer and pass the new information to the development engine to create new rules, or store the information to accomplish the same at a later time off-line when a domain expert is available. The inference engine must provide information to the explainer to produce explanation on demand. The explainer is a subsystem of the user interface and can provide explanations as to how a conclusion is reached. This can be accomplished in a variety of ways; the one selected by SPARTA is to leave the time history of SEES events in the blackboard for post mortem examination.

SEES User Interface

The user interface is the component of an expert system that acts as an interface between the expert system and the user who is not necessarily a domain expert. Thus, it should have the capabilities to: (1) Solicit input from the user, (2) Provide output to the user - this output may be in the form of questions, recommendations or conclusions, and (3) Provide explanations on demand. One important aspect that can be implemented is a possible data link between the user interface and the SSME controller. This would serve as a means to assume control of the engine in unusual situations, such as when an imminent engine failure has been detected, and an immediate engine shutdown becomes necessary to prevent catastrophic failure. This feature is, of course, not needed for off-line testing.

IMPLEMENTATION

Preliminary investigation to date has been carried out using a VAX780 computer in FORTRAN. It is anticipated that the final system will be implemented in FORTRAN or C to run on a MASCOMP computer. This computer is chosen because of the outstanding data acquisition capabilities which is a critical aspect of SEES. Equally important is the fact that a variety of languages and utilities are available commercially for this micro-supercomputer. A decision has to be made as to the language used to develop the embedded expert system (EES). One can choose the more traditional approach of using LISP or PROLOG (both of which are available to the MASCOMP, and both can interface with the rest of the system if it is written in C. We have decided

to use RULE MASTER by Radian Corporation. This is an expert system shell that would allow us to develop EES rapidly, and can be integrated to the rest of SEES easily.

PRELIMINARY RESULTS

The vibration time series is analyzed in a discrete data format. The data is first transformed into a power spectral density (PSD). Each discrete PSD is the power average over a small time interval at a frequency with a certain bandwidth. The power level of a frequency line is then temporalized. Figure 3 shows the amplitude time history of one important frequency band. The accompanying SSME power profile shows the shift from 100% to 104% power level. The amplitude of the time history shows a marked decrease at that time. Other bands show an increase at ramp up to 104%. The characterized signatures consist of a covariance matrix, C, which measures coupling between components of the sample vectors and the mean sample vector, **M**. A signature is a measure of the turbo-pump's performance profile at a given load condition. When a turbo-pump is operated at a load condition for an extended period, its performance may degrade from nominal to anomalous. This degradation is measured by the HMS and characterized into a class ensemble of signature, at a load condition. Two signatures characterized from the SSME test are presented in Figure 4. The spectral components of these signatures are very complicated; therefore, AI techniques must be used to classify data.

CONCLUSIONS

Preliminary analysis has shown that the SEES development engine successfully extracts signatures from SSME test data that can be formulated into rules for the SEES knowledge base.

REFERENCES

[1] Barkhoundarian, S. and A.T. Zachary, "Condition Monitoring For Space-Based Reusable Rocket Engines", ASME Winter Annual Meeting, New Orleans, LA, December 1984.

[2] Coffin, T. and J.Y. Jong, "Signal Detection Techniques for Diagnostic Monitoring Of Space Shuttle Main Engine Turbomachinery", NASA Technical Report 66938-01, February 1986.

[3] Evatt, J.C. and W.W. Palmquist, "Failure Control Techniques for the SSME", NAS8-36305, Phase I Final Report, Rockwell International.

[4] Hampson, M.E., J.J. Collins, M.R. Randell, and S. Barkhoundarian, "SSME Bearing Health Monitoring Using a Fiberoptic Deflectomer", NASA Conference on Advanced Earth-to-Orbit Propulsion Technology, Huntsville, Alabama, May 1986.

[5] Jong, J.Y. and Thomas Coffin, "Diagnostic Assessment of Turbomachinery by Hyper-Coherence Method", NASA Conference on Advanced Earth-to-Orbit Propulsion Technology, May 1986.

ORIGINAL PAGE IS
OF POOR QUALITY

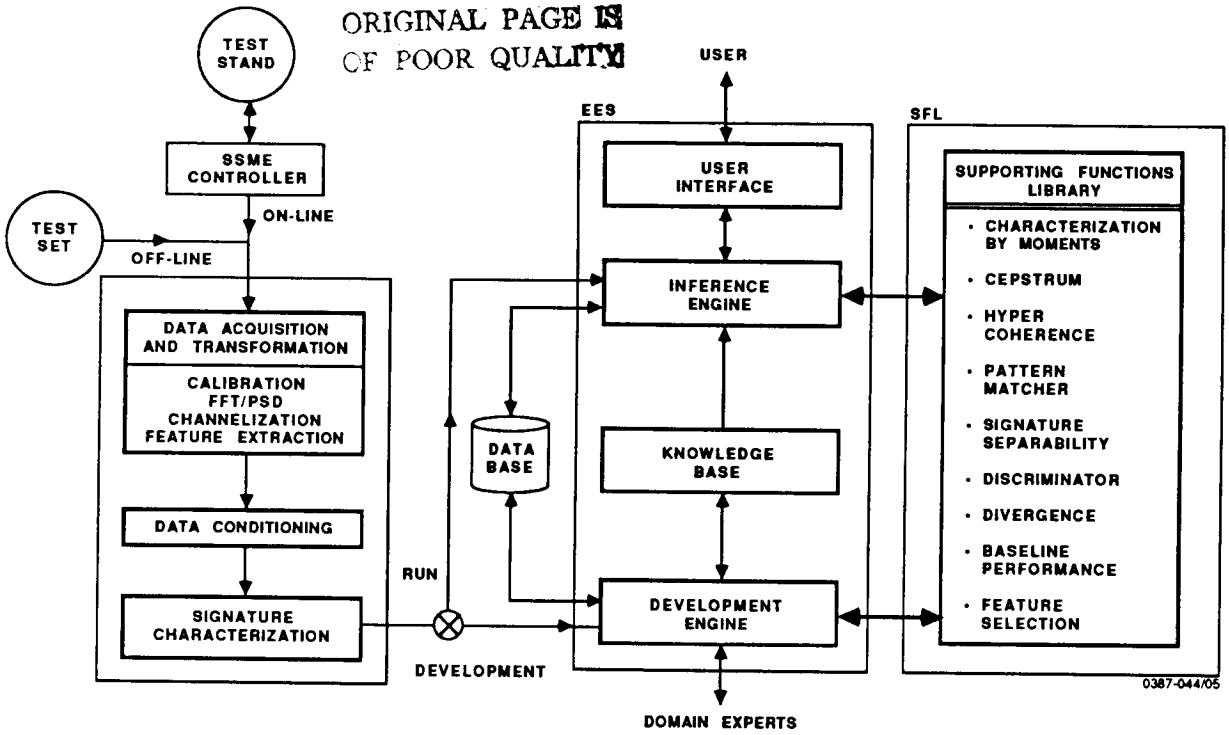


FIGURE 2. SEES - HMS ARCHITECTURE

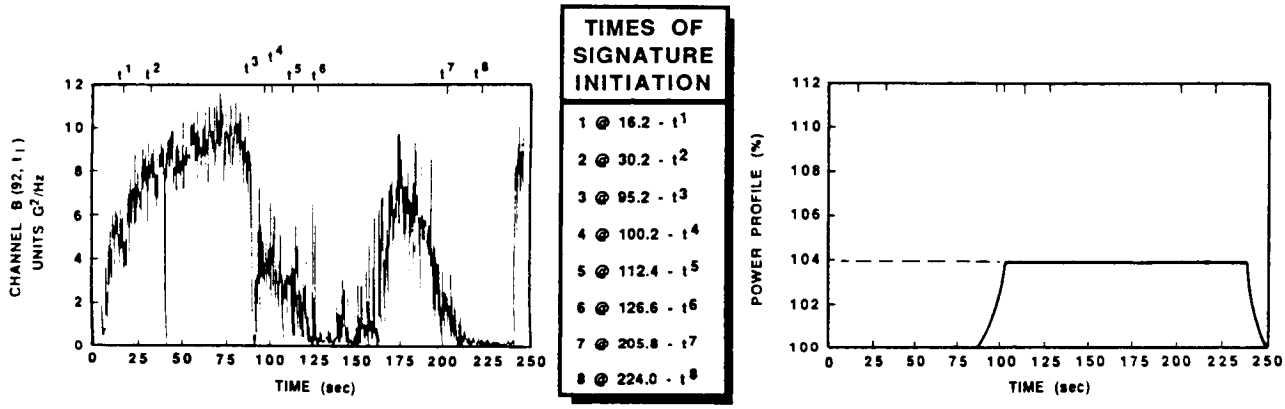


FIGURE 3. TEMPORALIZED DATA OF CHANNELS FROM SSME TEST #A2-356-5042

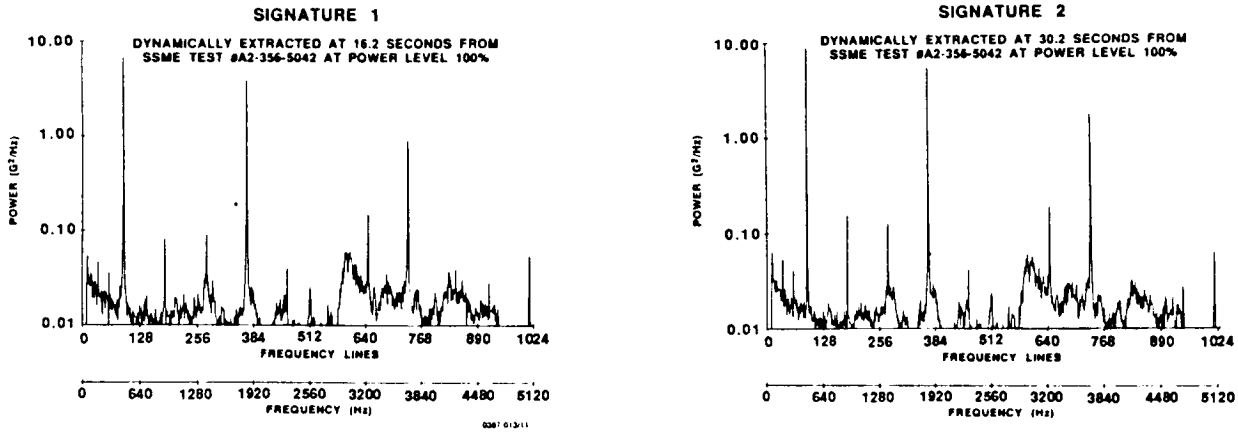


FIGURE 4. TYPICAL SSME SIGNATURES

0387-013/14

N88-16383

QUALITATIVE AND TEMPORAL REASONING IN ENGINE BEHAVIOR ANALYSIS

W.E. Dietz
M.E. Stamps
M. Ali
The University of Tennessee Space Institute
Tullahoma, TN 37388

ABSTRACT

One of the challenging difficulties in automatic fault diagnosis is generating a qualitative understanding of how a physical mechanism behaves in abnormal situations. Numerical data generated accross time by simulation models reveals changing internal states and mechanism behavior. However, understanding and interpreting such data by computers, an important part of monitoring, operating, analyzing, diagnosing, and debugging complex physical mechanisms, have been difficult tasks.

Diagnosis of physical mechanisms if often accomplished by analyzing various physical parameters qualitatively, i.e. by analyzing trends and relative magnitudes of parameters. Temporal information, such as the relative times at which various parameters exhibit changes in behavior, is also an important aspect of diagnosis.

An approach is being investigated at the University of Tennessee Space Institute to apply qualitative and temporal reasoning to analyze mechanism behavior and to diagnose faults in physical mechanisms. The present domain is jet engine diagnostics; however, the approach is applicable to other domains.

In the present study, numerical simulation models, engine experts, and experimental data are used to generate qualitative and temporal representations of abnormal engine behavior. Engine parameters monitored during engine operation are used to generate qualitative and temporal representations of actual engine behavior. Similarities between the representations of failure scenarios and the actual engine behavior are used to 1) diagnose fault conditions which have already occurred, or about to occur, 2) increase the surveillance by the monitoring system of relevant engine parameters, and 3) predict likely future engine behavior.

The stored failure representations include both sudden failures and slowly developing fault conditions. As a result, the system can detect developing faults which have not yet caused a parameter to exceed safety limits.

The stored representations of failure scenarios also allow

PRECEDING PAGE BLANK NOT FILMED

The stored representations of failure scenarios also allow predictions of engine behavior to be made. This information is used by the monitoring system to focus attention on the most relevant engine parameters. Predictive behavior is also used in advising the pilot on appropriate corrective action.

N88-16384

Exploring Hypotheses in Attitude Control Fault Diagnosis

Benjamin Bell
GE Astro-Space Division
U7049 P.O. Box 8555 Philadelphia PA 19101

Abstract

Recent activity in spacecraft design has been geared toward providing an assortment of new capabilities in space, in an attempt to satisfy the demanding mission requirements posed by such programs as the Strategic Defense Initiative (SDI) and Space Station. These requirements will include on-board fault detection and fault correction, and current work at GE Astro-Space is addressing this area through the use of knowledge-based systems. This paper describes a system which analyzes telemetry and evaluates hypotheses to explain any anomalies which are observed. Results achieved from a sample set of failure cases are presented, followed by a brief discussion of the benefits derived from this approach.

1. Introduction

The attitude control subsystem (ACS) orients and stabilizes the satellite after launch vehicle seperation, maintains pointing during on-orbit and payload operations, and controls the satellite attitude during orbit-adjust operations. Spacecraft attitude is of critical importance, since a slight error in vehicle alignment not only degrades mission performance, but also may cause changes in momentum rates that may propagate until the spacecraft is spinning out of control. Current architectures provide some level of autonomy in the ACS via closed-loop control, which can generally compensate for attitude errors attributable to normal vehicle dynamics. More serious errors are handled from the ground, during a process which includes a rapid effort to put the satellite into a safe state (thirty minutes to several hours), followed by an analysis phase which may take several weeks. During this time, mission performance may be interrupted, a situation which is not compatible with Government objectives.

To better maintain mission performance and to avoid propagation of attitude errors, anomalous conditions must be detected and isolated as rapidly as possible. This goal becomes harder to achieve as satellites become more complex; but if the satellite itself were capable of performing fault-isolation, then the risk of serious failure would be greatly reduced, along with the need for highly skilled ground personnel.

The **ACS Diagnostic System** demonstrates the potential of knowledge-based systems to offer this capability. As a preliminary step toward achieving satellite autonomy, this diagnostic program performs ground-based detection and isolation of faults within a satellite ACS. Detection of an anomaly triggers the generation of hypotheses. By extracting information from the telemetry useful to its diagnosis, this system independently pursues each possible explanation. The likelihood of each explanation depends on features identified in the telemetry; additionally, explanations are ruled out when contradictory evidence appears in the telemetry.

2. Current Diagnostic Procedures

Present techniques for diagnosing ACS faults rely on attitude error limits for detection. If the satellite attitude error limits are exceeded, the primary objective is to put the satellite into a safe state, which may involve unloading the momentum in the wheels, performing Auto Sun Acquisition, or disabling thrusters. Diagnosis proceeds only after the satellite is in a safe configuration. The principal technique employed in locating the source of the anomaly is to switch to a redundant component and then recheck the anomalous telemetry. In the case of a Reaction Wheel anomaly, the suspect wheel is turned off and the telemetry is monitored with the satellite operating on three wheels.

Effective application of these procedures requires some degree of expertise. For example, the choice of which component to switch with its redundant partner relies in part on an examination of certain informative telemetry behaviors. The diagnostic system, therefore, requires expertise, so that it may apply knowledge of the ACS and of telemetry analysis to explain a currently observed anomalous condition.

3. Expert Knowledge

To trouble-shoot anomalies, ACS analysts generally rely on their knowledge in the areas of telemetry interpretation, failure mode behavior, and diagnostic strategies. Telemetry interpretation knowledge is applied when the analyst extracts useful information from the large volume of telemetry data. Examples of such information are transients and trends.

Failure mode knowledge may be encoded by capturing the expert's mental representation of a failure. Analysts often characterize an ACS failure in terms of the symptoms (anomalies) which may appear during that failure, including qualitative measures of the support a particular symptom lends to a hypothesized failure. The expert might indicate, for example, that during a tachometer failure there is a high probability that the wheel speeds will oscillate.

Knowledge about diagnostic strategies defines the expert's own internal protocol for diagnosing faults. It was determined through interviews with ACS analysts that the expert initially reacts to an anomaly by considering all possible explanations, creating a mental model for each hypothesis. The expert then seeks evidence which can distinguish among the possible failures, primarily by comparing the actual telemetry to the predicted observations for that failure. Important also is the expert's ability to rule out a failure on the basis of evidence to the contrary. The expert may explain, for example, that the possibility of a drive failure may be ruled out if the motion in the opposite wheel indicates that the wheels are spinning normally.

4. Encoding Expertise

An appropriate representation must be selected for each of the three types of knowledge discussed above. Telemetry interpretation knowledge consists of categorical descriptions of anomalous behaviors, or *features*. This knowledge is encoded as rules, with each rule capable of identifying whether a specific feature is present in a telemetry point's recent

value history. The one-to-one correspondence between feature types and rules facilitates knowledge engineering and rapid prototyping.

Failure mode knowledge characterizes ACS failures as the collection of symptoms which may appear during that failure. A suitable representation for this type of expertise is a *schema*, an object which is composed of *slots* that specify the attributes of the schema. One schema completely defines a failure in terms of its symptoms, with each symptom described in its own slot. This slot description identifies the name of the symptom, the maximum time it would take for the symptom to appear during the failure, and the likelihood of the symptom appearing during the failure. Likelihoods are expressed qualitatively with the symbols 'H', 'M', and 'L' (High, Medium, and Low), 'A', and 'N'. An 'A' indicates that the symptom must **always** appear, and 'N' that the symptom will **never** appear. Figure 1 shows a sample failure schema. Because each symptom has attributes which

```
(defschema pss+.failure
  (symptom ((jump-to-zero pss+.current.use) A (00 00 35)))
  (symptom ((steady-decrease pss-.current.use) A (00 01 30)))
  (symptom ((steady-decrease yrss+.current) M (00 04 30)))
  (symptom ((steady-decrease yrss-.current) M (00 04 30)))
  (symptom ((gradual-decrease estimated.aom-y) M (00 06 00)))
  (symptom ((transient estimated.att-p) H (00 00 05)))
  (symptom ((transient estimated.att-r) H (00 00 30)))
  (symptom ((transient estimated.att-y) H (00 00 05)))
  (symptom ((unbalanced py+.wheel.speed py-.wheel.speed) M (00 02 30)))
  (symptom ((oscillation pr+.wheel.drive) L (00 00 00)))
  (symptom ((oscillation pr-.wheel.drive) L (00 00 00)))
```

Figure 1: A sample failure definition.

are independent of any one failure, schemas are used for defining symptoms as well. In this case the symptom is linked with failures by identifying each failure as a *possible cause* of the symptom, where appropriate. This schema organization allows rapid access to the predefined symptom and failure characteristics, and its modularity permits rapid prototyping as well.

5. Encoding Diagnostic Strategy

The third type of expertise, diagnostic strategy, differs in that it is comprised of *procedural* knowledge, whereas the two previous categories of expertise encompass *declarative* knowledge. This leads to a different implementation: rather than employing rules or schemas, this knowledge is represented as **meta-rules** which govern the way in which the declarative knowledge is applied to the problem-solving task.

The objective of this meta-structure is to implement a reasoning strategy derived from the techniques employed by human experts. The diagnostic knowledge discussed earlier is suitably represented using the Set Covering Model [2]. Applying this approach, generating hypotheses to explain a symptom is simply a matter of locating the symptom's schema

and reading the possible causes for that symptom. Each such possible cause then becomes a hypothesis. Hypothesis evaluation is accomplished by observing how well a hypothesis *covers* the symptom-set, i.e. how many of its symptoms have been detected. Symptoms vary in how strongly their presence supports a hypothesis, so this factor is incorporated into the evaluation procedure. In addition, some symptoms are required to support a hypothesis, and so that symptom's absence will allow the system to rule out the hypothesis. Conversely, some symptoms provide contradictory evidence, so that a hypothesis may be ruled out on the basis of that symptom's presence. This strategy is an appropriate representation for the diagnostic expertise discussed above, but requires a mechanism for handling the multiplicity of hypotheses. Fortunately, such a mechanism is available in this system.

Hypothetical Reasoning

The use of hypothetical reasoning is of prime importance to the diagnostic capability of this module. Using this approach, one hypothetical situation ('viewpoint') is created for each possible failure. The viewpoints are distinct from eachother, so each operates under its own set of assumptions (including of course the assumption about which failure occurred). When evidence rules out a hypothesis, the associated viewpoint is eliminated. Thus the use of hypothetical reasoning allows the system to pursue multiple hypotheses simultaneously. The reasoning within each viewpoint is geared towards supporting the hypothesis which generated that viewpoint, by setting goals to look for symptoms of the corresponding failure, and so is called 'goal-directed reasoning'.

Goal-Directed Reasoning

Hypotheses are generated in a forward-reasoning fashion. This means that the detection of a symptom triggers the hypothesizing mechanism, because the system has been 'told' that **if** a symptom occurs **then** it should hypothesize all failures which could cause that symptom. Once the hypotheses have been generated, however, the reasoning occurs in reverse. In backward reasoning, the system is told, for example, that **if** Symptom-A occurs **then** the likelihood of Failure-1 is increased. This generates a goal of detecting Symptom-A. Suppose the symptom knows that **if** the reaction wheels are unbalanced **then** conclude that Symptom-A has occurred. Then a *subgoal* is generated to detect an unbalanced wheel pair.

Within each viewpoint, then, unique subgoals are generated. This is a very important feature because of the computing expense involved in detecting symptoms. The savings is realized by only looking for symptoms which will help to support or rule out hypotheses. Savings also is achieved in the case when a hypothesis is ruled out, because all goals generated to support that hypothesis are eliminated.

6. Implementation

Combining this reasoning strategy, the failure and symptom schemas, and the feature-identification rules results in a system which emulates the diagnostic performance of a human expert. The process operates first in a fault detection mode, and then if triggerred, in a fault isolation mode.

Fault Detection

The current practice of fault detection by limit-checking has an inherent limitation, in that an anomaly may go undetected because no limits are exceeded. Moreover, even when a telemetry value exceeds its limits, it may not do so immediately upon failure, but instead may remain within limits for several minutes after the failure occurs. Further, simulations of ACS failures indicate that transients are reliable fault indicators. The fault detection mechanism used in this system, therefore, achieves the earliest possible fault detection, by using any ACS transient to trigger activation of the diagnostic procedures. This detection is the responsibility of the feature-identifying rule for transients, which monitors the histories of ACS telemetry points. The detection of a transient triggers the creation of **viewpoints**, each of which contains the assumption that one particular failure has occurred.

Fault Isolation

The meta-rules responsible for viewpoint creation not only hypothesize failures, but also scan each failure's list of symptoms, and set as *goals* the determination of these symptoms' presence or absence. These goals are satisfied by the feature-identifying rules, each of which is specific to a particular symptom type. A goal to find an oscillation in a wheel drive, for example, would activate a rule which 'knows' how to identify oscillations. In this way, only those symptoms relevant to the diagnosis are investigated.

The status of a symptom is initially `UNKNOWN`. Once it becomes a goal, it is assigned an expiration time, by adding that symptom's maximum appearance time to the current time. If it is observed prior to its expiration, it is assigned a status of `KNOWN-PRESENT`, otherwise its status is `KNOWN-ABSENT`. This information is processed by probability determination rules, which maintain a current probability for each hypothesized failure. The contribution a symptom's presence makes to a failure's probability depends on the likelihood, expressed qualitatively with the symbols 'H', 'M', and 'L', of that symptom occuring during the failure. These values are stored as part of the symptom slots in the failure schema, and are assigned numerical equivalents for computational purposes. If a symptom is known to be absent, it contributes a corresponding negative likelihood. Unknown symptoms are not included in the calculation. Probabilities are kept current by rules which react to a change in a symptom's status, by adjusting the probability of any failure related to that symptom (i.e. any failure identified as a **possible cause** in the symptom's schema).

The probability analysis helps the operator to distinguish among failures by providing a basis for comparing the alternative hypotheses. But more powerful than that is the system's capability to rule out failures on the basis of evidence in the telemetry. A failure may be ruled out by the program under two conditions (the operator may also rule out a failure). The first occurs when a symptom required to support the hypothesis is absent. This is detected when a symptom identified in a failure schema as an 'A' symptom has a status of `KNOWN-ABSENT`. The second condition occurs when a symptom which would not appear during this failure is observed. This is indicated when a symptom identified in a failure schema as an 'N' symptom has a status of `KNOWN-PRESENT`. When a failure is ruled

out, all goals set to support that hypothesis are removed, and the reason for ruling out the failure is recorded in its schema.

7. Test Environment

The diagnostic capability of this system was examined within a test environment which provided simulated spacecraft telemetry and an interactive operator station. The telemetry was derived from actual satellite telemetry (for 'normal' data) and from an ACS simulator (for failure data). Telemetry processing software generates telemetry frames and sends one frame to the diagnostic system every two seconds, to create a real-time environment. The operator station keeps the operator fully informed about detected anomalies and about the status of each failure being investigated. Three displays are available: a table of all detected anomalies, a symptom display detailing a selected symptom, and a failure display providing data about a selected failure.

8. Example

When the ACS Diagnostic System first detects a transient in ACS telemetry, the hypothetical reasoning mechanism creates four possible explanations: a tachometer failure, a sun sensor failure, a wheel drive failure, and an Attitude Control Electronics (ACE) failure. The goal-directed reasoning then sets as goals the detection of symptoms appearing in each of these failures. As a result, the feature-identifiers are activated and telemetry analysis begins. Figure 2 illustrates the result of the operator selecting a ruled-out failure for display. A few moments later, the operator selects an observed anomaly, which brings up the symptom display shown in Figure 3. After about forty seconds, a single failure remains, but the system continues its analysis until the operator decides to accept the failure. Figure 4 shows the main display after two and a half minutes, identifying the relevant features found in the telemetry since the failure was first detected.

9. Conclusion

This prototype demonstrates the promise of the approach used because effective reasoning was achieved using a straightforward representation and relatively simple heuristics. Further, because of the way the expertise is segmented, the system can not only be made to perform better in this domain, but can also be applied to different problem areas. Installing operational versions of an expert system such as this one thus becomes more cost-effective, as new diagnostic systems can be generated from existing ones by replacing a specific knowledge base and leaving the reasoning strategies intact.

Certainly, then, AI-based diagnosis will help to increase mission reliability and reduce the need for highly-skilled personnel. As this technology evolves, advances in the intelligence of these systems will provide even greater benefit. One way to improve knowledge-based systems is through the use of learning. Expert systems can learn by modifying their strategies when such strategies fail to provide acceptable solutions. This approach has been tested in a satellite diagnosis system [1], and future work will provide large reductions in the time and cost involved in knowledge engineering.

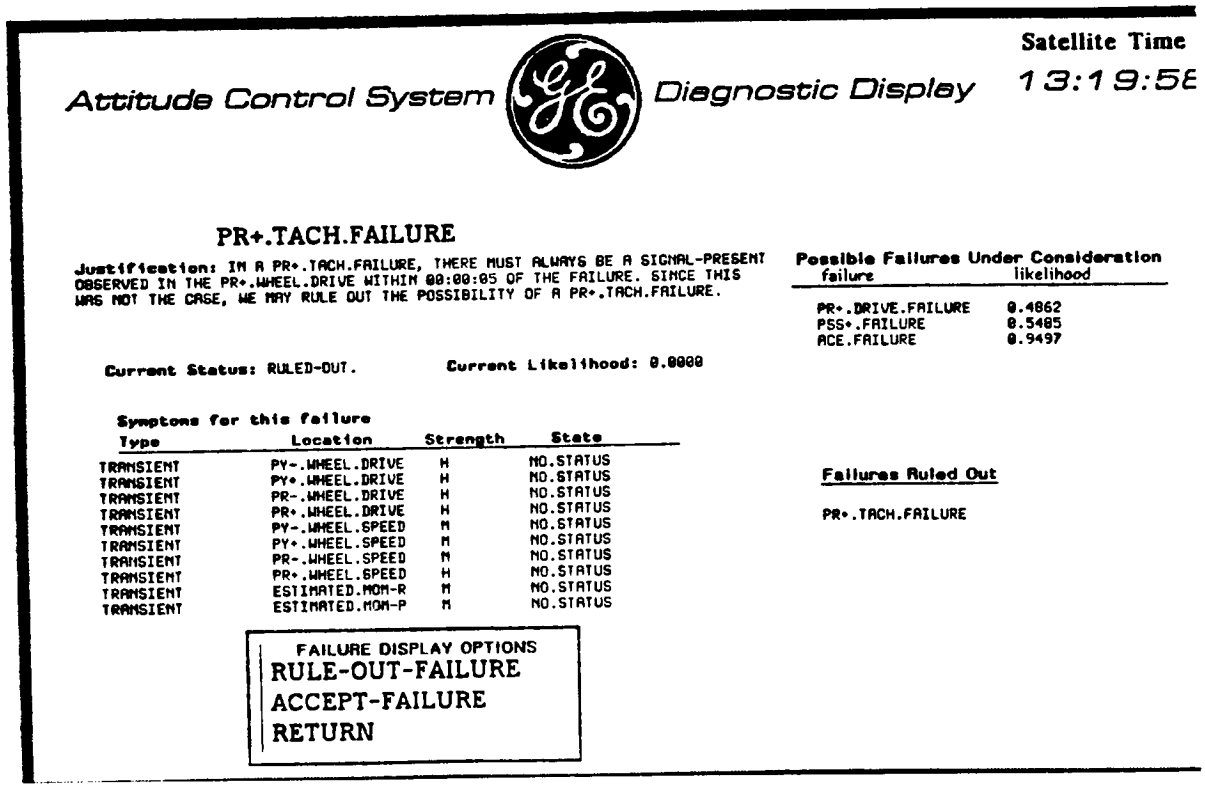


Figure 2: Failure Display Screen example.

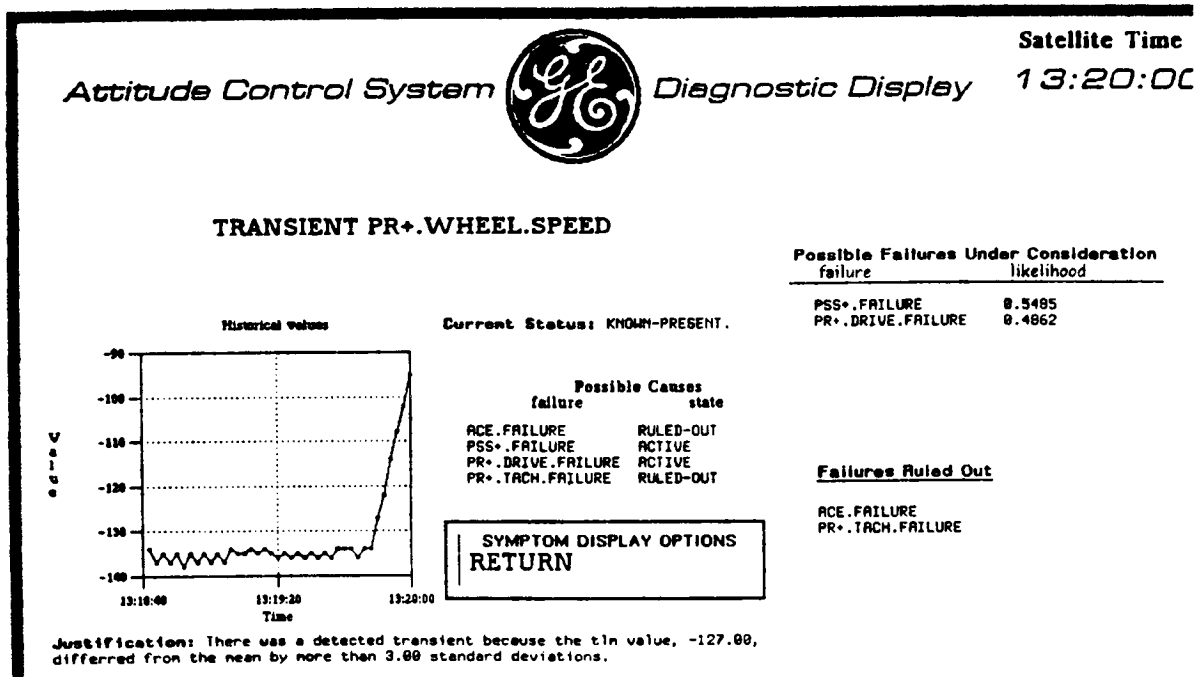


Figure 3: Symptom Display Screen example.

ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

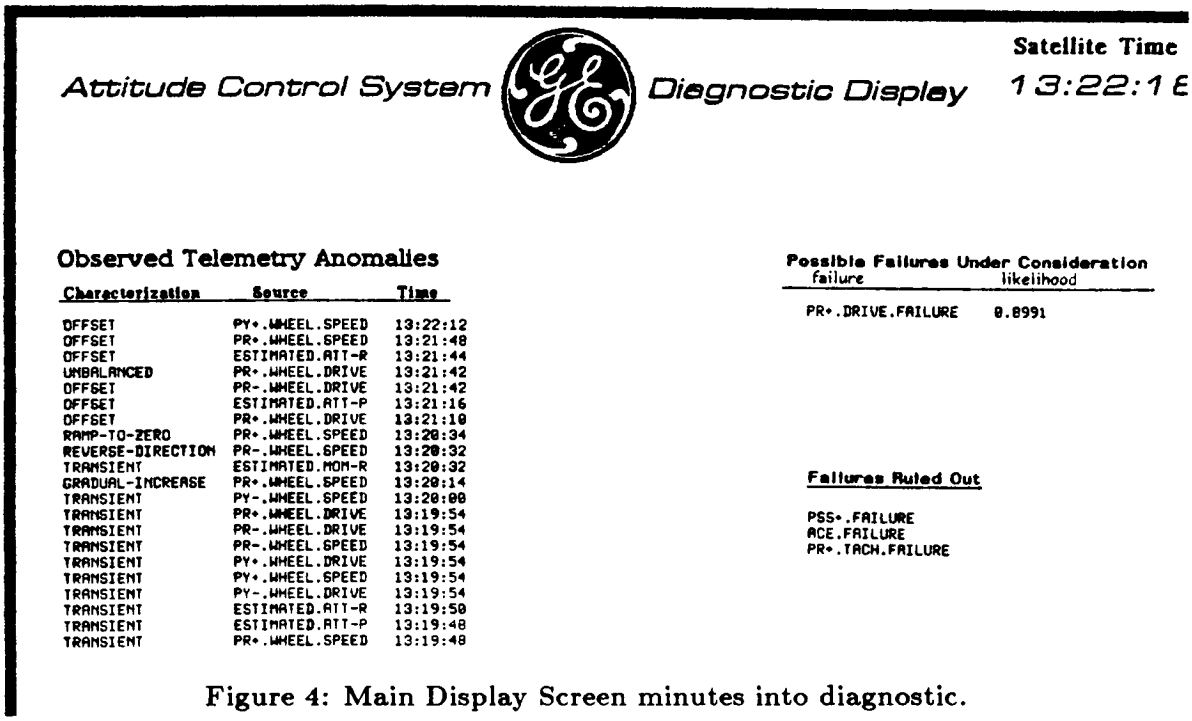


Figure 4: Main Display Screen minutes into diagnostic.

Another area under investigation is the use of model-based reasoning. An expert system could predict behavior using models of the satellite and its subsystems, and so could identify faults by differences observed between actual behavior and behavior predicted by the model. Isolation of faults is also facilitated by the causal links implicit in these models. Applying models to diagnostic expert systems is currently under study at GE Astro-Space. Results from this and other research promise to put more intelligence into AI, so that the increasing complexity of space systems is paralleled by our improved ability to control and maintain them, and eventually, by their ability to maintain themselves.

References

Pazzani, Michael J. (1986). Refining the Knowledge Base of a Diagnostic Expert System: An Application of Failure-Driven Learning. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1029–1035, AAAI, Philadelphia, PA.

Reggia, James A., D. S. Nau, and P. Y. Wang (1984). Diagnostic Expert Systems Based on a Set Covering Model. In M. J. Coombs (Ed.), *Developments in Expert Systems*, 35–58. London: Academic Press.

N88-16385

EUREX D

An Expert System for Failure Diagnosis and Recovery in the TCS of the European Retrievable Carrier EURECA

A. Kellner, W. Belau, N. Schielow
Space Systems Group
MBB/ERNO
Bremen, West Germany

ABSTRACT

In this paper an expert system for diagnosis and recovery of failures in the freon cooling loop of the European retrievable experiment carrier EURECA is described.
This system demonstrates the feasibility of a functional scope of expert diagnostic systems which appears to be essential for practical applications of such systems in space technology. This scope comprises : early warning and treatment of incomplete information, fault tolerance, identification of failure superpositions (particularly involving failed sensors), intelligent reaction to unforeseen events and detailed status display for optimal recovery action.

1. INTRODUCTION

Particularly in view of the implementation of expert systems for failure diagnosis and recovery on autonomous spacecraft, but also with respect to their application as ground-based consultant systems, a certain enhanced functionality of such systems appears to be essential, covering in particular early warning and treatment of incomplete information, failure superposition, intelligent reaction to unforeseen events, tolerance to isolated faults in the knowledge base and detailed status display for optimal recovery action.

In this paper an expert system is described which, in its first development phase, has been used to implement and assess the technology required for the realization of these requirements for the diagnostic process. As such it could initially be used as a ground-based operator's consulting system, serving at the same time as test-bed for a further refinement of this technology for subsequent on-board applications.

2. THE COOLING LOOP OF EURECA

As system to be monitored the cooling loop of EURECA was chosen with the aim of a later expansion of the knowledge base to also include the thermal control unit, thus providing the complete TCS as application domain in the final stage.

The cooling loop of EURECA is depicted in Figure 2 showing the pump package (FPP) consisting of two redundant pumps of which one drives the cooling medium, freon, through the experiment line (upper branch) where the freon cools the experiments, then through the two radiators (Rad +x and Rad −x) where the heat taken up by the freon is dissipated and finally through the equipment line (lower branch) where equipment such as batteries and power distribution units are kept approximately at room temperature.

The fine tuning of the freon temperature is achieved by small adaptive heaters positioned along the experiment and equipment line as well as on the radiators.

The sensors are shown as icons in Figure 2 and measure pump inlet pressure (P_{in}), pump outlet pressure (P_{out}), pump inlet temperature (T_{in}), radiator inlet temperature (T_{s_3}), radiator outlet temperature ($T_{s_{1,2}}$), freon quantity in the accumulator (Acc. Q.) and the electrical pump current (I_p). Moreover, a delta pressure switch (dP) gives a signal if the pump pressure head has

broken down. (It should be mentioned that the question whether all these sensors will be available for EURECA is still under discussion. However, as the initial development stage of the expert system only aims at a technology demonstration, the exact number and type of sensors used is not crucial).

In addition to these direct measurements, the total power consumed by the adaptive heaters for the experiment line (Pwr Ex), the radiators (Pwr +x and −x) and the equipment (Pwr Eq) is also used, since changes in this power can serve as indirect indications of changes in flow, thus substituting to a certain extent the fact that a direct flow measurement will not be available for the EURECA cooling loop.

3. KNOWLEDGE REPRESENTATION

EUREX D has a rule-based knowledge representation which is characterized by :

- **Global monitoring :**
  Diagnosis is not only based on single sensor monitoring but on a global assessment of the concurrent readings of all sensors, identifying characteristic data patterns and relations (such as temperature gradients along the different sections of the cooling loop) thus providing a broad basis for the diagnostic process.

- **Indirect monitoring :**
  Apart from a direct interpretation of sensor signals such as interpreting the activation of the delta pressure switch as pump performance degradation, extensive use is made of indirect monitoring such as taking a flow reduction as additional evidence for the pump performance degradation, or using temperature readings for a measurement of flow as indicated above, etc.

- **Redundant monitoring :**
  In the reasoning process ALL known evidence for a given anomaly is taken into account thus allowing for incomplete or even partially erroneous information or knowledge to be processed without grave consequences since the system's dependency on individual bits of information or knowledge is greatly reduced.

- **Multi-valued logic :**
  Taking all known evidence into account implies the use of non-conclusive evidence :
  For instance, the flow reduction mentioned above as being indicative of a pump performance degradation could also indicate a flow blockage or even a superposition of both anomolies.
  Therefore rules generally hold only partially, which is represented by implication strengths σ taking values between 0 and 1, assigned to the rules.

- **Causal connectivity :**
  Although EUREX D does not reason "from first principles" but relies on knowlege based on engineering experience and heuristics, it is able to identify causal connections between states (e.g. a flow blockage being the cause of a flow reduction) for greater transparency and completeness in the description of the system's state. The knowledge necessary to identify these causal connections is provided by pointers assigned to the states.

4. INFERENCE MECHANISM

The various steps in the knowledge-based data processing, which are also reflected in the system's architecture shown in Figure 1, are given by :

- **Data processing :**
  i.e. the computation of temperature gradients along the various sections of the cooling loop, pressure differences etc.

- **Data classification :**
  i.e. local classification of the data in relation to the nominal interval generating "observations" such as "temperature high", "pressure normal" etc.

- **Sensor state assessment :**
  Dedicated rules check the plausibility of sensor readings on the basis of these observations and assign belief values to the sensors : 1 for normal operation, 0 for degraded sensors.

- **System state assessment :**
  For each system state, all rules pointing to this state are "tickled" and the implication strengths of fired rules are collected by an accumulation function leading to an integrated certainty factor. In the case of several inference steps (where system states serve as evidence for other system states) composite implication strengths are computed by propagation functions. In particular sensor belief values of 0 simply cause any inference based on this sensor to drop out of the reasoning process.

- **State evaluation :**
  States with certainty factor 1 are displayed as diagnoses. In case such cannot be found, several states with the highest certainty factors are presented as possible but not conclusive diagnosis. Diagnosed states are displayed in columns, the states given in a certain column always being the cause of the states in the adjacent columns to the left, thus generating a detailed status display which is not just based on the primary cause of an anomaly.

- **Recovery actions :**
  Depending on the diagnosis, appropriate recovery actions are selected. When the diagnosis is not conclusive, i.e. when it consists of several states with certainty factors less than one, the suggestions for recovery actions obviously also cannot be conclusive but are qualified by priority factors which are functions of the certainty factors and possible additional information (such as the rule to always react to the most hazardous situation first, even if it has a comparatively low certainty factor). However, this identification of recovery actions has not yet been implemented in EUREX D.

5. IMPLEMENTATION

EUREX D is being developed on the LISP-based development environment KEE and runs on SYMBOLICS machines.

Sensor-dedicated demons are responsible for the data classification described above. States are objects (units in KEE terminology) automatically collecting the evidence pointing to the states and computing the integrated certainty factors.
Rules are grouped into classes corresponding to their firing priority in the inference process. Based on KEE's facilities the man-machine interface is strongly supported graphically, facilitating knowledge acquisition and explanation of the reasoning processes.
In particular, the sensor readings are shown as bar graphs and their classification as shaded/non-shaded areas, responding actively to the input data. Conversely, the bar graphs can be mouse-manipulated to preset the sensor readings for an initial nominal and a final anomal state for an in-built test simulator. This simulator generates, at fixed time intervals Δt, the sensor readings of the intermediate states which develop as the anomaly evolves from the initial nominal state to the preset end-state.

At each Δt EUREX D then performs its diagnosis on the sensor readings of these intermediate states.

6. FUNCTIONAL SCOPE

EUREX D displays the following functional scope corresponding to the requirements listed in the introduction :

- **Early warning and treatment of incomplete information :**
  Due to this fact that the reasoning mechanism is based on multi-valued logic and can process non-conclusive evidence, the system does not depend on the sensor data patterns characteristic of fully evolved anomalies for its diagnosis, but is able to process first symptoms of developing failures (i.e. incomplete information), presenting assumptions of several possible failures (weighted with certainty factors less than 1) for early warning and preventive action. An example of an assessment of the evolving symptoms of a flow blockage is given in Figures 2–3. A similar treatment applies when the incompleteness of information is due to other reasons, such as reduced sensor availability etc.

- **Failure superposition :**
  Obviously a premeditation of all failure superpositons for a diagnostic system is impossible, and, like most other diagnostic systems, EUREX D is designed for the diagnosis of single failures only. However, it does display the ability to treat failure superpositions to a fair extent :
  Degraded sensors are detected by dedicated sensor rules and taken out of the diagnostic process as described in Chapter 4. The diagnosis of simultaneously occuring system anomalies then proceeds on the basis of incomplete information as shown above. Thus concurrent sensor failures and system anomalies can be discerned.
  Concurrent system anomalies can obviously be detected if they do not have opposing effects on the same sensors. Otherwise the system will again offer several assumptions with certainty factors less than 1. For example the superposition of a flow blockage and a leak leads to the assumption of pump performance degradation (c.f. = 0.5), leakage (c.f. = 0.7) and flow blockage (c.f. = 0.5).

- **Treatment of unforeseen events :**
  Processing of non-conclusive evidence in EUREX D also facilitates intelligent reaction to non-premeditated events. On the basis of the subset of recognized features, known situations are enumerated which have the greatest similarity to the unforeseen event.
  At the same time, the fact that these are just assumptions is signalled by certainty factors less than 1.

- **Tolerance to isolated faults in the knowledge-base :**
  Regardless of whether such faults are due to erroneous coding or some irradiation of computer memory in case of on-board expert systems, it is imperative that an expert system does not react catastrophically in case of such error.
  Due to the excessive knowledge redundancy (see chapter 3) this tolerance is indeed given to a great extent in EUREX D, where most isolated faults are "drowned" in the "majority vote" of the remaining evidence.

- **Detailed status display :**
  The inclusion of states causally connected to primary failure states for greater detail of status display has already been described in Chapters 3 and 4.

7. CONCLUSIONS

An expert system for the failure diagnosis of the cooling loop of EURECA has been described which displays a couple of features which appear to be essential for practical applications of such expert systems in space technology, the main aspects of the underlying methodology being given by knowledge redundancy and multi-valued logic.
It should be noted, however, that the management of uncertainty involved still poses some problems in the case of very large knowlege-bases and future work will have to concentrate on this aspect.

ORIGINAL PAGE IS
OF POOR QUALITY

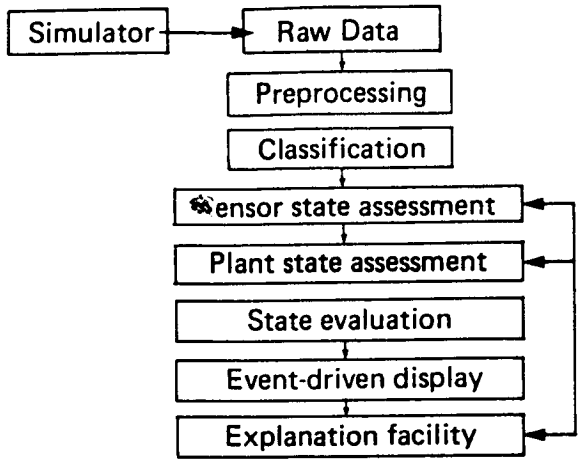


Figure 1 : Components of EUREX D

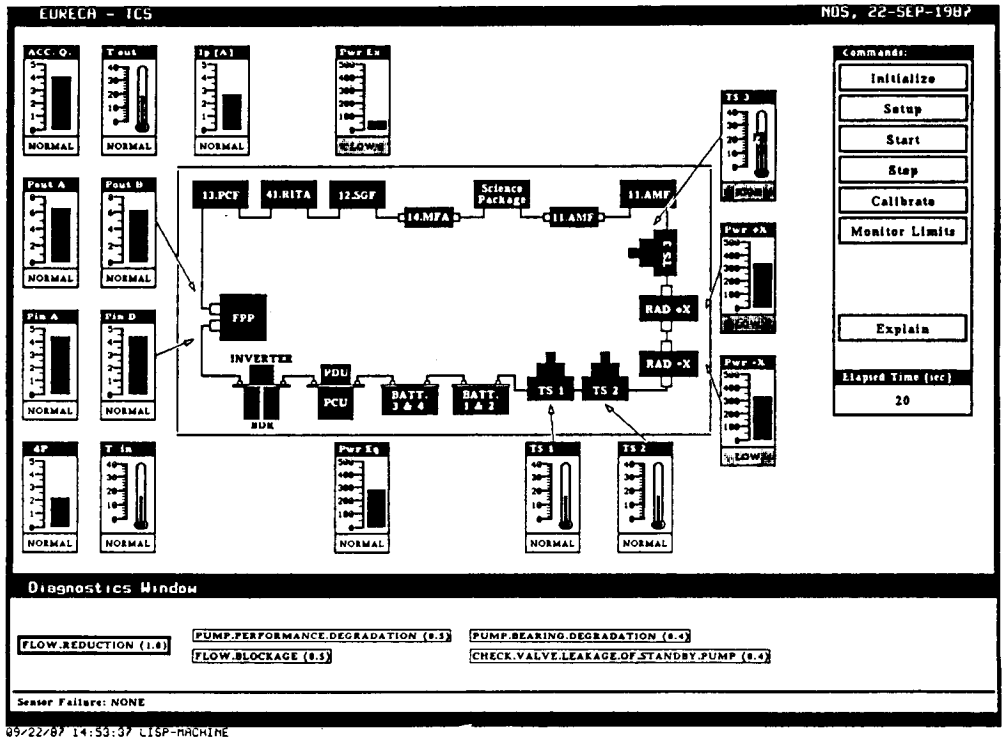


Figure 2 : Display of cooling loop and diagnostic window showing an assessment of first symptoms of a flow blockage

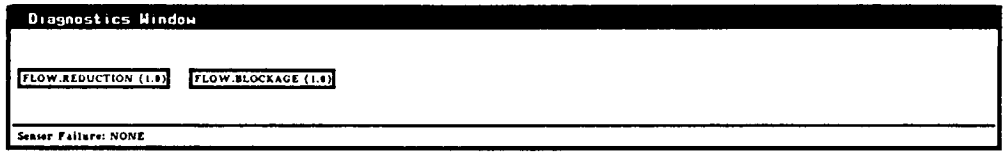


Figure 3 : Diagnosis of fully evolved flow blockage

N88-16386

Task Path Planning, Scheduling, and Learning for Free-Ranging Robot Systems

G. Steve Wakefield
Research Assistant, Johnson Research Center
Research Institute, Room A-4A
University of Alabama in Huntsville
Huntsville, AL 35899
(205) 895-6217

ABSTRACT

The development of robotics applications for space operations is often restricted by the limited movement available to physically guided robots. Free-ranging robots can offer greater flexibility than physically guided robots in these applications. This paper presents an object oriented approach to path planning and task scheduling for free-ranging robots which allows the dynamic determination of paths based on the current environment. The system also provides task "learning" for repetitive jobs. This approach provides a basis for the design of free-ranging robot systems which are adaptable to various environments and tasks.

INTRODUCTION

The development of robotics applications for space operations is often restricted by the limited movement available to physically guided robots. Free-ranging robots can offer greater flexibility than physically guided robots in these applications. However, the use of free-ranging robots introduces two major complexities: path planning and task scheduling [5]. This paper presents an approach to path planning and task scheduling for free-ranging robots which allows the dynamic determination of paths based on the current environment, and provides task "learning" for repetitive jobs. This approach provides a basis for the design of free-ranging robot systems which are adaptable to various environments and tasks.

The system described is being developed for applications which are best accomplished through the use of free ranging robots. Some examples of such applications are operations in the various modules of the space station, work in hazardous environments, and industrial warehouse operations. Several goals are established for this research. The system should provide for:

PRECEDING PAGE BLANK NOT FILMED

- the ability for the robot to operate in different environments, dependent only upon the existing knowledge of each environment;

- the ability to execute requested tasks based on the knowledge of spare parts', tools', and work stations' locations;

- the ability to operate in a multi-robot environment;

- the requirement of a task definition only once; and,

- the use of previously defined paths whenever possible.

The system depends upon the use of a central control computer which contains knowledge of each operating environment, and controls all modifications to that environment.

CONTROLLED OPERATING ENVIRONMENT

The system is based on the concept that all dynamics in an environment are controlled by a central computer which uses a combination of artificial intelligence techniques and classical algorithms. In the current project, the environment is defined initially by the user; subsequent changes to the environment are performed by, and "remembered" by, the centrally controlled robots within the environment. All tasks and operations (including inputs to and outputs from the environment) are scheduled and planned before execution. The central computer defines the shortest path between the starting point and ending point of a task, monitors the location and use of all items required to complete a task, and coordinates the task with other robots within the environment.

The controlled environment also contains storage sites (for inventory, tools, spare parts, etc.), work stations (where the task is actually performed), and home bases (where the robot are be recharged when as necessary). The central computer maintains all necessary information and knowledge on storage sites, work stations and home bases. Such information includes location parameters, content status, idle/busy status, etc., and appropriate updates are made based on the dynamics of the environment.

The control system described in the paper has been developed on a Symbolics 3620 AI work station. The programs are written in Common Lisp but also utilize the Windows and Flavors packages available on the Symbolics, which provide an object oriented programming base. Each work station, storage site, robot, path, and task is represented as an object, with particular characteristics associated with each. In order to monitor robot

operations within the environment, the current program graphically simulates the results of the path planning and scheduling routines performed when a task request is processed.

PATH PLANNING AND LEARNING

Several researchers have investigated path planning in various environments. Trivedi, Gonzalez, and Abidi worked on robotics control for hazardous environments [6]. Path planning for a manipulated arm robot has been researched by Jentsch [4]. Taghaboni and Tanchoco have investigated path planning of free ranging robots [5]. Their work is based on selecting the best path from known paths depending on the path providing the shortest time to completion [5]. The system developed by Kaiser and Hawkins deals with the control of free-flying robots, and selects a path based on the cost function [3]. Weisbin has described work in path planning, as well as learning concepts, for autonomous robots [7]. The system described here draws from the above mentioned research, but takes a somewhat different approach. Not only is collision avoidance used in path planning, but the knowledge of item locations allows the robot to choose where it needs to go to obtain tools and spare parts, and where the work is to be performed. Knowing this, the most efficient path for completing an entire task can be determined.

Tasks are given to the AI control center by the user. The user specified task information is stored in a relational data/knowledge-base. Tasks, therefore, need to be defined only once. Thereafter, user specification of the task name will allow the control system to access information and knowledge required for task completion. As more and more tasks are defined, the power of the system increases and the amount of task information required from the user decreases.

Once a task is defined, the central computer searches the data/knowledge bases for the required item information. The movements of the robot to acquire the items and perform the desired operations are then planned, including collision avoidance of static objects and other moving robots. The robot movement is scheduled via a recursive routine which locates any obstructions in the direct path between two points and determines the shortest path around those obstructions. It then checks for obstructions in that path, until a clear path between the two locations is found. Once the task has been planned and scheduled, the control sequence is sent to the robot for task performance. The control computer can also be used to define actions to be taken by the robot upon task completion.

When a path is determined, any other tasks requiring movement between the same points can use the previously determined path, without requiring recalculation by the recursive functions. However, if the environment has changed (objects

added or moved) the recursive path finding routine is called and the new path subsequently stored for future use. Even when path modifications are made for a given task, the items and operations required to complete that task are "remembered" by the control computer from the initial definition of the task.

The pre-performance planning routine also allows for collision avoidance between robots in a multiple robot environment. Since all tasks and paths are determined prior to execution, the central computer knows where each robot is going to be at a given time and can use this information to determine the path for a new task. The path may include avoiding other robots in the environment by altering the path or simply waiting at a given location for another robot to pass.

LIMITATIONS AND ADVANTAGES

Among the advantages of this approach over other approaches being investigated are the increased flexibility of the system and the reduced overall CPU time. The increased flexibility is provided by: (1) shortest path determinations based on the current environment, (2) multi-robot operations within the environment, and (3) path modifications when required by the dynamics within the environment. The reduced CPU time results from not only the removal of continuous collision avoidance requirements but also the task and path "learning" capabilities of the control computer.

The major future enhancements to this approach should include: (1) developing the capability for a robot to assist in defining the initial environment via sensory enhancements; (2) providing the robot with the ability to search for a clear path through or around closely grouped objects; and (3) integrating classical algorithms for the performance of routine tasks once a robot has been placed at a work station. Each of these areas are being examined by current research and development efforts [1][2][6].

CONCLUSIONS

This paper presents an alternative solution to the problems of path planning and task scheduling for free-ranging robots. The approach is not limited to pre-defined paths and intersection nodes, but rather defines each individual path based on the current environment. The use of artificial intelligence techniques allows the control computer to monitor and modify the dynamic environment and to plan and schedule tasks accordingly. Complete development of this approach will allow the continued advancement of applications for free-ranging robots.

REFERENCES

1. Hopkins, S.H., and P.J. Drazan, "Semiautonomous Systems in Automatic Assembly," IEEE Proceedings, vol. 132, Part D, No. 4, July 1985, pp. 174-177

2. Huang, C.L., and J.T. Tou, "Automatic Generation of 3-D Pictorial Drawing from Intensity Image," Proceedings of SPIE Conference: Applications of Artificial Intelligence V, May 1987, p. 502

3. Kaiser, Donald Leo, and P.J. Hawkins, "Motion Planning for a Free-Flying Robot," Proceedings of NASA Conference on Artificial Intelligence for Space Applications, November 1986, pp. 247-255

4. Jentsch, Winfried, "Off-line Planning of Collision-free Trajectories and Object Grasping for Manipulating Robots," Proceedings of the Third Annual Conference on Artificial Intelligence and Information-Control Systems of Robots, June 1874, pp. 193-196

5. Taghaboni, F. and J.M.A. Tanchoco, "Dynamic Scheduling and Control of Free-Ranging Automated Guided Vehicle Systems," Proceedings of the 1986 Fall Industrial Engineering Conference, December 1986, pp. 127-129

6. Trivedi, M.M., R.C. Gonzalez, and M.A. Abidi, "Developing Sensor-driven Robots for Hazardous Environments," Proceedings of SPIE Conference: Applications of Artificial Intelligence V, Vol. 786 May 1987, pp. 185-188

7. Weisbin, Charles R., "Intelligent-Machine Research at CESAR," AI Magazine, Vol. 8, No. 1, Spring 1987, pp. 62-74

N88-16387

DEVELOPMENT OF A TASK-LEVEL ROBOT PROGRAMMING AND SIMULATION SYSTEM*

H. Liu, K. Kawamura, S. Narayanan**, G. Zhang, H. Franke & M. Ozkan
Center for Intelligent Systems, Vanderbilt University
Nashville, Tennessee 37235, USA

H. Arima
Process Control Equipments Design Dept.
Tokico, Ltd., Kawasaki, JAPAN

ABSTRACT

This paper presents an ongoing project in developing a Task-Level Robot Programming and Simulation System (TARPS). The objective of this research is to design a generic TARPS that can be used for a variety of applications. Many robotic applications require off-line programming, and a TARPS is very useful in such cases. Task-level programming is object-centered in which user specifies tasks to be performed instead of robot paths; graphics simulation provides greater flexibility and also avoids costly machine setup and possible damage. A TARPS has three major modules: world model, task planner and task simulator. The system architecture, design issues and some preliminary results are given in this paper.

I. INTRODUCTION

Robot programming systems can be divided into three broad categories: guiding systems in which the user leads a robot through the motion to be performed, robot-level programming systems in which the user writes a computer program specifying motion and sensing, and task-level programming systems in which the user specifies operations by their desired effects on objects [8]. Guiding systems are primitive, e.g., there are no loops, conditionals, or sensors, but are easy to use and can be implemented without a general-purpose computer. Robot-level programming systems have the capabilities lacked by guiding systems; however, the user must be familiar with both computer programming and robot manipulation. Task-level programming is an attempt to shift the burden of detailed robot programming from the user to the computer where only goals or tasks need to be specified by the user. Recently, more and more robotic applications require robot programming to be done off-line. This is often complicated by frequent task change and critical timing requirement. A Task-Level Robot Programming and Simulation System (TARPS) can be very useful in such cases since task-level progrmming is much more efficient than robot-level programming and guiding, and through computer simulation extensive experiments and analyses can be performed without the high cost of machine setup and risk of damage.

The system architecture of the TARPS being developed consists of three major modules: world model, task planner and task simulator as shown in Fig. 1. Based on the world model, the task planner translates object-centered task specifications to appropriate robot motion sequences. The robot motion se-

* This work was supported in part by Tokico, Ltd.

** Mr. Narayanan is now with Dept. of Elec. & Comp. Engr., Univ. of Calif., Davis, CA 95616.

PRECEDING PAGE BLANK NOT FILMED

PRECEDING PAGE BLANK NOT FILMED

quences and the world model can be simulated on the graphics terminal. The high-level task planning part is implemented in LISP, object model in FLAVORS, and robot motion synthesis in FORTRAN. Certain important issues such as object representation, collision avoidance and trajectory planning are also addressed.

II. WORLD MODEL

Task-level programming and simulation cannot be achieved without a world model. This world model should include a robot model, physical object model and environment model. The robot is modeled as a mechanical linkage system with various joint parameters and constraints. These parameters and spatial geometry of the manipulator are needed to compute and simulate robot motion. 3-D object representation has been a major research topic in computer vision, CAD/CAM and computer graphics. A 3-D object can be represented by one of the three general classes: (1) surface or boundary, (2) sweep and (3) volume. We adopted a surface-based representation scheme based on the evaluation in [4] and the following reasons:

1) We plan to derive the object model from the CAD model, and the sweep representation has been used only to a very limited extent in CAD.

2) Surfaces can be recognized by vision or range sensors and so any representation scheme utilizing surface descriptions can be easily integrated into a sensor-based robot planning system.

Any 3-D planar object can be represented by a graph where every vertex, edge and surface of the object corresponds to a node in the graph, and the arcs of the graph are the connected_to relations. Each node can be implemented as a computational object with certain slots. For example, a vertex node may have slots for vertex_id, x, y, and z coordinates, and related_edges. Data from other nodes can be obtained by message sending. At present, the object model is entered through a menu-driven interface; simple object models such as rectangular blocks, cylinder, cone and sphere can be entered through the menu. Once simple object models are defined, composite objects can be constructed in a way similar to that of Constructive Solid Geometry (CSG) representation. Algorithms for constructing object model from CAD data and B-splines representation are under development. The environment model has a world frame and coordinate frames for objects and robots. Semantic network and homogeneous transform are used to express their positions with respect to one another.

A. Surface-Based Object Representation

The surface representation of objects makes use of "faces." Any "face" of the object can be considered as a subset of the enclosing surface or boundary of the object. Conversely, the union of all possible "faces" of an object constitutes the boundary of that object [11]. For 3-D planar object the representation primitives are boundary, faces, edges and vertices. The primitive of a "physical object" is represented as a "computational object" [12]. A computational object is typically characterized by a set of "instance-variables" whose values are used to determine the current state of the object and its relation with other objects. Furthermore, there are procedures which can be used to determine the attributes of other objects and to make decisions to schedule operations concerning that object. Listed below are some of the

information required to characterize the face. Each face of the object is represented using the same scheme as is used for other primitives.

face_id: Used for the purposes of identification of a face.

adjacent_to: Has a list of face_id's as its value.

normal_vector: Equation of the vector normal to the face. This value can be computed from the face vertex positions of a planar surface. For a quadric surface, however, we require information about the surface shape.

related_edges: A list of edge_id's that belong to the face.

B. Conversion from CAD Data

The modelling scheme described earlier requires a lot of information to represent the object. There is an increasing need to obtain such information from a CAD data base of the task environment [4]. For example, the dimensions of a rectangular block can be obtained from a CAD model of the block. From these dimensions, it is possible to select a reference frame for the block and then compute the relevant information such as: 1) Location of vertices with respect to the assigned object reference frame; 2) labelling of edges by applying a rule that any two vertices constitute an edge; 3) labelling of faces by computing those sequences of edges which form loops, i.e., staring from any vertex, find those edges which, when traversed, lead us back to the initial vertex without going through any vertex twice; and 4) computation of the face normal vector from the equation formed by the plane described by the face's vertices.

III. TASK PLANNING

The role of the task planner is to take task-level specifications from the user and generate manipulator-level specifications which can be used for simulation or sent to the robot controller. Task specifications may appear in various forms ranging from a pair of initial and goal states to an explicit sequence of subtasks. If only a goal is given, the task planner would need substantial domain knowledge in order to generate sequence of subtasks. An intermediate step might be to let task planner check task specifications and provide recommendations. Task planning can be carried out in two steps: task synthesis and robot motion synthesis. At the top-level is a task manager responsible for the selection and coordination of task skeletons, procedures to perform specific subtasks. It is also plausible to use a robot-level programming language between task synthesis and robot motion synthesis, i.e., task specification (high level) --> robot program (medium level) --> robot motion (low level). The robot-level programming should be accessible by the user.

Robot motion synthesis depends largely on type of applications. For example, assembly operations require compliant and guarded motion, while arc welding and spray painting operations require accurate motion and trajectory control. Object and world model is indispensable in robot motion synthesis. We attach homogeneous transform to symbolic spatial relation to express quantitative relation among objects and robot. The object and world model changes

dynamically as task goes on. An efficient and elegant way to update the model is by message sending in object-oriented paradigm. Collision avoidance is another important problem in robot motion planning which will be discussed in later sections.

We use a typical spray-painting robot to illustrate task synthesis and robot motion synthesis. Assume that five faces of a large rectangular block are to be painted except the face attached to a fixture which rotates the block.

A. User Interface

The user interface elicits information that is required from the user in order to accomplish the task. Typically, TARPS requires to know about the environment and objects. The user supplies data to completely define the object size, shape and location through the object representation primitives. This helps TARPS to configure the object and environment model. Besides information required to define the environment, the user also supplies some parameters that are required for the performance, monitoring and analysis of the task.

B. Task Synthesis

The task synthesizer acts as a scheduler and utilizes the relevant plan information for the performance of the task. The input to the task synthesizer comprises user-specified parameters defining the environment and the task. A given task, e.g., paint the block, is decomposed into a sequence of subtasks using heuristic planning rules, which are primarily concerned with the selection of task parameters to obtain a satisfactory task performance. One such heuristic rule is that "adjacent faces of an object are painted in sequence". The selection of adjacent faces, however, further depends on the constraints on the mobility of the work-piece and the reachability or work-envelope of the robot. After decomposition, each subtask is considered independently and manipulator paths are generated for each subtask in accordance with subtask constraints. An example of such a plan decomposition sequence for painting the block is: (1) paint face_1, (2) rotate block (+90), (3) paint face_2, (4) rotate block (+90), (5) paint face_3, (6) rotate block (+90), (7) paint face_4, (8) paint face_5.

C. PATH GENERATION

Generally speaking, for painting robot there are two types of motion: free motion and paint motion. Free motion comprises those motions between home position and initial positions of subtasks. Collision avoidance is usually the only concern in free motion. Paint motion, on the other hand, requires more accurate trajectory control. For example, the spray gun must always be perpendicular to the surface to be painted, and the distance between spray gun and the surface must remain constant. Once task parameters such as initial configuration, etc. have been determined, the task skeletons are responsible for calculating the path of the manipulator that can satisfactorily accomplish the task.

1. Motion Planning

Once the manipulator is at the initial position it is ready to start performance of the task. The path followed by the end-effector depends on the "paint-patterns." A paint pattern is the path that a spray gun attached to the end-effecter moves to deposit paint on the work-piece. Such a pattern depends on various parameters like shape of the work-piece, material properties of the paint, etc. Until now, generation of a painting pattern has been mostly experimental by using a guiding system. Such a trial and error approach is often time consuming and inefficient. We felt it necessary to develop an algorithm for the painting process. Algorithms for painting planar surface have been developed; those for painting curved surfaces can use polyhedral approximation and are under development.

Once a particular face of the work-piece has been painted, there is relative motion between the robot and the work-piece in order to position the manipulator at a suitable initial position to paint the next face as scheduled by the task planner. Fig. 2 shows the robot path from home position to the initial position for painting face_1. The inter-subtask motion has to take into account the following two factors: 1) Avoidance of collision during the manipulator and/or work-piece motion and 2) movement of the work-piece between subtasks to provide an easy access for the manipulator to paint the relevant face subject to the constraints on the freedom of work-piece to move in the workplace. Inverse kinemetics and collision detection are computed in this phase.

2. Collision Avoidance

There have been many studies relevant to the planning of a collision-free path. Two approaches have been used most often. One approach is "hypothesize-and-test" method which focused on algorithms for detecting collision among solids. Another approach consists of explicitly representing the set of those robot configurations which are collision-free [6],[7]. However, an efficient algorithm for computing a collision-free path for general robots is still not available.

We employ two strategies to plan collision-free paths in TARPS. The first one is using an heuristic method to plan a collision-free path which is similar to the second approach mentioned above. The other uses a collision detector to detect a possible collision during simulation. Whether the collision detector need to be executed can be determined by the high level task manager or by the user. Since two different motions are used here, we developed different algorithms for free and paint motion. In free motion we need to make sure that the end effector is in the safe space. In paint motion we consider other links since end effector is always some distance away from the workpiece. In inverse kinemetics computation, usually multiple solutions are obtainable. In such case we can select a collision-free solution. If none of them is collision free, then a different path must be generated.

IV. SIMULATION AND GRAPHICS

To evaluate and ensure good performance of the robot task planning we need to have the capability of analyzing the kinematics and dynamics of the robot manipulator. This also calls for the capability of presenting actual

ORIGINAL PAGE IS
OE POOR QUALITY

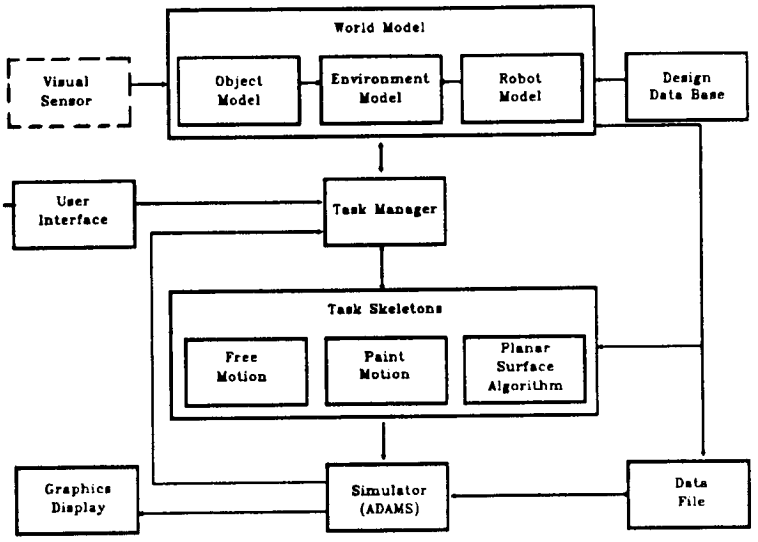


Figure 1: TARPS system architecture

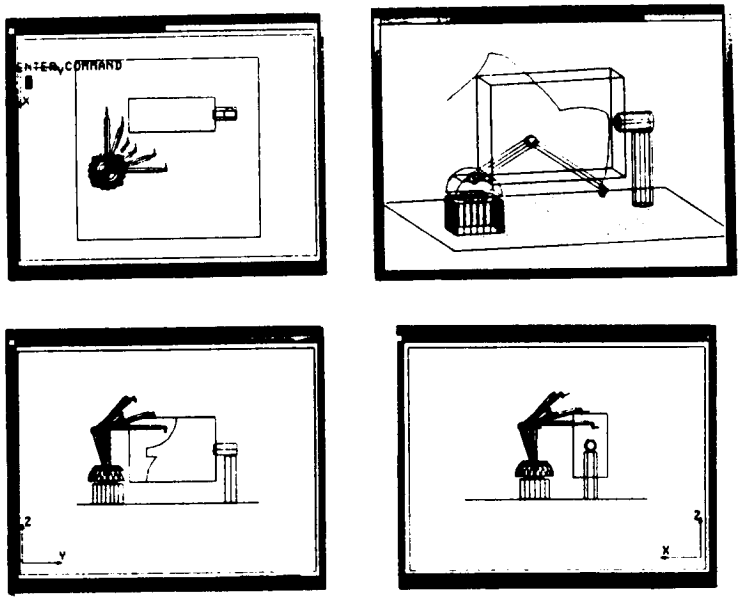


Figure 2: The complete (collision-free) motion sequence from x-direction to y-direction.

robot and workpiece motion graphically. The software used for this purpose must be extensible to encompass special requirements, e.g., nonlinear force and torque calculations, in the form of user-defined functions or subroutines. The Automatic Dynamic Analyses of Mechanical Systems (ADAMS) was selected as our simulation tool which provides us with such facilities. The actual equations governing the manipulator kinematics and dynamics can be programmed in ADAMS which can then be interfaced and run as a process when required by the LISP-based planner. The results of the simulation can be analyzed by the user or planning program to decide on the actions to take.

V. CONCLUSION

In this paper we have presented a prototype of Task-Level Robot Programming and Simulation System. The key issues addressed here are world modeling, object representation and collision-free task planning. We adopted a surface-based object representation for the reasons that it is ideal for sensor-based robot control and is easily accessible from CAD database. Task planning is based on a hierarchical approach while collision problem is taken into consideration during path synthesis. When sensor systems are incorporated, parts localization techniques can be applied to obtain actual position and orientation of the objects. It is our belief that task-level programming will be useful in a wide variety of applications, and we are also investigating the parallel hardware architecture for task-level systems.

VII. REFERENCES

1. ADAMS User's Manual, Mechanical Dynamics, Inc., 1985.
2. D.A. Ballard & C.M. Brown, Computer Vision, Prentice-Hall, Inc., New Jersey, 1982.
3. P.J. Besl & R.C. Jain, "Three-Dimensional Object Recognition," ACM Computing Surveys, 17(1):75-145, March, 1985
4. B. Bhanu & C-C. Ho, "CAD-Based 3D Object Representation for Robot Vision," Computer, 20(8): 19-35, August 1987.
5. M. Brady, J.M. Hollerbach, T.L. Johnson, T. Lozano-Perez & M.T. Mason, (Eds.), Robot Motion: Planning and Control, The MIT Press, Cambridge, Mass., 1983.
6. R.A. Brooks, "Solving the Find-Path Problem by Good Representation of Free Space," Proc. 2nd AAAI Conf., Carnegie-Mellon, August 1982.
7. T. Lozano-Perez, "Automatic Planning of Manipulator Transfer Movements," IEEE Trans. on System, Man, Cybernetics, SMC-11, 10, 1981.
8. T. Lozano-Perez, "Robot Programming," Proc. IEEE, 71(7): 821-841, July 1983.
9. R.P. Paul, "Manipulator Cartesian Path Control," IEEE Trans. on System, Man, and Cybernetics, SMC-9, 11, 1979.
10. R.P. Paul, Robot Manipulators, Mathematics, Programming, and Control, MIT Press, 1981.
11. A.A.G. Requicha, "Representations for Rigid Solids: Theory, Methods, and Systems," ACM Computing Surveys, 12(4):437-467, Dec. 1980.
12. D. Weinreb & D. Moon, LISP Machine Manual, Symbolics, Inc., 1981.

N88-16388

GOAL DRIVEN KINEMATIC SIMULATION OF FLEXIBLE ARM ROBOT FOR SPACE STATION MISSIONS*

P. Janssen
A. Choudry
Center for Applied Optics
University of Alabama in Huntsville
Huntsville, AL 35899

ABSTRACT

Flexible arms offer a great degree of flexibility in maneuvering in space environment. We have studied the problem a space station based Flexible Arm Robot to transport an astronaut for EVA (extra vehicular activity). In particular we developed the inverse kinematic solutions of the multilink structure. Our technique is goal-driven and can support decision making for configuration selection as required for stability and obstacle avoidance. Details of this technique and results will be presented.

*Work supported in part by NASA Grant #NAGW-847 and State of Alabama Research Council.

PRECEDING PAGE BLANK NOT FILMED

N88-16389

HEURISTIC SEARCH IN ROBOT CONFIGURATION SPACE USING VARIABLE METRIC

Ben J. H. Verwer
Faculty of Applied Physics
Delft University of Technology
The Netherlands

ABSTRACT

Intelligent path planning methods are of utmost importance for robots operating in space. In unexpected situations, reliable solutions must be generated. Trial and error may not suffice, nor does active human involvement. An autonomous robot path planning method is not yet availabe. Obstacles are, if at all, detected but not avoided.

We propose a method to generate obstacle free trajectories for both mobile robots and linked robots. The approach generates shortest paths in a configuration space. The metric in the configuration space can be adjusted to obtain a tradeoff between safety and velocity by imposing extra costs (= distance) on paths near obstacles.

A configuration space is a space in which each point corresponds to a unique position and shape of the robot in real space. The number of dimensions of the configuration space equals the degrees of freedom of the robot. E.g., a mobile vacuum cleaner has 3 degrees of freedom (2 translational and 1 rotational), an industrial manipulator with 6 rotational joints has 6 degrees of freedom. A survey on configuration spaces was presented by Lozano-Perez in [1]. A point in configuration space may be allowed or forbidden; forbidden if the robot in real space would collide with an obstacle or with itself, else allowed.

In [2] it was suggested to quantize the configuration space and to obtain a shortest path using a constrained distance transformation. Distance waves were propagated until convergence occured, a rather time-consuming procedure.

We now propose a standard heuristic search method in the configuration space: the A*-algorithm [3]. From a given start point, costs are propagated unitl the goal-point is reached. A suitable heuristic is the lenght of the shortest path to the goal, assumed that obstacles are not present.

Metrics which can be used are approximations of the euclidean metric (like the chamfer metric, see Borgerfors [4]) or non-homogeneous metrics. We

PRECEDING PAGE BLANK NOT FILMED

PRECEDING PAGE BLANK NOT FILMED

propose to use the latter, to be built of chamfer distances, multiplied by a factor dependent on the distance to the obstacles. The distances to the obstacles can be calcualted by a standard distance transformation. The minimum distance in the real space from the robot to the obstacles is decisive for the metric in the configuration space.

In the surroundings of obstacles local distances are larger. The robot will avoid these areas whenever possible. During execution of the path the robot is given a fixed velocity in the configuration space. The local metric provides a slow speed near obstacles and a high speed away from obstacles. If the goal and start are defined as "obstacles" in the determination of the metric an acceleration from the start and a deceleration towards the goal is obtained simultaneously.

Note that our approach differs from the penalty approach [5]. in the penalty approach the finding of a path can not be guaranteed, because only local information is used. With the A*-algorithm however, the shortest and safest path is always found.

References:

[1] T. Lozano-Perez, "Spatial Planning: A Configuration Space Approach", IEEE Trans. Computers, Vol. C-32, No. 2, February 1983.

[2] P. W. Verbeek, L. Dorst, B. J. H. Verwer and F. C. A. Groen, "Collision avoidance and path finding through constrained distance transformation in robot state space", Proc. Intelligent Autonomous Systems, Amsterdam, Holland, December 8-11, 1986.

[3] P. E. Hart, N. J. Nilson and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths, IEEE Trans. on Systems Science and Cybernetics, vol. SSC-4, no. 2, pp 100-107, 1968.

[4] G. Borgefors, "Distance transformation in digital images", Computer Vision, Graphics and Image Processing, vol. 34, pp 344-371, 1986.

[5] K. S. Fu, R. C. Gonzalez and C. S. G. Lee, "Robotics, Control, Sensing, Vision and Intelligence", MacGraw-Hill, 1987, page 512.

N88-16390

A Multiprocessing Architecture for Real-Time Monitoring

Thomas J. Laffey
James L. Schmidt
Jackson Y. Read
Simon M. Kao

Lockheed Artificial Intelligence Center
2710 Sand Hill Road
Menlo Park, CA 94025
(415)-354-5209

Abstract

This paper describes a multiprocessing architecture environment for performing real-time monitoring and analysis using knowledge-based problem solving techniques. To handle asynchronous inputs and perform in real time, the system consists of three or more separate processes which run concurrently on one or more processors and communicate via a message passing scheme. The Data Management Process gathers, compresses, scales and sends the incoming telemetry data to other tasks. The Inference Process consists of a proprietary high performance inference engine that runs at 1000 rules per second using telemetry data to perform a real-time analysis on the state and health of the Space Telescope. The I/O Process receives telemetry monitors from the Data Management Process and status messages from the Inference Process, updates its graphical displays in real time, and acts as the interface to the console operator. The operator sees a hierarchy of displays which can be traversed using a mouse, and on which the user can display graphs of the monitors. The multiprocessing architecture has been interfaced to a simulator and is able to process the incoming telemetry in "real-time" (i.e., several hundred telemetry monitors per second). In this paper we will also describe why commercial knowledge-based building tools are not well suited for real-time domains, thus forcing us to develop our own proprietary shell. The system has been applied to the real-time monitoring of telemetry data from the NASA Hubble Space Telescope (HST) and the application will be described in another paper at this conference.

Introduction

As the application of knowledge-based systems evolves from an art to an engineering discipline, we can expect more challenging applications to be addressed. Some of the most challenging and interesting environments are found in real-time domains.

A knowledge-based system operating in a real-time situation (e.g., satellite telemetry monitoring) will typically need to respond to a changing task environment involving an asynchronous flow of events and dynamically changing requirements with limitations on time, hardware, and other resources. A flexible software architecture is required to provide the necessary reasoning on rapidly changing data within strict time requirements while accommodating temporal reasoning, non-monotonicity, interrupt handling, and methods for handling noisy input data.

The Problem

Like other existing satellites, the NASA Hubble Space Telescope (HST) has not been designed to to be an autonomous spacecraft. Its engineering telemetry will be monitored for vehicle health and safety 24 hours a day by three shifts of operators in the ST Operations Control Center (STOCC) at the NASA/Goddard Space Flight Center in Greenbelt, Maryland.

Six operator workstations (four to monitor the major subsystems and two for command and supervision) will be used to monitor the incoming telemetry data. Each workstation consists of two color CRTs which display numeric values, updated in real time.

- On one CRT the operator can bring up a page of formatted telemetry data (where a page consists of about 50 different monitor mnemonics and its associated value) or a page consisting of a chronological history of events that have occurred (e.g., a monitor out of limits)
- The other CRT is a slave to any other console and can be used to display what is being shown at another workstation

For the HST there are 4,690 different telemetry monitors in 11 different formats available for interpretation. In normal operating mode, each monitor is sampled at least once every two minutes, with some being sampled many times during that interval. The telemetry format may be changed manually by ground operations or autonomously by the HST under certain situations. The telemetry data is subject to a variety of problems including loss of signal and noise in the transmission channel.

As in any large system, the job of the console operator is difficult because of the complexity of the HST and because it is hard to determine the exact state of the satellite at any time due to the massive amounts of data arriving at such short intervals and the ever present possibility of non-nominal behavior.

Why Commercial Tools are Inadequate for Real-Time Monitoring

Real-time domains present complex, dynamic problems because of their dependence on the time factor. A real-time expert system must satisfy demands that do not exist in conventional domains. Current shells are not generally appropriate for real-time applications for the following reasons:

1. The shells are not fast enough
2. The shells have few or no capabilities for temporal reasoning
3. The shells are difficult to integrate *in an efficient manner* with conventional software
4. The shells have few or no facilities for focusing attention on important events
5. The shells offer no integration with a real-time clock
6. The shells have no facilities for handling asynchronous inputs
7. The shells have no way of handling software/hardware interrupts
8. The shells cannot efficiently take inputs from external stimuli other than a human
9. The shells cannot guarantee response times
10. The shells are not built to run continuously

We next describe a monitoring system called *L*STAR* (for Lockheed Satellite Telemetry Analysis in Real Time) being built to aid the HST console operator in performing the real-time monitoring and analysis of telemetry data from the HST. *L*STAR* runs on a DEC VAXStation II/GPX running under VMS and uses data produced by the BASS Telemetry System at the HST Hardware/Software Integration Facility (HSIF) in Sunnyvale, California.

Solution Method

Three separate processes are used for the real-time analysis of rapidly changing satellite telemetry data. Each of the processes operates independently and communicates information via message passing. (NOTE: We use the terms *process* and *task* interchangeably in this paper.) The different processes are shown in Figure 1:

- INFERENCE PROCESS — used to analyze the dynamic data by means of forward or backward chaining rules
- DATA MANAGEMENT PROCESS — used to gather, scale and compress the incoming telemetry data

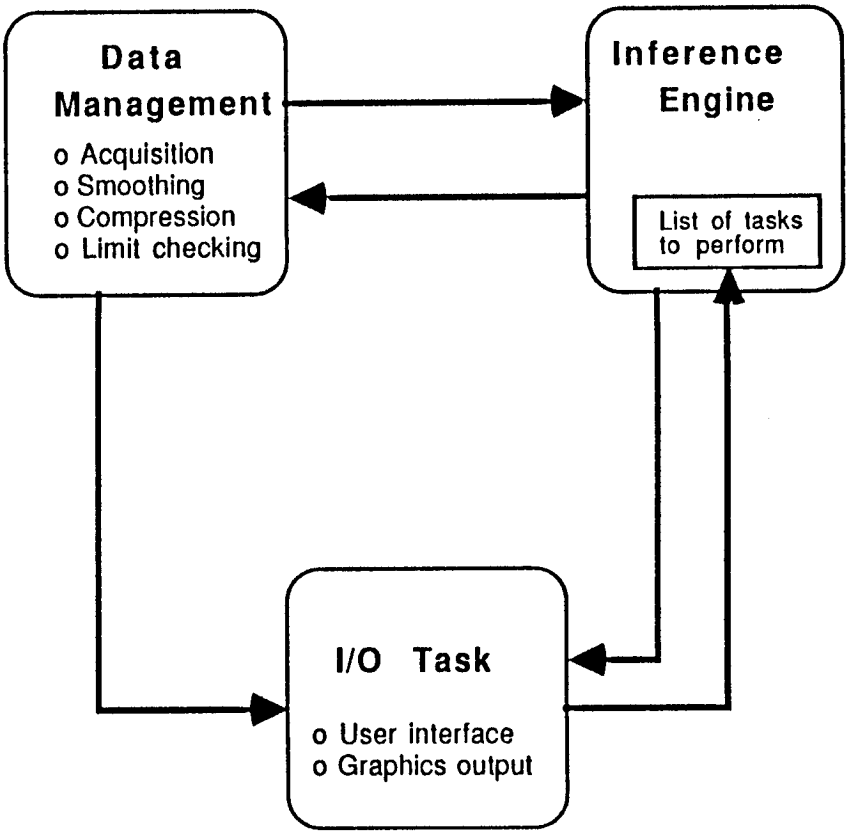


Figure 1: Software Architecture

- I/O PROCESS — used to provide an interface (including real-time graphics) to the operator

Having three independent tasks allows us to distribute the system across different processors. When all three tasks are operating on one processor, the timesharing facilities of the operating system take care of scheduling when each is run.

If all the tasks were done in one process, i.e. sequentially, the Inference Process could not be reasoning with existing data while the Data Management Process was getting new data, or while the I/O Process was performing screen output. The main purpose of having separate tasks was to give the Inference Process complete freedom from input/output worries and let its limiting factor be the processing power of the CPU on which it is resident.

Mailboxes are used for the message passing between tasks. They are used as fast, one-way, in-memory channels for communication of data. Using this mechanism, the Inference Process is only slowed by having to read or write to the mailbox.

A typical scenario follows: The Inference Process examines its knowledge base and sends a set of messages to the Data Management Process indicating which telemetry monitors it needs to perform its analysis. It also sends messages containing other information the Data Management Process needs to know about each telemetry monitor such as the sampling rate, whether it should be smoothed, the scaling factor, alternate names, and to which telemetry set it belongs.

Incoming telemetry data streams are captured from the flight hardware and after initial preprocessing of the raw data by ground computers are fed to the Data Management Process. After some scaling and data compression, this process sends the data of interest to the Inference and I/O processes. The Inference Process can ascertain, using its knowledge base, if the data correspond to nominal vehicle behavior. The I/O Process consists of a data flow diagram of the flight system software and magnitude vs. time plots of the telemetry data. The plots, which are strip charts updated in real-time using data from the Data Management Process, can appear by using mouse clicks when the cursor is over appropriate parts of the diagram. Should the HST change state or non-nominal behavior be detected, messages will be sent from the Inference Process to the I/O Process and subsequently displayed.

The knowledge in this real-time monitoring system is contained within the rules and frames which make up the knowledge base. The knowledge base, used primarily by the Inference Process, contains the critical telemetry items to be monitored and rules to infer the current state and health of the HST. Of the over 4,000 different telemetry monitors, only a small number (about 10%) are used by the operators to determine spacecraft behavior. If, however, non-nominal behavior is detected, other telemetry monitors might be used to diagnose the problem. This would be done by having a rule fire which causes a message to be sent from the Inference Process to the Data Management Process, indicating which new set of telemetry monitors it needs to know about. Rules can also send messages changing the sampling rate of telemetry at which it is already looking.

Discussion

As more and more complex vehicles are put into orbit, it becomes essential that sophisticated methods for evaluating the health of and diagnosing problems within these vehicles be developed. Real-time knowledge-based systems offer promise as an excellent means of dispersing such information. During development, testing is an important part of the cycle. In doing such testing, system designers have to be able to monitor and diagnose the telemetry streams. This kind of expertise should not have to be independently learned by the vehicle operators after launch, when the developers are out of the picture. Thus, for a system such as the one described in this paper, having the expertise saved for the operations aspect is an invaluable step.

Expert Systems Relations in Space Applications

N88-16391

Michael Brady
Cognitive Systems Lab
University of Alabama in Huntsville
Huntsville, AL 35899

Abstract

Independently designed expert systems that operate in common environment will almost certainly share some resources and data. They will also be connected in a network of interdependencies, each one reliant upon the others. Measures must be taken, therefore, to insure that these expert systems can communicate with one another.

This paper addresses the problem of expert systems relations as they pertain to space applications. First, these systems will be categorized and the relationships between them will be analysed. Then, an expert systems cooperation paradigm will be proposed. This paradigm will address various types of communication and coordination issues in an attempt to create a general model applicable in a variety of situations.

Introduction: The Common Goal

The interactions between expert systems that one might expect to find on a modern spacecraft are likely to be as complicated as the systems themselves. These systems must interact, however, in order to achieve their common goal of a successful mission. Like any team, these systems should each do what they do best, while assisting their teammates as much as possible.

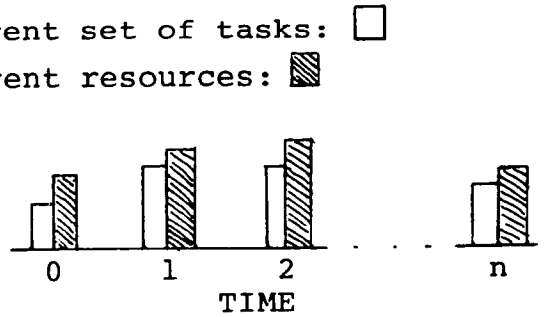


fig. 1 Mission Model

A simple model of a mission is depicted in fig. 1. The terms "task" and "resource" are used in the most general sense. Example tasks include life support systems, experiments, and fixing a

broken camera. Example resources include oxygen, electrical power, and tools. The common goal of the expert systems, then, is to cooperate in order to insure that at any given point in time the spacecraft's available resources either meet or exceed the requirements of the current set of tasks.

The Expert Systems Cooperation Paradigm

Ten categories of expert systems are listed in [1]. This list can be condensed into four classes which one would be likely to find on a spacecraft:

Diagnosis	--	Monitoring and interpreting sensory data. Possibly making predictions based on that data, ultimately trying to identify any problems that might affect the resource pool.
Scheduling	--	Developing a plan whereby all tasks are completed taking into consideration resource constraints.
Repair	--	Helping provide for resource repair either directly or in the form of advice.
Control	--	Controlling mission tasks in accordance with the schedule.

Given these four classes, the author proposes a paradigm for expert systems cooperation depicted in fig. 2. The "Physical System" may be the spacecraft as a whole or any subsystem therein. The paradigm is, therefore, meant to be valid in many different applications of varying scope. It is also possible to ignore any components of the paradigm that are not needed. If, for example, some subsystem does not have a repair expert system, that module and its associated data (the Symptoms and Repair Reports) can be ignored.

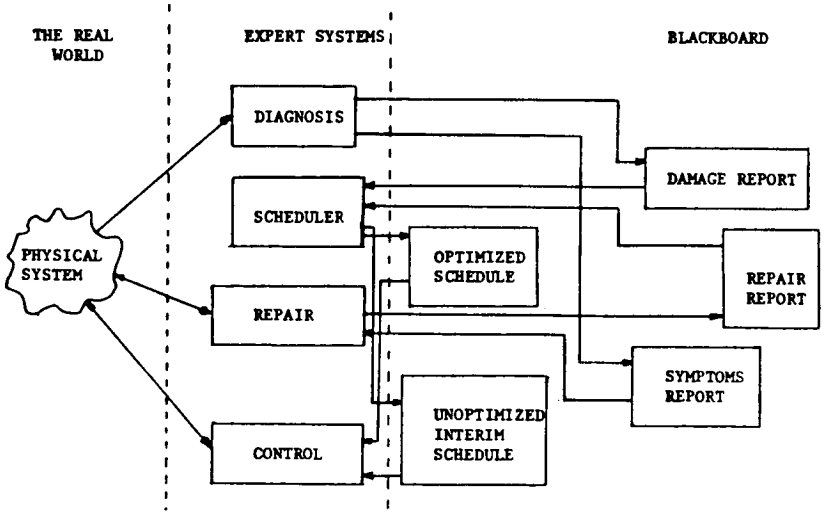


fig. 2 The Expert Systems Cooperation Paradigm

The paradigm utilizes a blackboard architecture, as described in [2], to facilitate expert systems cooperation. According to this scheme, each system monitors the blackboard, looking for something it can contribute. For example, when a Symptoms Report is written by the Diagnosis system, the Repair system will become active. Conversely, whenever the Symptoms Report is empty, the Repair module is inactive because there is nothing it can contribute. It is important to note that all communication between expert systems is through the blackboard.

In a typical case, considering the complete paradigm, both the Diagnosis and Control modules are active at the outset of the mission. At this point all the data objects except the Optimized Schedule are empty. When the Diagnosis system observes or predicts some abnormal change in the resource pool, it writes a Damage Report for the Scheduler and a Symptoms Report for the Repair system.

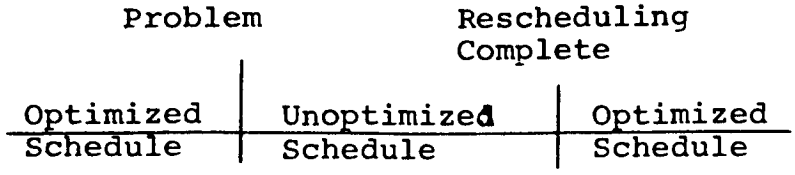


fig. 3 Unoptimized Interim Schedule

At this point there is a problem, however. The Scheduler must revise the optimized schedule, taking into consideration the resource changes, but what about the tasks that must continue during the rescheduling process? Moreover, the Scheduler cannot operate without a definite start time and a definite set of tasks to consider. Clearly, there must be an Unoptimized Interim Schedule for use in the time between the problem diagnosis and rescheduling completion (see fig. 3). The time of this interim should be an upperbound of the runtime of the Scheduler. The Unoptimized Interim Schedule should include as many tasks of the highest priority as can be accommodated. The Control system, then, will use the Unoptimized Interim Schedule whenever one exists. This process is the same when the Repair system, via a Repair Report, signals the Scheduler that the Resource problem has been fixed.

Conclusions

This paradigm will handle multiple resource faults, but includes no methodology for the isolation of or recovery from catastrophic failures. This sort of problem should be handled by the individual expert systems, and is independent of the paradigm in question. The paradigm will, however, help system designers plan the input to and output from their systems. It will also, hopefully, provide a useful framework for the development of the system or subsystem as a whole.

References

1. Nii, H. P., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," "AI Magazine," Summer, 1986, pp. 38-52

2. Waterman, D. A., "What Expert Systems Have Been Used For," A Guide to Expert Systems, 1986, pp. 32-48

N88-16392

PROBLEM SOLVING AS INTELLIGENT RETRIEVAL FROM DISTRIBUTED KNOWLEDGE SOURCES

Zhengxin Chen

Department of Computer Science,
Louisiana State University,
Baton Rouge, LA 70803

Abstract. *We propose a model of problem solving by dynamically distribute the knowledge sources to several processors in a controlled manner. Example is given, the features of this approach are also summarized.*

Introduction

Recently, intelligent distributed systems have drawn much attention. Researches in distributed artificial intelligence (DAI) have focused on cooperative solution of problems by a decentralized and loosely coupled collection of knowledge sources (KSs), each embodied in a distinct processor node [18]. Most previous works in DAI deal with distributed problem solving techniques, for instance, the investigation of phases of problem decomposition, sub-problem distribution, sub-problem solution, and answer synthesis [16]. In this paper we investigate distributed computing in intelligent systems from a different perspective. From the viewpoint that problem solving can be viewed as intelligent knowledge retrieval, we propose the use of distributed knowledge sources in intelligent systems. Owing to space limitation, no technical detail is given in this paper.

A model that integrates knowledge

We start from a cognitive model for knowledge retrieval reported earlier [2,3]. Information chunks or pieces (will be referred to as documents) are acquired, mapped into internal structure and integrated into an overall knowledge base, while the documents (which form the sources of the knowledge) are still identifiable. Notice that this model is very general. The documents may be either English-like texts or numerical data sets, and may have quite heterogeneous structures. The mapping mechanism may also vary a lot. For instance, it may be a natural language understander to "understand" the natural language-like input or a kind of data analyzer to analyze the input numerical data. These internal structures (i.e., the result of the mapping) are referred to as knowledge, integrated to an overall knowledge base, and can be retrieved. This kind point of view is consistent with the view that knowledge is condensed information [15]. After knowledge is retrieved, they will be presented in a easily readable form (e.g., by reconstructing the documents) to the user.

Problem solving and knowledge retrieval

The central idea of this report is to relate problem solving to knowledge retrieval. This is a topic which needs further investigation, although it is not new. In fact, the relationship between information retrieval and question-answering system which has been discussed by many authors is basically also true for the relationship between knowledge retrieval and problem solving. According to [8], systems having broad, possibly interrelated data bases whose answer-computation mechanisms is not capable of great depth tend to be called question-answering systems while systems having less-interrelated data bases whose answer-computation mechanism is capable of more depth tend to be called problem-solving systems. Based on this understanding, if a question-answering system is a kind of information retrieval system that understands the texts, it is reasonable to say that a problem-solving system may be realized as a kind of knowledge retrieval system which needs in-depth understanding and handling of the knowledge. Procedurally, a problem solving system utilizes the knowledge in a manner which results in a sequence of retrieval steps. The objective of the problem solving system is to make decisions to

identify and integrate certain parts of knowledge for certain goal(s) and actually use the related knowledge in an intelligent way. The tasks of the decisions are to make a coherent final plan or to integrate various partial solutions, to name a few.

The use of distributed knowledge sources

What is more, frequently it is desirable to retrieve knowledge from more than one knowledge source (KS). For instance, operational system exists in space science which is able to combine evidence from multiple sources [1]. But along with this direction, much work is still ahead. This particularly includes to develop a useful control mechanism to make this scheme work systematically. The rationale of using our model for this purpose can be justified as below. It has been recognized that sufficiency of knowledge is one of the most important requirements in generating some sequence of partial interpretations that culminates in correct complete interpretation [11,12]. In case of lacking the proper tool of handling the entire knowledge at once, we may try to distribute the entire knowledge into several smaller knowledge sources, each of them can be handled by an independent processor. The various knowledge sources serve various documents in our model; the task of intelligent retrieval is to capture the underlying meaning of these knowledge sources handled by the processors. Each knowledge source provides part of the knowledge needed to solve the problem; therefore, an additional task involved in the problem solving process is to control these processors and to force convergence of the solution, and an overall solution based on all the partial solutions can thus be finally obtained.

By "intelligent retrieval" we mean that (1) the problem solver deals with the "internal form" (or meaning) of the knowledge sources, not necessarily its original form; (2) the problem solver is able to use its rule base to handle the partial or conflicting information obtained from different knowledge sources. Therefore, even though each node has only a limited view of the input data, the problem solver is able to integrate the partial solutions and to convergent to a final solution.

The architecture of our problem solver is explained in Fig. 1. The conceptual memory serves a role of index of the knowledge sources; it is used in integration knowledge from different knowledge sources as well as retrieval of these knowledge sources. The rule base provides rules for integration of knowledge sources.

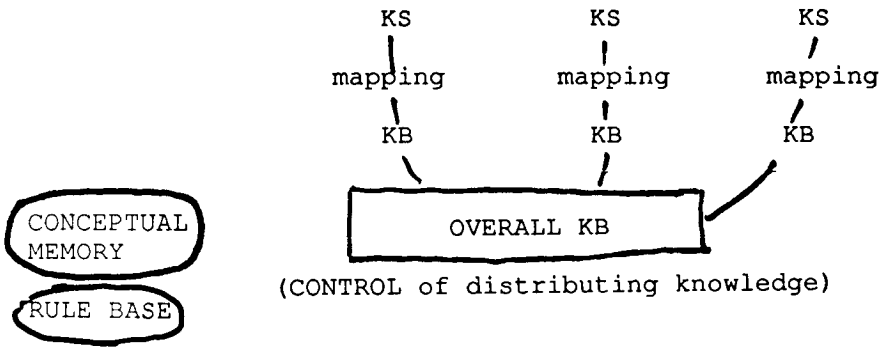


Fig. 1

The fundamental idea is going to be illustrated as below. Previously we described in model which concerns the problem of generating a plan to access heterogeneous numerical database dealing with observational data. In the following we consider another application, which concerns qualitative reasoning on partial results obtained from distributed quantitative processors (part of this work was reported in [5]). This second application, which handles the conflicting information obtained from partial solutions, is more interstring.

This approach utilizes the original model in a "reversed" manner. Traditionally, in a knowledge-based system, knowledge is acquired first, and serves as the environment of

solving the problems. In our approach, *input data are treated as knowledge source*, and, consequently, solving this specific problem means to understand the data, i.e., to *intelligently retrieve the knowledge* implied by these data. This also means to distribute data (instead of problem) and assign them as knowledge sources to processors in a *controlled* manner. The function of the processors is to process (or map) them into internal form (or "knowledge").

Our approach is somewhat related to the work of distributed Hearsay-II [11,12], in which a distributed approach of problem solving has been investigated. An interpretation system accepts a set of signals from some environment. Two major questions are how to interpret the data and how to decompose a given interpretation technique for distribution. It is necessary to operate on local databases that are incomplete and possibly inconsistent and to integrate incomplete partial solutions to construct an overall solution. The elimination of explicit synchronization has increased parallelism. Our approach shares some common features with these previous approaches, the difference is only at what is to be decomposed or distributed.

To illustrate, let us consider the solving of the following problem. This problem solver deals with periodically collected observational numerical data which involve a lot of variables. Only one of the variables is considered as system function (dependent variable), the others are treated as independent variables (although they may be somewhat interrelated). The problem is to find, among a large set of independent variables, the most important variables which effect the system function. Algorithms exist to deal with a limited amount of variables, and they can be actually carried out by existing software (for instance, the technique of utilizing entropy data analysis introduced by [9]). Since each time only a limited set of variables can be considered, each time we can only obtain a partial solution. The type of problem discussed in this paper is similar to the data compression schemes for inertial navigation systems discussed in [13], in which frequent data are collected while the computation capability is limited, but the techniqued used here is entirely different.

For this particular problem, our scheme of solving problem through retrieval of distributed knowledge sources can be explained as follows. Data are decomposed into several subsets, each is able to be handled by a single processor. The part of data distributed to a processor (in our current example, in addition to the dependent variable, each decomposed data set includes several independent variables), is viewed as knowledge source associated with it. (The knowledge sources are not necessarily disjoint). Each process can treat its own knowledge source either as a single unit or a set of knowledge sources at lower levels. All these processors can work on its own knowledge source simultaneously and find the most important variables based on this knowledge source. As the result of this processing (or "mapping") is a set of rules which reflect the knowledge implied by this particular set of data. Each assumes its knowledge source is the only existing knowledge to the system, and claims the variables it found are the dominant ones to the whole system. Under this architecture, the type of problem to be solved can be restated as follows: given a set of data which are distributed to the KSs (with arbitrary number), how to determine the limited number (say, 4) of dominant variables from the results of the competing processors?

Basically, our problem solver solves this problem in the following manner. A set of rules maintained in the central node is used to integrate the intermediate results obtained from the processors. Integration includes to handle the conflicting information and draw similarities among the partial solutions provided by the independent processors. A few new sets of data which includes reduced number of variables are thus created; they are treated as knowledge sources and are then assigned to several processors. The number of variables remained in the knowledge sources are thus reduced and finally they are convergent to the solution. There is a centralized control over the knowledge source the processor possesses.

The problem can be solved by following those steps:

1. Decompose the input data into several subsets, each consists several independent variables and the dependent variable. These subsets form the distributed knowledge sources, each of them is associated with processor(s) which is(are) able to process the associated knowledge source in some way. In addition to this kind of decomposition, a set of rules also exists so that these partial solutions may be integrated later by these rules.
2. Retrieve knowledge processed by processors, use rules to corporate information and get rid of conflicting information.
3. Form reduced knowledge sources, and assign back to some processing nodes.
4. Repeat steps 1-3 until a convergent solution set is obtained.

Notice that in step 1 all the processors related to knowledge sources are not necessarily homogeneous. But to simplify the discussion, in the following example, we will assume all the processors take the same form. Notice also that since the number of variables to be considered at each iteration is at least decreased by one, our scheme can guarantee the convergence of the solution, although this does not necessarily means optimal at all (see the conclusion part of this paper).

To illustrate, suppose we have the original data including variables A, B, C, D, E, F (Fig. 2a), but processor is able to handle up to four variables at each time. Suppose based on domain related knowledge, it is able to organize knowledge sources in a way shown in Fig. 2b. After processing these knowledge sources parallelly, it is possible to identify the partial solutions obtained from these knowledge sources, and rules may be used to form "better" knowledge sources which only includes variables A, B, C, or A, B, E. Therefore only variable combination A, B, C, E needs to be considered. The size of the solution set is thus reduced. The final solution of the original problem can be found by processing this data set.

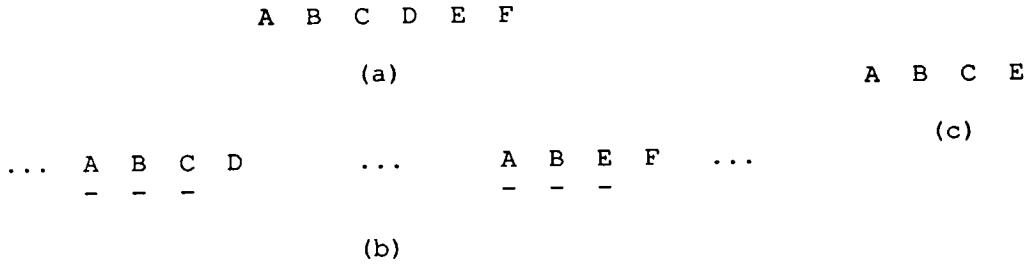


Fig. 2

Features and comparisons with other works

Usually, in distributed problem solving, a single task is envisioned for the system, while distributed processing systems synthesize a network which is able to carry out a number of widely disparate tasks. Since our system is aimed to solve one single task at one time, it is close to distributed problem solver, but the control in our system is not decentralized. Briefly, our scheme has the following features:

(1) Deliberately distribute input data as knowledge sources rather than decompose the task.

(2) Centralized control is only restricted at each knowledge source level.

(3) The problem is solved gradually by reducing knowledge sources, there does not exist a separate phase of answer synthesis.

The system described in this paper may be referred to as distributed knowledge processing system. Although the majority works related in distributed problem solving deal

with knowledge sources which cooperate in the sense that no one of them has sufficient information to solve the entire problem [17], our scheme is the only one which controls the distribution of the knowledge sources (instead of the problems) in a dynamic manner. This is the fundamental difference between our approach and the others.

A systems level approach to distributed processing was suggested in [14], in which a scalable, dynamically reconfigurable architecture was claimed to be necessary. This means a computer with no architecturally imposed performance limits. If this approach is to find a hardware solution, then our purpose is to find a software solution for a similar problem.

Integrating knowledge sources for computer "understanding" tasks was discussed by [6]. Our scheme is similar to that scheme which is a system of cooperating experts running in separate images. But ours is aimed to be a general knowledge integration scheme extended from the existing IR model, and is not restricted to text (written in English) understanding. Therefore, in this sense, ours is more general.

Concluding remarks

The method introduced in this paper does not necessarily generate the "optimal" solution; but, it does provide an acceptable one. We have successfully used the method described in this paper to find the effect of some most important variables to the system function [5]. Moreover, since the method introduced in this paper involves symbolic (qualitative) reasoning attached to numerical processors, it may be viewed as an example of coupling symbolic and numerical computing [10], which has been recently more and more discussed in space science as well as many other research fields.

References

[1] Campbell, S. D. and S. H. Olson, "Recognizing low-altitude wind shear hazards from doppler weather radar: an artificial intelligence approach," *J. Atmospheric and Oceanic Technology*, vol. 4, No. 1, March 1987, pp. 5-18.

[2] Chen, Z., "Some aspects of a cognitive model for information retrieval," *Proc. 18th Pittsburgh Conf. on Modeling and Simulation*, 1987.

[3] Chen, Z., "A language for modeling users virtual machine of information retrieval," *Proc. 1987 IEEE Workshop on Languages for Automation*, 1987.

[4] Chen, Z., "PENDS: a prototype expert numeric database system," paper presented at 2nd AIRES Workshop (AI Research in Environmental Science), 1987.

[5] Chen, Z. et al., "Qualitative reasoning for numerical systems," paper presented at 2nd AIRES, 1987.

[6] Cullingford, R., "Integrating knowledge sources for computer 'understanding' tasks," *IEEE Transactions on Systems, Man and Cybernetics*, vol. SMC-11, No.1, Jan. 1981, pp. 52-60.

[7] Davis, R. and R. G. Smith, "Negotiation as a metaphor for distributed problem solving," *Artificial Intelligence*, 1983, pp. 63-109.

[8] Green, C., *The Application of Theorem Proving to Question-Answering Systems*, Garland Publishing, New York, 1980.

[9] Jones, B. "Reconstructability considerations with arbitrary data", *Int. J. Gen. Sys.*, vol. 12, 1986, pp. 1-6.

[10] Kowalik, J. S. (ed.), *Coupling Symbolic and Numerical Computing in Expert Systems*, North Holland, 1986.

[11] Lesser, V. R. and D. D. Corkill, "Functionally accurate, cooperative distributed systems," *IEEE Transactions on Systems, Man, and Cybernetics*," vol. SMC-11, No.1, Jan. 1981, pp. 81-96.

[12] Lesser V. R. and L. D. Erman, "Distributed interpretation: a model and experiment," *IEEE Transactions on Computers*, vol. C-29, No. 12, Dec. 1980, pp. 1144-1163.

[13] Medan, Y. and I. Y. Bar-Itzhack, "Batch recursive data compression schemes for INS error estimation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-21, No. 5, Sep. 1985, pp. 688-697.

[14] Meier, R. J. Jr., "A systems level approach to distributed processing," *Computer Sciences and Data Systems* (Proceedings of a symposium), NASA Conf. Pub. 2459, vol. 2, pp. 143-172.

[15] Michalski et al.(ed.), *Machine Learning: An Artificial Intelligence Approach*, Vol. 2, Kauffman, 1985.

[16] Smith, R. G. and R. Davis, "Framework for cooperation in distributed problem solving," *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-11, No. 1, Jan. 1981, pp. 61-70.

[17] Smith, R. G., "Report on the 1984 distributed artificial intelligence workshop," *AI Magazine*, Fall 1985, pp. 234-243.

[18] Smith, R. G., "The 1985 workshop on distributed artificial intelligence," *AI Magazine*, Summer 1987, pp. 91-97.

N88-16393

Application of Parallel Distributed Processing to Space Based Systems

J.R. MacDonald, H.L. Heffelfinger
TRW, Huntsville Operations
213 Wynn Drive
Huntsville, Alabama 35807

Abstract

This paper explores the concept of using Parallel Distributed Processing (PDP) to enhance automated experiment monitoring and control. Recent VLSI advances have made such applications an achievable goal. The PDP machine has demonstrated the ability to automatically organize stored information, handle unfamiliar and contradictory input data and perform the actions necessary. The PDP machine has demonstrated that it can perform inference and "knowledge operations" with greater speed and flexibility and at lower cost than traditional architectures. Current automated process and control algorithms use knowledge and inference mechanisms to solve problems which would ordinarily require the expertise of the best human practitioners. In applications in which the rule set governing an expert system's decisions is difficult to formulate, PDP can be used to extract rules by associating the information an expert receives with the actions taken. There are many potential applications for very large scale hardware parallelism in the execution of space based process monitor and control systems.

Introduction

The practical possibilities of large scale parallel machines have been significantly enhanced by recent technological advances in VLSI research and production. Relatively low cost and easy access to VLSI hardware has enabled researchers to more closely examine parallel processing problems in general, and the application of neural nets to real world problems. The application of knowledge based systems to on-line, real-time environments such as automated experiment monitoring and control typically demands large complex systems reasoning. Control of these experiment systems stresses the current space based

PRECEDING PAGE BLANK NOT FILMED

* C-3

computer environments well beyond current technology. Size, reliability, and response time constraints of the space environment quickly overload conventional von-Neumann machine knowledge applications. However, the fine grained parallelism made possible by PDP technology has provided an alternative route to space based process monitor and control.

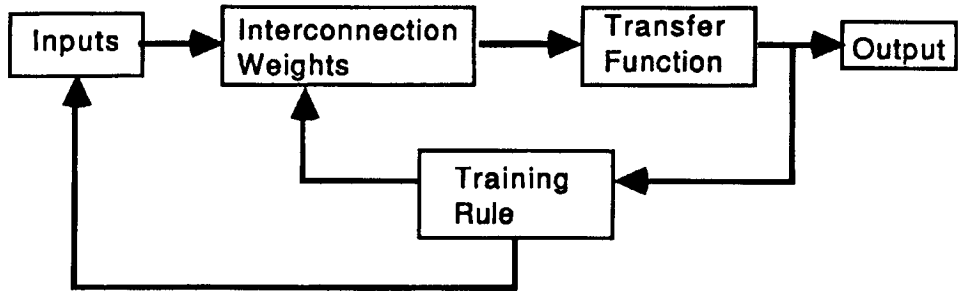
PDP networks are a concept of knowledge representation that consists of a number of processing elements interconnected in a weighted, user-specified fashion. The interconnection weights between the processor nodes are analogous to the memory of a conventional system. Each processing element calculates an output value based on the weighted sum of its inputs. A training rule is used to correlate the input data with the output or desired output (specified by an instructive agent) and adjust the interconnection weights. In this way the network is able to learn patterns or imitate rules of behavior. It is this ability to define and control decision making that would make this sort of system well suited to support space based processing problems. The division of the processing demands among processors makes it possible to achieve linear performance improvements for space based processing applications. This paper will explore PDP network techniques in the application of massively parallel primitive processors to achieve very high speed, fault tolerant, low cost support for the types of problems associated with controlling experiments in a space based environment.

Space Based PDP Architecture

PDP network technology is a concept of computational processing based upon highly interconnected low level processing units. PDP technology seeks to develop and enhance processing capabilities in areas such as real-time high-performance pattern recognition, knowledge processing for inexact knowledge domains, and fast, precise control of VLSI simulation. The aims of this technology are, therefore, clearly related to those of Artificial Intelligence (AI). Over 30 years of research has provided broad processing requirements for pattern recognition, knowledge processing, and simulation of processes. PDP machine research has been directed toward finding solutions to these difficult computing problems.

The PDP architecture is the formal specification of a particular network or bulk system configuration (i.e., the equations of the dynamical system that defines the PDP). This architecture is analogous to a mathematical algorithm or logical procedure being coded and run on a wide variety of computer hardware. The word implementation is reserved for use in describing the engineering process of developing an appropriate PDP architecture and selecting suitable implementation hardware for the specific application.

The PDP conceptual processing abilities are based upon an array of highly interconnected Processing Units (PU)[1,3]. Figure 1 shows a schematic diagram of a PDP network and an individual PU. The network is make up of input units, hidden units, and output units. The hidden units are used to extract progressively more complex features from the input units. This allows more complex tasks to be learned. Note that each PU receives several input signals and has a single output that fans out as input signals to other PUs. Each PU assigns a weight to each input they receive. The sum of the weighted signals determine whether a receiving PU will itself output a signal. Thus this process continues through the PDP machine. The processing cascades through many iterations and the output could be a binary answer to a question or a complicated signal to be transmitted across an external interface. The weights are determined by tuning the system until it consistently produces the right output based upon the system inputs. A typical processing element operation might be:



The PDP adjusts itself by way of feedback control systems that combine stimuli and feedback from the responses to adjust the weights so as to get increasingly correct responses. It is these adaptable, programmable connections that makes the PDP machine special. The PDP machine was initiated in the belief that coupling parallel processing and artificial intelligence could accelerate the rate of progress toward machines that could "learn" and make predictable, weighted decisions within the confines of their knowledge base. The PDP machine encodes knowledge in the connection of its processing units. Each set of connections represents a pattern of values for a few features. Together the features describe environmental states, that is, values that occur in the system's environment. The strengths of the connections encode the frequencies with which each of the different patterns occur in the environment. Thus, the interconnections are used to represent events in the environment. The design goal of a PDP machine for a space based environment should be guided by the following goals:

- Effective use of massively parallel processing.
- Flexibility of interaction with different sensors, actuators, and interfaces.
- Distributed parallel decision-making capabilities.
- Flexibility and ease of expansion of the system capabilities.
- Graceful integration of specific experiment control plans.

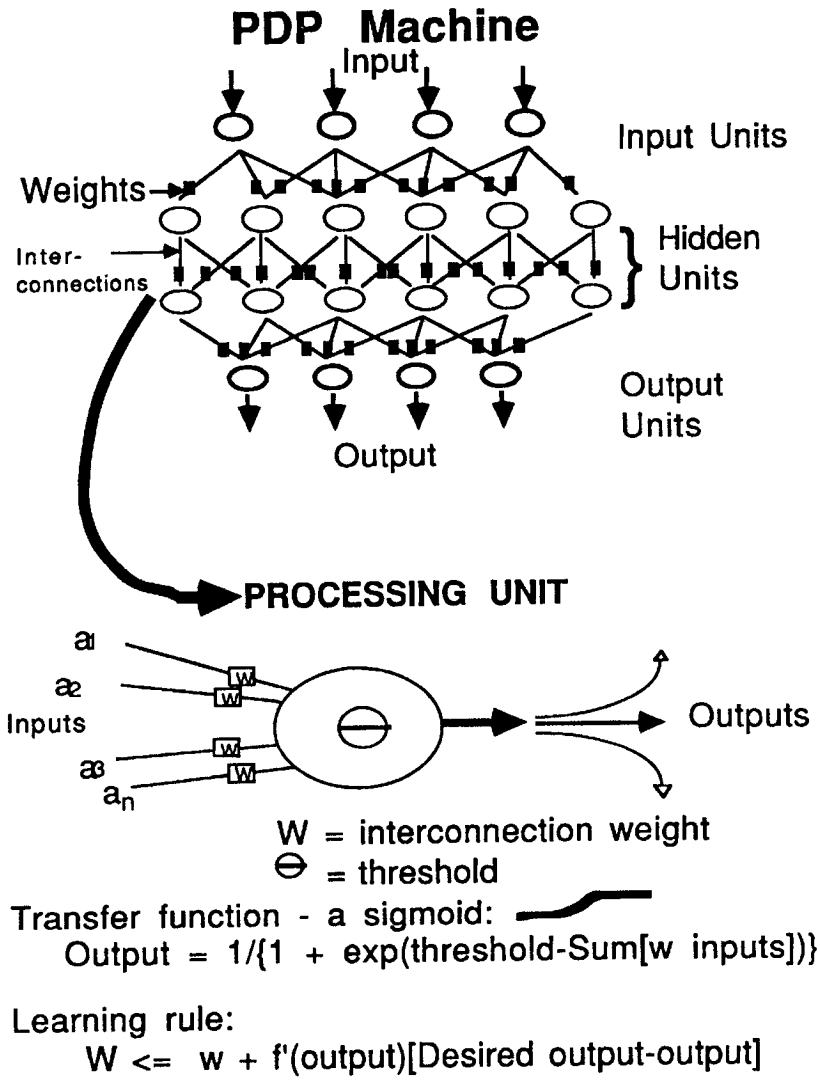


FIGURE 1. Schmatic diagram of a PDP Network

A key approach in the PDP architecture is the use of an adaptive filter. The adaptive filter can accept a real-world analog input and compare it to a stored pattern. The stored pattern represents the desired input level which can then be modified as necessary so that echo-free signals can be produced the correct results. A more abstract argument driving the design of the PDP network is machine efficiency, that is, the optimal utilization of the total computer circuitry. In the more familiar structure of a modern computer (the von Neumann architecture) most of the computer's circuitry in the memory is not in active use most of the time. This is very wasteful from a pure resource management standpoint. The PDP machine addresses this problem by using many processors and memories, but does so

on a much grander scale than most. The parallelism in the PDP machine is extremely fine-grained; it is essentially "data-level parallelism." The fine-grain division of labor and the high speed allows the accomplishment of tasks traditionally outside the operation capacity of conventional systems, among which are constraint search, pattern recognition, and so forth. Even though it fits quite neatly into the definition of an parallel processor system design, the PDP machine goes well beyond most machines in this category in the flexibility of its structure and its amenability to various problem types.

Command and control of typical space based experiments involves both telemetry and status monitoring and the generation of command messages. These tasks are typically carried out in conventional (sequential) computer systems. As systems and environments become more complex, distributed systems become increasingly attractive as well as necessary to achieve higher throughput for a given level of computational power and higher overall system availability. As knowledge-based systems grow in size and scope, they push conventional computing systems to their limits of operation. For space-based experiment control systems, response from a conventional implementation of an expert system may not be practical. Significant performance increases in process monitor and control systems applications could be realized through distributed processing or the use of specialized massively parallel hardware.

Typical space systems computer operations involve monitoring and control processes such as system initialization and shutdown and power system control. Although the performance of tasks such as sensor monitoring, particularly exception monitoring, is often automated, the corrective action - the reaction to anomalies - is typically done by on-board personnel. The exponential increase in system complexity and processing speeds in distributed processing systems pose a serious problem for meaningful, effective human interface as well as timely, effective corrective action. When these factors are coupled with non-linear increases in costs, safety considerations, and longer mission durations, they provide a significant incentive for improved knowledge based system processing concepts and applications. The demands of the space based environment, that is, the real time processing of experiment feed-back and other sensor based data, suggest the necessity for efficient handling of data and telemetry in the event of environment degradation, or sensor failure. The process and control system will need to respond to anomalous events that will be both instantaneous, non-specific, and dependent upon current machine state. The space based processing system must be effective and reliable in its response to both nominal and anomalous events.

A PDP network implementation of an expert system as shown in Figure 2 is well suited for the space based environment. An expert system is designed to explore and symbolically manipulate problems. Expert systems can be distinguished from other artificial intelligence systems in that by design they bring large amounts of knowledge to bear in problem solving. There are two general ways to define an expert systems. One way is by problem domain or

competency. An expert system is a computer system which uses domain-specific knowledge as well as inference procedures to solve problems. The specific problems involved are sufficiently difficult to require significant human expertise in the weighing and evaluation of data. Restated, an expert system operates in a complex domain and is competent at the level of a human expert.

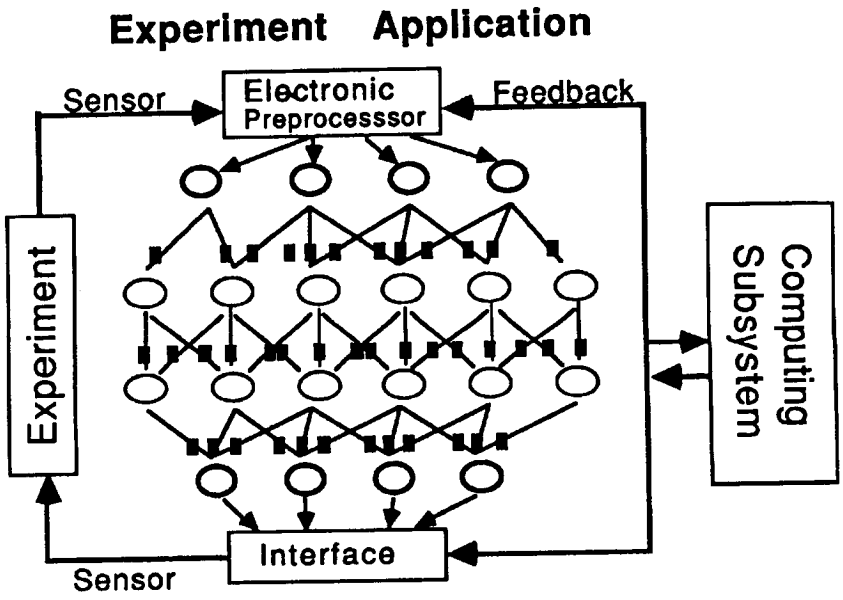


Figure 2. Space Based PDP Expert System

Another way of defining an expert system is by its structure. An expert system consists basically of a knowledge base and a control structure. The knowledge base contains facts about the domain. It also employs heuristics and rules of behavior. The control structure determines the direction of problem solving. In general, control structures provide for goal directed problem solving (solving a problem to reach a certain goal) or data directed problem solving (problem solving that makes use and sense of data available from the domain environment). A related structural aspect of expert systems is a working memory in which interim problem solving steps and other information may be temporarily stored while they are being used in working toward a solution. This structural definition of an expert system is clearly related to PDP networks functional capabilities.

Typical applications of expert modules are in the control and operation of sensors and actuators, interpretation of sensory and feedback data, devising strategies to accomplish proposed tasks and the execution of these strategies. A goal might be to operate complex

space based experiments independent of human intervention for significant periods of time. If this be the case, the long term space based systems goal is to define control mechanisms that will enable integrated experiments to detect error conditions in its working environment and either perform or provide corrective actions. Such systems must be able to interpret and integrate qualitatively different, sometimes incomplete, and sometimes conflicting sensory information. In other words, it must construct an internal model of the real world environment in which the experiment will operate. A general problem-solving technique must be employed. However, specific real-time information must also be integrated in the execution of plans to solve a given task. It must be a control system which is responsible for and capable of independent control execution through parallel and coordinated control of multiple sensors and actuators.

The system must evaluate the outputs from sensors and update the state of the experiment actuators according to the rules established by the principle investigator. Various rules are said to "fire" based on input. An example of a rule for an Earth-bound experiment environment might be " If temperature greater than 150 degrees then power up fan". The integrated computer system is in a constant state of sensing and updating the state of an experiment. A simple experiment environment might involve 1000 such rules. Because of memory requirements and execution speed, even this relatively small knowledge base would involve significant overhead in a conventional von-Neumann machine and could quickly overload the system. The PDP machine offers another method of implementation of constraint based rules. This alternative provides the speed necessary to respond to the space based system needs.

PDP Knowledge Acquisition Paradigm

In the preceding sections we have discussed a parallel processing architecture and its ability to build a knowledge base - to "learn." This learning process is, of course, a complex and controversial problem. Learning algorithms have been proposed as a way to program massively parallel processors [1,2,3,4,5]. Experiment control system programs using appropriate learning algorithms can be automatically decomposed into small sub-tasks. It is these sub-tasks that provide an opportunity to distribute processing across parallel processing units, which translates into the connections of a PDP machine.

The PDP machine assigns a weight to each input it receives; the sum of the weighted signals determines whether a receiving PU will itself fire a pulse, which in turn triggers other PU's. This process cascades through many iterations, and the result, is the output. The PDP machine consist of a number of PU's interconnected in a weighted, user-specified fashion. Each PU calculates an output value based on the weighted sum of its inputs. To program the PDP machine, the input data is correlated with the output or desired output

using a learning rule that will adjust the interconnection weights. In this way the machine learns patterns or imitates rules of behavior and decision making the allows the PDP machine to support space based processing. PDP model as presented in Figure 1 typically consists of many simple processing units that interact using weighted connections. Each unit has a "state" or "activity level", that is determined by the input received from other units in the network. The threshold term can be eliminated by giving every unit an extra input connection whose activity level is fixed. The weight on this special connection is the negative of the threshold, and it can be learned in just the same way as other weights.

The particular PDP machine architecture discussed here is a variation of the Rummelhart et. al. [3] multi-layer perceptron employing the generalized delta rule (GDR). The GDR provides a method of modifying any weight in a network, based on locally available information, so as to implement a gradient descent process that searches for those weights that minimize the error at the output units. The PDP system can learn to associate arbitrary input/output pairs by use of the generalized delta rule. Using this the rule, neural networks can learn to compute arbitrary input/output functions. The mathematics of this learning approach can answer many questions about the weight, adjustment, and summation of the interconnection weights. It can also provide some insight into noise sensitivity, feedback, and system layering. The GDR learning procedure is a generalization of the delta rule procedure that works for networks which have layers of hidden units between the input and output units. Multilayer networks can compute more complicated functions than networks that lack hidden units. However the price that must be paid is a slower learning process as the system explores the possible ways of using the hidden units.

The application of the generalized delta rule as presented in Figure 3 involves two phases. First the input is presented and propagated forward through the network to compute the output value for each unit. This output is then compared with the targets, resulting in an error signal for each output unit. The second phase involves a backward pass through the network during which the error signal is passed to each unit in the network and the appropriate weight changes are made. This second, backward pass allows the recursive computation of the error signal as indicated above. The first step is to compute the error signal for the output units and all of the hidden units. Notice that computation performed during the backward pass is very similar in form to the computation performed during the forward pass (though it propagates error derivatives instead of activity levels, and it is entirely linear in the error derivatives). The GDR generates a gradient descent method for finding weights in any feed-forward network with semilinear units. The learning procedure involves the presentation of a set of pairs of input and output patterns. The system first uses the input vector to produce its own output vector. Then this is compared to the desired output, or target vector. If there is no difference, no learning takes place. If any difference exists, the weights are changed to reduce the difference.

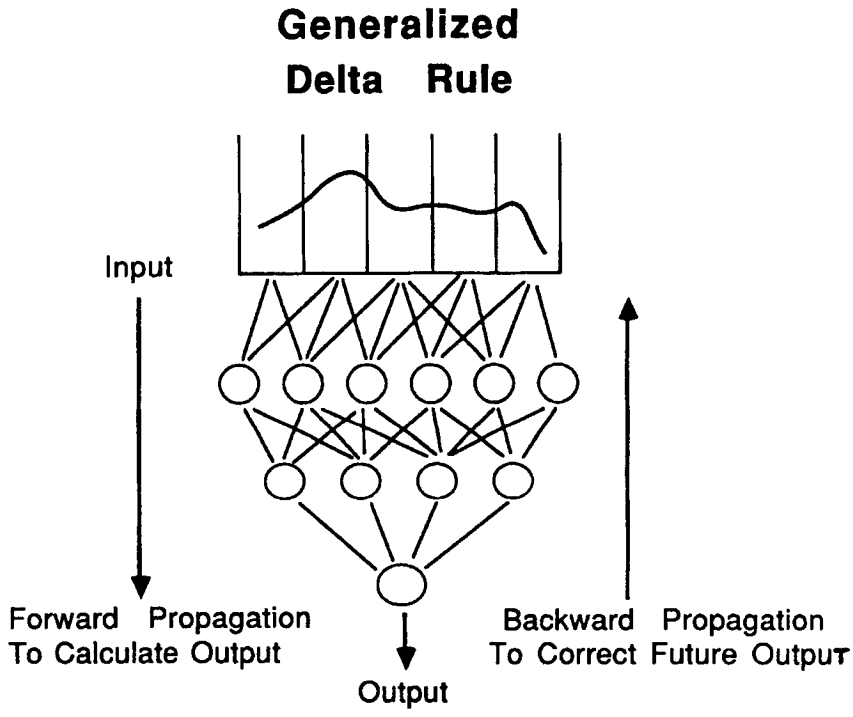


Figure 3. PDP Learning Procedure

When weight change increments are sufficiently small, this learning procedure is guaranteed to find the set of weights that gives the least mean squared error. The delta rule essentially implements gradient descent in sum-squared error for linear activation functions. The central idea of the generalized delta rule is that these derivatives can be computed efficiently by starting with the output layer and working backwards through the layers. The weight on each line should be changed by an amount proportional to the product of an error signal available to the unit receiving input along that line and the output unit sending activation along that line.

From the above learning methodology we see that the PUs in a PDP machine are trained by the cyclic input and output of data vectors. In this way a computer operating in the batch mode can be very effective in training the system. However there is a clear need to provide a real-time interface with a human in order to effect more particular training [5]. The iterative process through which a PDP machine learns is not suited for human interaction. Therefore there is a clear need to both enhance the man-machine interface and develop more efficient data/response patterns in a PDP architecture.

Conclusion

System designs based on the traditional von Neumann approach will be considerably slower than a systems based on the PDP machine simply because of the limited bandwidth of the memory-processor connection. These traditional systems cannot match the flexibility in function that the software-programmable connections, which are the hallmark of the PDP machine, allow. Networks of larger processors, the MIMD concept, can possibly outperform a PDP machine on computation-intensive problems; however, in constraint-intensive applications, they suffer from the same problem as the von Neumann designs. Traditional SIMD machines, such as systolic and pipeline machines, suffer from the problem of requiring a particular structure to solve a problem. Again, this problem is overcome by the flexibility of the PDP machine's communication network.

PDP machines employing fine grained parallelism consist of a number of processing elements interconnected in a weighted, user specified fashion. The interconnection weights act as memory for the system. Each processing element calculates an output value based on the weighted sums of its inputs. In addition, the input data is correlated with the output (or desired output) through use of a training rule that adjusts the interconnection weights. In this way, the network learns patterns or imitates rules of behavior and decision making. Process information is not obtained by passing through a normal process and control algorithm, but is provided by the interconnection structure of the network itself. It is our belief that PDP machines can in fact support high-speed execution of a very large class of space based process monitor and control systems. The number of processing elements in the interconnection network of a PDP machine makes overall network reliability and fault tolerance a key consideration in space based systems. Computer systems employing fine grained parallelism can provide an approach to a number of long standing problems involving space based experiment applications.

References

1) Hitton, G.E. (1987). *Connectionist Learning Procedures.* (Tech. Rep. No. CMU-CS-87-155). Pittsburgh, PA: Carnegie-Mellon University, Department of Computer Science.

2) Hopfield, J.J., Tank, D.W. (1985) *"Neural" computation of decisions in optimization problems.* Biological Cybernetics, 52, 141-152.

3) Rumelhart, D.E., McClelland, J.L., et. al. (1986). *Parallel Distributed Processing.* Cambridge, MA: MIT Press/Bradford.

4) Sejnowski, T.J., Rosenberg, C.R. (1986). *NETtalk: A Parallel Network that learns to Read Aloud.* (Tech. Rep. No. JHU/EECS-86/01). Baltimore, MD: John Hopkins University.

5) Shepanski, J.F., Macy, S.A. (1986). *Manual Training Techniques of Autonomous Systems Based on Artificial Neural Networks.* Redondo Beach, CA: TRW, Unpublished manuscript.

N88-16394

SPACELAB DATA PROCESSING FACILITY (SLPDF) QUALITY ASSURANCE EXPERT SYSTEMS DEVELOPMENT

Lisa Basile (NASA/GSFC/Code 564)
Angelita C. Kelly (NASA/GSFC/Code 564)

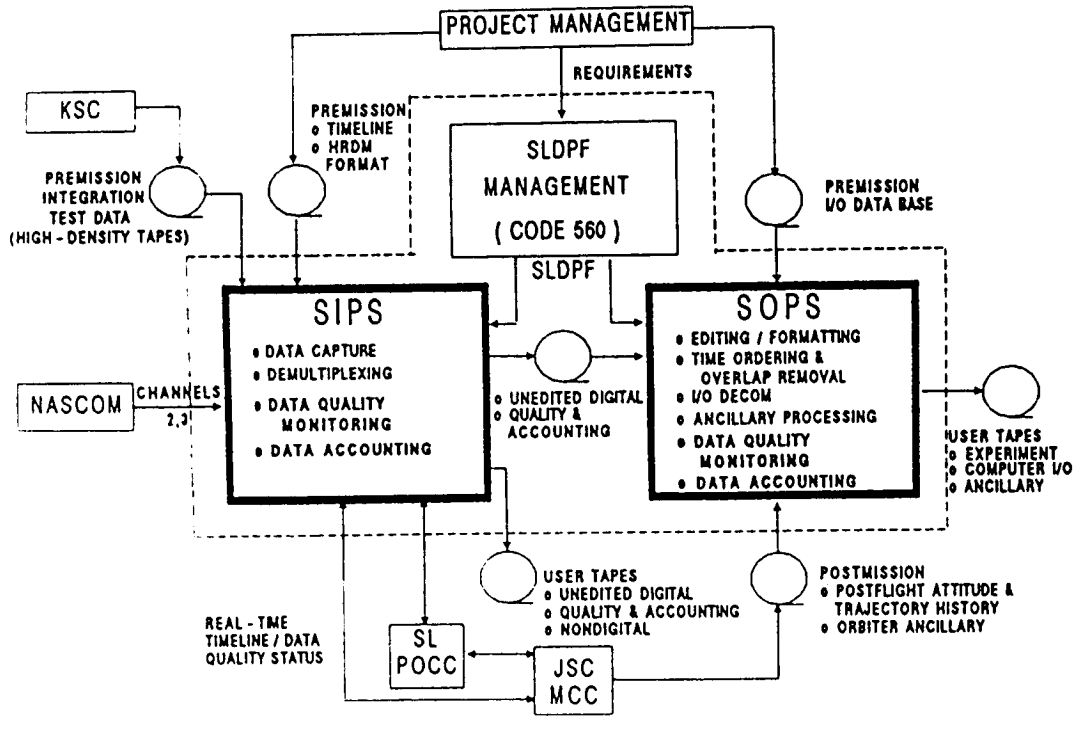
Goddard Space Flight Center
Greenbelt, MD 20771

ABSTRACT

The Spacelab Data Processing Facility (SLDPF) is an integral part of the Space Shuttle data network for missions that involve attached scientific payloads. The SLDPF has developed expert system prototypes to aid in the performance of the quality assurance (QA) function of Spacelab and/or Attached Shuttle Payloads (ASP) processed telemetry data. The SLDPF functions include the capturing, quality monitoring, processing, accounting, and forwarding of data from Spacelab and ASP missions to various user facilities. The SLDPF consists of two functional elements: the Spacelab Input Processing System (SIPS) and the Spacelab Output Processing System (SOPS). The two expert system prototypes were developed to determine their feasibility and potential in the quality assurance of processed telemetry data. The SIPS expert system, Knowledge System Prototype, (KSP), uses an IBM PC/AT with the commercial expert system shell OPS5+. Expert knowledge (from SIPS experts) emulating the duties of quality assurance analysts was implemented. In an interactive mode, a SIPS analyst responds to queries resulting in instructions and decisions governing the reprocessing, releasing or further analysis/troubleshooting of data. Released data is forwarded for further processing on the SOPS Sperry 1100/82. The data are edited, time ordered with overlapping data removed, decommutated, and quality checked before release to the user. The SOPS QA analysts isolate problems and select the appropriate action: either accept the data or request the data to be reprocessed. The SOPS expert system emulates this process by utilizing an expert system shell, CLIPS, and the Macintosh personal computer. To date, these prototypes indicate potential beneficial results; e,g., increase analyst productivity, decrease the burden of tedious manual analysis, provide consistent evaluations of data, provide concise historical records, provide training for new analysts, and expedite the operational training of Spacelab analysts. The logic implemented in the prototypes, the limitations of the personal computers utilized, and the degree of accessibility to input data have led to an operational configuration to be implemented on a SUN 3/160 Workstation. This configuration is currently under development and on completion will enhance the efficiency, both in time and quality, of releasing Spacelab/ASP data.

INTRODUCTION:

The SLDPF processes payload data from Spacelab and ASP missions. The SLDPF functions include the capture, quality monitoring, processing, accounting, and shipping of data to users. The SLDPF is composed of two functional elements: the SIPS and the SOPS. In SIPS, Ku-band channel 2 and/or channel 3 data are captured onto high-density tapes (HDTs). The primary functions of SIPS are the realtime capture, the monitoring of data for quality and status coordination with the Spacelab external interfaces such as the Spacelab Payload Operations Control Center (POCC), the Mission Control Center (MCC), and the Network elements. See Figure below. The data captured, including playback and direct access channel data, are processed to produce Spacelab Experiment Data Tapes (SEDTs) and/or Spacelab Input/Output Data Tapes (SIDTs). To assure completeness and high quality of SIPS processing, analysts currently perform quality assurance and data accounting (QA/DA) analysis by the manual evaluation of Spacelab Quality and Accounting Records (SQARs) aided with information from Spacelab reports and logs. The results of the QA analysis determine the release of SEDTs, SIDTs and Spacelab Quality and Accounting Tapes (SQATs) to the SOPS or to other users. Additional data processing is performed by the SOPS. The data are edited, time ordered with overlap removed, decommutated, quality checked, and shipped to users. The QA/DA analysis is a manual process of evaluating and correlating information from various reports and logs to determine the quality of the data and its status: release or reprocess.



SLDPF INTERFACES

Expert system applications in the Information Processing Division were first considered for their potential to expedite the SLDPF operations, in particular, the QA/DA analyst functions of both the SIPS and the SOPS. The extremely large volume of data from one mission and the short turnaround requirement for delivery to users often makes the QA/DA task demanding and tediously repetitive. The objective of the operational expert systems is to assist the analyst by making decisions and suggesting logical analysis paths based on given data quality information. The strategy formulated to accomplish the prototypes was to use commercial expert system shells, code the QA/DA knowledge bases within the shells and implement them on personal computers. The SIPS KSP uses the OPS5+ Development System with a C language interface installed on an IBM PC/AT. The SOPS Expert System (ES) prototype was implemented with the expert system building tool CLIPS and an in-house-written interface on an Apple Macintosh.

SIPS KSP:

The SIPS KSP is designed to emulate the performance of experienced SIPS QA/DA analysts in the evaluation of Spacelab data quality and accounting information. This function is currently performed through the examination of data quality and accounting reports.

The first task was to gather the analysis expertise of the QA/DA analysts to determine that this area was a practical application for an expert system. The scope of the initial effort was restricted due to the extensiveness of the application and the limitations of the prototype hardware and software configuration. Three stages of analysis were established: initial data evaluation, comparison of initial and redo processing run data, and data trends. Each can stand alone logically but need access to the data and decisions of the others. The use of a database to store data quality and accounting information as well as the decisions of each stage allows the expert system to be divided into independent modules which run with the available memory of the prototype configuration. As each module runs, pertinent data and decisions are written to report files from which database updates and printed summary reports are generated. The code for database control, the Front End module, grew to include database creation and loading, data validation, data maintenance, data selection, expert system module selection, and expert system report selection.

The rule-based expert system tool OPS5+ was used to develop the knowledge base for the KSP. The knowledge elements (rules) are in the form "IF <condition(s)> THEN <action(s)>." The KSP Stage 1: Initial Data Evaluation knowledge base consists of 201 rules; Stage 2: Comparison of Initial and Redo Processing Runs, 130 rules. Completion of Stage 3: Data Trends has been deferred to direct the use of resources to the operational system requirements definition.

The Front End interfaces with the user in the form of selection and input screens. Required responses are limited to one-keystroke if default values are selected. Page forward and backward options are provided. Data input/viewing screens are provided to allow input and data maintenance. Stage 1 interfaces with the user in the form of a running dialog. It is initiated by loading and initializing the evaluation program after entering the OPS5+ environment. Data not directly downloaded is obtained by user-query; responses are limited to one keystroke. Stage 1 generates a summary report printed on user request. The Stage 2 program, after being loaded and initialized, operates without intervention from the user; both a summary report and a detailed report are created and printed on user request.

SOPS ES PROTOTYPE:

The knowledge base for the SOPS ES prototype was developed using the rule-based expert system language CLIPS. All knowledge elements are represented in the form "IF <condition(s)> THEN <action(s)>." The knowledge base can be logically divided into sets called knowledge islands consisting of rules to diagnose a problem, drive the user interface, and to retrieve data specific to that knowledge island. A knowledge island can be modified or replaced to reflect a procedural change in SOPS without affecting the other knowledge islands; this simplifies the modification process. The prototype consists of four knowledge islands: Run Stopped Early, Data Gap Between Files, Data Coverage, and Data Quality. Each was implemented only to the detail required to realistically demonstrate the feasibility of an operational SOPS ES. The knowledge islands will be expanded for future implementation to include particulars uncovered by this prototype.

The SOPS ES prototype uses many of the standard features for applications running on the Apple Macintosh. The features include the use of multiple windows, pull-down menus, and dialog boxes. Dialog boxes and windows may contain buttons, scroll bars, or space for the analyst to type in additional information called a text field. Whenever possible, the ES will set a default value for the text fields; if the analyst changes the value of a text field, the ES performs a consistency check to prevent the entering of unacceptable values. The primary windows viewed by the analyst are the Transcript, Timeline, and Conclusion windows. The Transcript window maintains a log of the ES session containing all questions asked by the ES, the analyst's responses, all recommendations from the ES, and any analyst-added comments. The Timeline window displays the run in a graphical format with the ES's current focus of attention flagged. The Conclusion window displays the conclusions reached (rules fired) by the ES. All windows can be printed upon completion of the ES session.

CONCLUSIONS:

The prototypes show that the expert systems offer many benefits. They are fast. They are consistent. The expertise of the most experienced staff members is now made available to all. The prototypes can act as training tools when refined to a detailed level. Throughout development, ways in which current procedures could be further automated to increase accessibility to information, to improve processing speed, and to decrease the monotony of repetitious tasks were identified. Also, areas in the expert systems' own operation will be streamlined to make the expert system concept not only workable but operationally practical.

The goal of the expert system prototypes was to define the design and configuration of expert systems in the mission environment. These new operational systems will be larger, more efficient, and more automatic, incorporating the capabilities indicated by, but not present in the prototypes. Both the SIPS and SOPS operational expert system configurations will use the same hardware, the SUN 3/160 workstation, and software, CLIPS with C language interfaces, for consistency and maintainability. Network interfaces will be established to automatically transfer necessary information from the existing SIPS and SOPS mainframes to the workstations for the expert systems' analysis. It is planned that this configuration will be operational by December 1988, in time to support ASTRO-1, the first of several scheduled SLDPF missions in the post-Challenger period.

ACKNOWLEDGEMENTS:

This project could not have been successful without the contributions of the following personnel: Troy Ames (GSFC/Code 522), Ellen Herring (formerly an SLDPF/Code 564 mission manager), Janice Watson, William Dallam, Michael Alvarez, Franz Berlin, Warren Case, Michael Garner, James Pizzola, and Beth Pumphrey (all of Lockheed). The SLDPF personnel also wish to acknowledge Joe Bishop (NASA Headquarters/Code TS) for his continuing support in the enhancement of the SLDPF.

N88-16395

FMEAssist: A KNOWLEDGE-BASED APPROACH TO FAILURE MODES AND EFFECTS ANALYSIS

James R. Carnes
Dannie E. Cutts

Boeing Huntsville AI Center
PO Box 1470, MS JA-65
Huntsville, AL 35807

ABSTRACT

A Failure Modes and Effects Analysis workstation (FMEAssist) has been designed for use during development of the Space Station. It assists engineers in the complex task of tracking failures and their effects on the system. Engineers experience increased productivity through reduced clerical loads, reduced data inconsistency, and significantly reduced analysis time. System developments benefit from a more thorough analysis than was available using previous methods.

The wide variety of design information required to support the FMEA process is modeled by FMEAssist in a network of different discipline and design data views generated by a data base to knowledge base translation tool. System designs are displayed graphically allowing engineers to manipulate information or to induce and record failure on appropriate parts within the network. Propagation of functional effects for each node can be controlled from one or more nodes within the design to any desired level or until special conditions are encountered.

1. INTRODUCTION

Space Station design information is modeled by FMEAssist in a network of nodes (representing components) connected by arcs (representing relationships between the various parts of the design). The wide variety of information required to support the FMEA process is acquired from a view across several heterogeneous discipline and design data base tables and are mapped into a network structure through Foundation, a data base to knowledge base translation tool[4]. System designs are displayed graphically allowing engineers to analyze design information and to induce failure on appropriate parts within the network.

The architecture of this system is built upon a hierarchically decomposed functional model that determines "failure" through abnormal component behavior. This representation permits a more detailed description of "failure modes" beyond those typically pre-defined through either design or system engineering.

PRECEDING PAGE BLANK NOT FILMED

These functional failures are composed of component constraint violations. Propagation of functional effects is controlled from one or more nodes within the design to any desired level or until special conditions, such as a critical failure, are encountered.

2. APPLICATION

Previous work on FMEA automation used a frame-based approach where frames and slots contain pre-defined failure modes and first order effects. While this approach offered powerful descriptive capabilities, first-order analyses were performed manually and entered into the model. Since assertional capabilities were not provided, functional information was not modeled. Failures were propagated through pre-programmed frame connections using messages contained in failure mode slots.[3]

Alternately, a number of fault analysis systems have been developed using assertional models, that is, a rule-based functional approach. While assertional models provide an excellent medium for describing functional behavior, they do not provide the representational convenience of the structural model. Ongoing work has pursued a useful mixture of structural and function models.[1,6]

The FMEAssist approach integrates some of this work on the coupling of structural and assertional components by combining the connective strength of flavors with the expressiveness of logic. A failure mode is defined as the effect a group of abnormal properties has on a component. If the status of the component, due to the its properties, is "failed" or "abnormal", then these properties (and not the fact that the component has malfunctioned) are propagated to connected or neighboring components.[2,5]

Abnormal properties are grouped into failure modes at the component level, but only to promote analysis within the failed component. The knowledge of the mode or even the failure itself is not distributed to the connected or surrounding nodes, but it is the connection and environmental properties that carry a component's fate to neighboring components. That is, a malfunctioning component has no knowledge of how it affects other parts. If the malfunction changes any of its outputs, those values change for the connected component input ports. Input properties for components are the output results from previous inference upon abnormal properties. If these new properties constitute failure or can be classed in a failure mode, they are so recorded, but inference on component properties, normal or otherwise, continues.

"Effects analysis" becomes a deductive process, reasoning from the properties contained in a highly structured part model[5]. The need for pre-programmed propagation responses is replaced with a more refined description of an assembly or

component. This type of descriptive and behavioral information is readily available and can be captured during the design and system engineering process.

3. EXAMPLES

The examples in this section are designed to promote understanding of the descriptive structures and behaviors found within FMEAssist. Figure-1 illustrates how components are defined and described. The defcomponent macro in this example serves to create a component type PUMP-A which inherits characteristics (descriptive and behavioral) from another type called PUMP. The defport and defproperty macros are used to define port/port types and property/property values for a component type.

Some behavioral characteristics of a component type remain constant regardless of how and where instances of it are used in the system. These "generic" behavioral characteristics are shared by all instances of the component type. Examples of generic behavioral descriptions can be found in Figure-2 and Figure-3. Other characteristics however, might change depending on the operating conditions of the instance component. These behavioral characteristics are determined at the particular instance level.

While working with the FMEAssist application on a Foundation Workstation, engineers are able to graphically display networks from various perspective relations, such as subcomponent, failure, and connection spanning tree. Inference mechanisms provide the basis for the application's single, multiple-sequential, and multiple-parallel failure propagation. FMEAssist also provides graphical justification or narrative explanation for any inference produced during the failure propagation. Finally, various reports are generated on the analytical results from each engineering session.

```
; define some generic component descriptions
(defcomponent PUMP-A (generic-type 'PUMP))
(defcomponent VALVE-C (generic-type 'VALVE))

; define input and output for generic component descriptions
(defport PUMP-A ((port-1 'thermal) (port-2 'electrical)))
(defport VALVE-C ((port-1 'thermal) (port-2 'electrical)))

; define properties for port connections
(defproperty THERMAL
  (temperature (nominal high low))
  (pressure (nominal high low))
  (medium (air co2 water)))
```

FIGURE 1: Examples of descriptive definitions

```
; define some rules about components
(defrule PUMP-SHUTDOWN
  (IF [AND [PUMP-PRESSURE-STATUS =some-pump 'LOW]
           [PUMP-TEMPERATURE-STATUS =some-pump 'LOW]]
   THEN (tell [PUMP-STATUS =some-pump 'ABNORMAL])))

(defrule PROPAGATE-PRESSURE-STATUS
  (IF [PUMP-PRESSURE-STATUS =some-pump 'LOW]
   THEN (tell [PUMP-PRESSURE-OUTPUT =some-pump
                                    =some-port 'LOW])
        (tell [(connected-to =some-pump =some-port) 'LOW])))

(defrule PROPAGATE-TEMPERATURE-STATUS
  (IF [PUMP-TEMPERATURE-STATUS =some-pump 'LOW]
   THEN (tell [PUMP-TEMPERATURE-OUTPUT =some-pump
                                       =some-port 'LOW])
        (tell [(connected-to =some-pump =some-port) 'LOW])))

; create some part instances
(make-component PUMP-99 (component-type 'PUMP-A))
(make-component VALVE-46 (component-type 'VALVE-C))
(make-connection '(PUMP-99 VALVE-46)
                 '((port-1 port-1) (port-2 port-2)))
```

FIGURE 2: Examples of specific assertive definitions

```
;Output behavior rule
(defrule LEAKY-THINGS-CONTAMINATE-SURROUNDINGS
  (IF [AND [LEAKS =something]
           [> (PRESSURE =something)
              (PRESSURE (contained-in =something
                                      =something-else))]
   THEN (tell [CONTAMINATES (medium =something)
                            =something-else])))

;Input behavior rule
(defrule CONTAMINATED-ELECTRONIC-COMPONENTS-MIGHT-SHORT
  (IF [AND [CONTAMINATES =medium =area]
           [CONTAINED-IN =component =area]
           [CONDUCTOR =medium]
           [ELECTRICAL-COMPONENT =component]]
   THEN (tell [HAS-SHORTS =component])))
```

FIGURE 3: Examples of generic assertive definitions

4. CONCLUSIONS AND FUTURE DIRECTIONS

Engineers will experience increased productivity through reduced clerical loads, reduced data inconsistency, and significantly reduced analysis time with the added benefit of a more thorough analysis than was available using previous methods. FMEAssist is easy to use, produces FMEA and Critical Items List (CIL) reports, and keeps records of critical failures, as well as sequences of events leading to failures.

In the future, tools like FMEAssist will make it possible for initial failure analyses to be performed early during the system design phases. These tools will be able to identify significant failure modes for single and multiple-point failures. This will free engineers from the tedious task of enumerating the simple single failure mode groupings and to provide an extended capability of correlating complex failure modes with groups of components.

REFERENCES

1. Brachman, R.J., R.E. Fikes, and H.J. Levesque, KRYPTON: A Functional Approach to Knowledge Representation, IEEE Computer, Vol. 16(10), 1983, pp. 67-73.

2. de Kleer, J. and B.C. Williams, Diagnosing Multiple Faults, Artificial Intelligence, Vol. 32, 1987, pp. 97-130.

3. Kamhieh, C.H., D.E. Cutts, and R.B. Purves, Failure Modes and Effects Analysis Automation, Proceedings of the Conference on Artificial Intelligence for Space Applications, November, 1986, pp. 169-176.

4. Purves, R.B., J.R. Carnes, and D.E. Cutts, Foundation: Transforming Data Bases into Knowledge Bases, Proceedings of the Conference on Artificial Intelligence for Space Applications, November, 1987.

5. Reiter, R., A Theory of Diagnosis from First Principles, Artificial Intelligence, Vol. 32, 1987, pp. 57-95.

6. Rowley, S., H. Shrobe, R. Cassels, and W. Hamscher, Joshua(TM): Uniform Access to Heterogeneous Knowledge Structures, AAAI-87, Proceedings of the Sixth National Conference on Artificial Intelligence, July 13-17, 1987, pp. 48-52.

N88-16396

ESSAA: EMBEDDED SYSTEM SAFETY ANALYSIS ASSISTANT

Peter Wallace
Joseph Holzer
Sergio Guarro
Larry Hyatt

ABSTRACT

Automating spacecraft control functions with Software introduces novel failure modes into a system, some unique to an individual application. At the same time, when embedded in an environment that is (at least partially) unreliable, unpredictable, and unspecifiable, software is vulnerable to a whole range of unforeseen eventualities. In contrast to "off-line" applications, however, the consequences of an erroneous software output in an embedded system context are, typically, immediate and potentially disasterous.

We have been building a knowledge-based tool, the Embedded System Safety Analysis Assistant (ESSAA), that can assist in identifying disaster scenarios, in which embedded software could issue hazardous control commands to the surrounding hardware. In short, it attempts to answer the question, "How could this disasterous output ever occur?" Existing software-analysis tools tend to work in the other direction - "What if these were the starting conditions?" or, in the case of debugging - "How did we get from these starting conditions to this (erroneous) output?" For the Safety issue, such analyses generally do not suffice, since it is often the unvisualized combinations of conditions that lead to disaster.

ESSAA, by contrast, is intended to work from outputs to inputs, as a complement to simulation and verification methods. And rather than treating the software in isolation, it examines the context in which the software is to be deployed. Given a specified disasterous outcome, ESSAA works from a qualitative, abstract model of the complete system to infer sets of environmental conditions and/or failures that could cause it. The scenarios can then be examined in depth for plausibility using existing techniques.

At the core of our approach is the Logic Flowgraph Method (LFM) representation language, suitable for capturing the functionality of hardware, software, and physical law within a single unified framework. The languages focuses on the key sysetm parameters and expresses the nature of their functional interconnections. Associated with the LFM

PRECEDING PAGE BLANK NOT FILMED

language is a set of inference rules, which can examine the LFM-expressed system model, understand the causality involved, and deduce what combinations of things would have to be true for a specified disaster-outcome to occur. The system model can, of course, be re-used to examine a series of such outcomes.

As with most Model-Based Reasoning approaches, a non-trivial question is how to construct the initial model. Our research on this aspect is progressing on several fronts, including 1) an Intelligent Model-Building Assistant, incorporating a Knowledge-Base of common satellite components and their functions, the laws of Physics relevant to satellites, and some of the control tasks that may get implemented in satellite software, 2) an analyzer to deduce functional connectivity of key SW parameters directly from the code, and 3) enrichment of the LFM modelling language to capture new patterns of functional connectivity.

N88-16397

Intelligent Process Development of Foam Molding for the Thermal Protection System (TPS) of the Space Shuttle External Tank

S.S. Bharwani*, J.T. Walls, and M.E. Jackson

Martin Marietta Manned Space Systems
MSFC, AL

ABSTRACT

Martin Marietta has designed a knowledge-based system to assist process engineers and technicians in evaluating the processability and moldability of poly-isocyanurate (PIR) formulations for the thermal protection system of the Space Shuttle external tank (ET). The Reaction Injection Molding - Process Development Advisor (RIM-PDA) is a "coupled system" which takes advantage of both symbolic and numeric processing techniques. Process knowledge, consisting of heuristic knowledge acquired from domain experts, such as case histories of chemical formulations and their moldability in test mold configurations, and the knowledge of causal relationships derived from the empirical data will aid the process engineer in 1) identifying a startup set of mold schedules, and 2) refining the mold schedules to remedy specific process problems diagnosed by the system.

INTRODUCTION

Research in expert systems and their application to machine and process control and diagnostics has received much attention in recent years. However, relatively little has been done to provide the application experts with any intelligent, generic tools for organizing and representing their discoveries and knowledge of novel processes. In particular, little attention has been given to exploratory processes whose feasibility must be confirmed by significant experimentation.

A wide array of process management tools is available for modeling and design of both discrete and continuous processes. At one end of the spectrum are commercially available algorithms for numerical simulation of chemical and fluid flow processes. These algorithms are useful only for modeling processes which are relatively well understood and hence amenable to rigorous mathematical treatment. At the other end are tools for symbolic representation that are well suited for describing process domains which are too complex to be modeled numerically, yet represent a significant body of experiential process knowledge. The knowledge acquired from those in the field (also referred to as domain experts) is usually available in the form of process heuristics or "rules of thumb" which have been developed and refined through a combination of intuition and trial and error over an extended period. Such knowledge, represented in the form of condition-action pairs, can be called upon by a novice to help with local process problems.

A major drawback of the symbolic reasoning systems (also called expert systems) developed thus far is their lack of mechanisms for guiding the process engineer in exploring the causal relationships between process parameters and their effects on process performance. That is, they provide no capabilities for generic process development, such as deriving and representing parameter interactions between successive stages of a multi-stage process, which is crucial for model development and/or refinement. This paper highlights the characteristics of intelligent information processing technology that would most appropriately address the important issues in process development. Our discussion is based on our experience with the design of a process development advisor to assist process engineers in developing a complex foam Reaction Injection Molding (RIM) process. Both the advisor and the process are currently under development as a joint effort between Martin Marietta Corporation and NASA at NASA's Productivity Enhancement Center.

In the following, process development tasks common to a large class of process and manufacturing domains are identified, followed by a description of the RIM process and the design of a system to address the development tasks that are relevant to the RIM domain.

*Affiliated with Martin Marietta Laboratories, Artificial Intelligence Group, 1450 S. Rolling Road, Baltimore, MD 21227.

PROCESS DEVELOPMENT TASK: A PERSPECTIVE

How generic can a process development task be? Although the problem domain of immediate interest to us is the manufacture of poly-isocyanurate (PIR) moldings with RIM technology, our principal concern is developing tools and techniques that are useful for generic process development tasks. Hence, the system design we propose for this purpose is equally applicable to development activities in composites, metals, and semiconductor fabrication.

In a polymer molding operation, a process is considered developed if the molded parts consistently meet the functional requirements of the intended application. For our domain, these requirements would be in the form of mechanical and physical properties and thermal protection criteria for the external tank (ET), as stated in the design specifications. In this case, process development also includes process problem diagnosis and solution, as well as process optimization for greater consistency, higher quality, and improved efficiency.

Expert process engineers are very successful at diagnosing process problems because they can envision a process as a continuum as well as a construct of discrete major functional components. This duality allows them to track major process events and relate them to the relevant interactions between the parameters of the components. The interactions of specific interest are later characterized by modeling the parameter relationships empirically through carefully designed sets of experiments. The experience so gained is then used for qualitative analysis of the relative importance of the control parameters and quantitative manipulations of the functional relationships to produce a desired improvement in the properties and thus quality of the molded parts. The experience described above is believed to be quite generic and common to all process development domains.

To perform these representation and reasoning tasks, an engineer needs a system that will allow construction of a causal model for simulation of the process behavior at a high level of abstraction. This model can be refined as more information is obtained about the precise quantitative relationships from the experimental data, and knowledge of these relationships can then be reasoned with to determine the tolerance windows on all material and equipment controls for optimum performance. A schematic describing the three-layered process development scheme applicable to the RIM process is shown in Figure 1. The schemes for knowledge acquisition, knowledge representation, and reasoning are distinctly different at each layer.

A system that embodies a hybrid representation comprising the three layers discussed above should be able to answer a wide variety of process-related questions, ranging from very general to very specific. For example, a process engineer may want to understand the role of a catalyst in PIR polymer formulation. Catalysts come in many varieties, ranging from those that accelerate reaction rates to those that promote selective precipitation. A process engineer should be able to ask the following types of questions of such a system:

Causal Level: How does the catalyst concentration impact the flowability of PIR formulations?

Empirical Level: If the catalyst concentration is increased by 10%, how will the gelation time of the polymer change?

Heuristic Level: Do catalysts usually affect flowability?

Here, the concerns addressed are primarily those of knowledge acquisition, knowledge representation, and reasoning at the second layer. Issues pertaining to causal modeling, causal simulation, and integration of all three layers for efficient reasoning will be a subject of future study.

Systematic and efficient acquisition of process knowledge derives from the application of appropriate experimental methods and analysis tools to identify the process-critical variables and determine their impact on the processability of the polymeric material. Such tools must couple numeric processing algorithms for analysis with symbolic processing schemas for reasoning and interpretation of the analytical results [Kitzmiller]. Hence, our approach is to design a coupled system with the following features:

1. Computationally efficient numeric algorithms for statistics, analysis, and graphics, which exist as separate modules and are callable by the system as needed

2. Symbolic processes to guide the user in identifying the right selection of numeric routines to extract the empirical relationships and in interpreting the results.

Additionally, the knowledge acquired has to be represented in a manner natural for reasoning within the process development environment, i.e., for dealing with process problem diagnosis and corrective actions. Within the RIM-PIR domain, the reasoning tasks rely heavily on the explicit representation of parameter effects on the characteristic properties of the process. A detailed description of the RIM process and its requirements follows.

The RIM Process

Reaction injection molding is a process in which polymeric products are formed from highly reactive chemicals in high-pressure impingement mixing machines. Obtaining the desired functional properties of the molded part requires control of a wide variety of process variables associated with four major process areas:

1. The chemical systems which produce the urethane and PIR polymers
2. The RIM machine itself, including at least two metering pumps, a self-cleaning impingement mixer, a set of temperature-controlled conditioning tanks, and the piping, hoses, filters, and controls required for operation
3. The mold support or mold-handling system
4. The mold temperature-control system.

Activities in these four areas are interdependent. The chemical systems must form polymers with the physical properties required for the part being molded; the metering system must meter accurately and mix thoroughly; the mold support must position the mold to best facilitate the expansion process; and the mold itself must be designed to facilitate the flow of the mixed reactants during filling and expansion, and to control the temperature of the chemical reaction within a relatively narrow range. The fluid expansion operation inside a mold is particularly complex because it involves multiphase flow of reactive polymers that undergo rapid state changes.

Figure 2 shows a process flow diagram for a RIM process. Since the process is still in its prototype phase, all pre- and post-process operations are performed either manually or by offline dedicated systems. Engineering evaluation and quality inspection tests are performed at the end of a complex part molding process to provide the process engineers with early information on the status of the manufacturing process. Although the tests could indicate a variety of process problems at several levels of complexity, the level at which a process engineer may choose to diagnose a problem usually depends on his experience with the process and his training in modeling and analyzing the process. For example, a novice process engineer may try to resolve inconsistencies in part quality by making ad hoc changes in mold setup or RIM machine parameters because he does not recognize the true source of the problem. Diagnosing the actual cause of substandard part moldings requires careful analysis of parameter interactions at multiple stages of the molding operation.

The multistage diagram of the RIM process shown in Figure 3 is a simplified representation of the typical parameters that affect the process. At each stage, there are generally several options for modifying the process behavior. One such option is to change the reactivity of the chemical formulation, which, in the case of PIR formulations, is known to have a major impact on the processability of the material. For example, improved flowability could be achieved by changing the concentrations of the catalysts, or the blowing agents or some combination of the two. On the other hand, for duplicating the processing capabilities of a prototype operation on a delivery system, it is more appropriate to scale the reactivity up or down (within limits) by changing the impingement pressure at the RIM machine stage or the temperature of the mold at the mold setup stage. The best or the optimal of the available alternatives is generally not obvious and may require a thorough and careful analysis of multivariate effects on the response surface of the characteristic properties.

In addition to understanding the process behavior modifications produced by intra-state parameter variations, the process engineer also needs to study their effect on the process over the succeeding stages. For example, the complexity of a part to be RIM-molded is usually determined by the overall size of the part and the maximum cumulative resistance to flow through the mold. If the part is too big and/or the flow resistance is too high, then the part may have to be made from two or more simpler molds. This decision, in turn, will govern the appropriate settings of the mold setup parameters at a succeeding stage. The process variables that may affect the molding quality are:

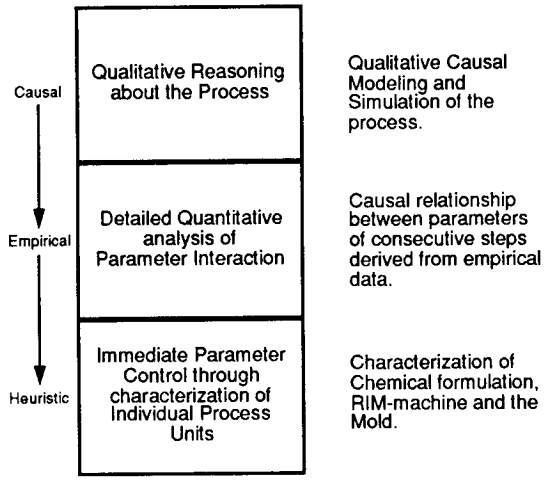


Figure 1. Multilayered representation of process knowledge

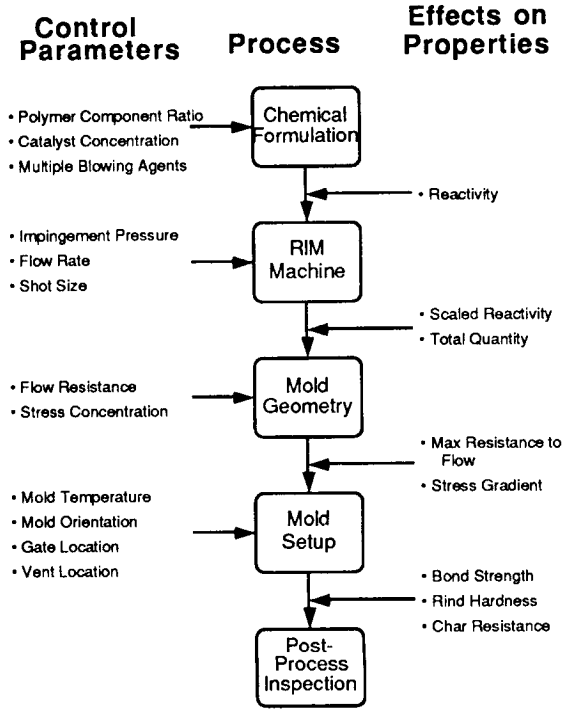


Figure 3. Functional decomposition of the RIM process showing typical control parameters and their effect on the process

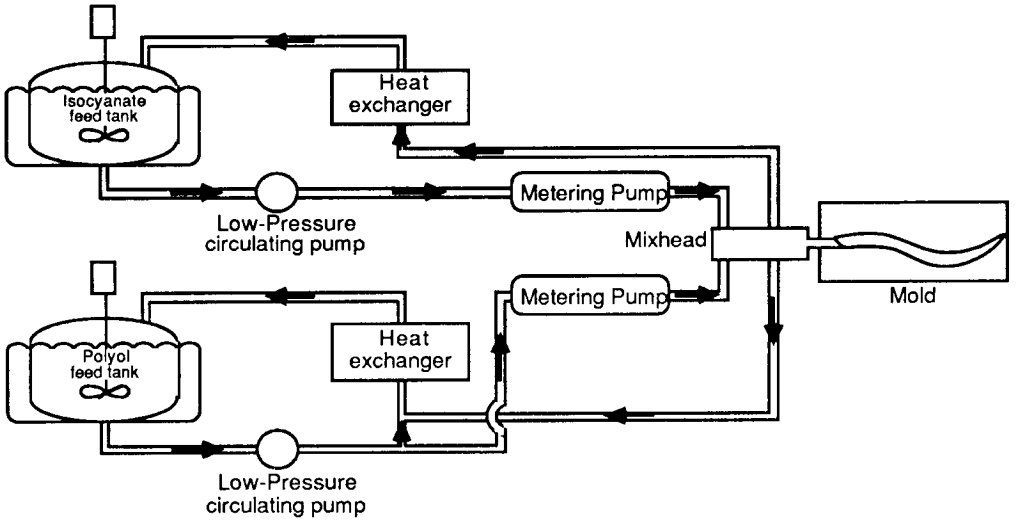


Figure 2. The RIM Process

- •Polymer component ratio
- •Catalyst concentration
- •Mold temperature
- •Mold orientation
- •Impingement pressure
- •Gate location
- •Vent locations
- •Shot size.

Some typical problems resulting from lack of control of one or more of these variables are:

1. Incomplete fill - Failure of the expanding foam to reach all parts of the mold cavity
2. Voids - Presence of large blowholes or pinholes in the molded part
3. Warp - Bent part, deviating from flat condition or dimensions out of tolerance
4. Flash - A film of excess material outside the mold cavity
5. Density Reject - Part density not within specification
6. Hardness Reject - Rind hardness not within specification
7. Strength Reject - Tensile strength not within specification.

Some of these problems belong to a class referred to as gross defects; i.e., these are quality defects in the molded part whose remedy requires gross adjustments in one or more control parameters. Such problems are best handled by resorting to the use of heuristics. For example, the presence of large voids in a molded part is usually attributed to lack of appropriate vent holes in the mold to bleed off air entrapped during the expansion process. Hence, the system can capture this knowledge through a rule as follows:

IF there is evidence of LARGE VOIDS in the part
AND there is CERTAINTY > 0.9 of VOIDS due to ENTRAPPED AIR
THEN provide VENT HOLES in the mold where the voids are found

Other problems are more subtle and require careful analysis to establish the ideal window of process parameter values. All problems regarding properties that are out of specification belong to this class.

KNOWLEDGE BASE

The knowledge of the system consists of information on chemical formulations and part geometries that have been used in the past and have relevance to the ongoing process development. Each chemical formulation exhibits a characteristic behavior which can be expressed in terms of its state changes over time (for reactive chemical systems, such behavior is also referred to as its reactivity). Qualitative descriptions of observable states, such as "cream," "gel," "string," and "tack-free," then generate a quantity space [Forbus] for a continuous process over a temporal dimension. The case history for each chemical formulation is thus represented by such a quantity space because it contains critical phase transformations of the fluid which provide important information regarding the processability of the chemical formulation.

The case history for part geometry is grouped into classes of parts requiring similar startup process parameters. Currently, there are three distinct part classes: spherical, rectangular, or prismoidal. Within each class, there are further fine-grained distinctions or subclasses. Assignment of a part to a particular subclass is determined by the number and type of its primitive flow obstruction features. Such a classification scheme minimizes the process development time significantly by encompassing mold schedules already developed for parts in the subclasses of the case history.

Process data acquired from the RIM machine and the process-monitoring instruments are represented in the form of records. Query language facilities of a relational database manager are used to generate data summaries, which are then used by the analysis module as needed to derive individual parameter effects on the processability of the material. All data within the analysis module are represented in the form of arrays

because of their efficient representation of numeric processing algorithms as well as graphics algorithms to communicate the parameter effects graphically to the user.

The knowledge base also contains data analysis and data interpretation schemas. These determine when to call certain analysis functions to determine statistical distributions and variances, and provide methods for evaluating the effects of individual parameters on the behavior of the process, respectively.

KNOWLEDGE REPRESENTATION AND CONTROL

The process diagnostic knowledge of the system consists of heuristics represented in the form of production rules [Davis] and data objects. The rules explicitly state the relationship between part defects and the actions that could remedy such defects, while the object framework is a representation of data on gross visual defects and the test results on the properties of the molded part. Generally, objects belong to one or more classes and have properties with value slots, as shown in Figure 4.

The parameter effects are represented in the form of intensities, distributions, and explicit mathematical formulations [Blum] and may be accessed from the knowledge base and/or database by the inference engine. All data analysis and interpretation schemas have a Shank's "script" [Shank] flavor. The schemas are represented as frames [Stefik], enabling them to be instantiated with relevant parameter values.

The system inferencing scheme is mainly data-driven/forward chaining. The initial set of findings establishes the focus of attention, which then controls the evaluation of only those hypotheses which are relevant to the current line of reasoning. Hypotheses that share common data objects are grouped into clusters called "knowledge islands." The presence of such clustering significantly improves the speed and efficiency of inference. Hypotheses within a knowledge island are organized hierarchically and are explicitly categorized to control the order of their evaluation. Such a categorization is necessary to ensure a consistent dialogue with the user-- a dialogue that is logically relevant to the problem under consideration.

Sample Session with RIM-PDA

The control within the system is illustrated in Figure 5. The user usually needs advice from the system if a new part with a different geometric configuration is being considered for molding or a new chemical formulation with a different reactivity characteristic is being evaluated for use as a molding material. The user initiates a dialogue by generating an appropriate event to inform the system that a new set of mold schedules is required. The system searches through the case histories to find the mold schedules of a case which closely resembles the current one. The startup set of schedules is then used to execute the first iteration of the molding operation. In-process sensory data collected during the molding operation, together with the post-process test results, are then analyzed to inform the user of the health of the process. If the user informs the system about any problems in the quality of the molded part, the system enters the refinement phase [Bharwani], diagnosing the source of the problem and recommending a new set of mold schedules each time a problem is reported.

CONCLUSIONS

This paper has addressed process development as a generic engineering task and identified its demands on advanced information technology. A coupled system is proposed to take advantage of conventional algorithmic approaches as well as state-of-the-art artificial intelligence methodologies to cope with the identified tasks. Several issues identified earlier are currently under investigation. Mechanisms for switching between layers of reasoning based on the type of question asked of the system have yet to be developed. Additionally, the concept of a quantity space for causal simulation needs further refinement.

ACKNOWLEDGEMENT

This work was funded by NASA NAS8-30300 under Technical Directives 1.6.2.1-249-R2 and 1.6.2.1-673. The authors would like to thank the NASA management team for making these activities possible. They would also like to thank Steve Barash, John Thorp, Judith Marcus, and Rob Cochran for help on several occasions, and John Lewis for his encouragement in pursuing the ideas expressed in the paper.

ORIGINAL PAGE IS
OF POOR QUALITY

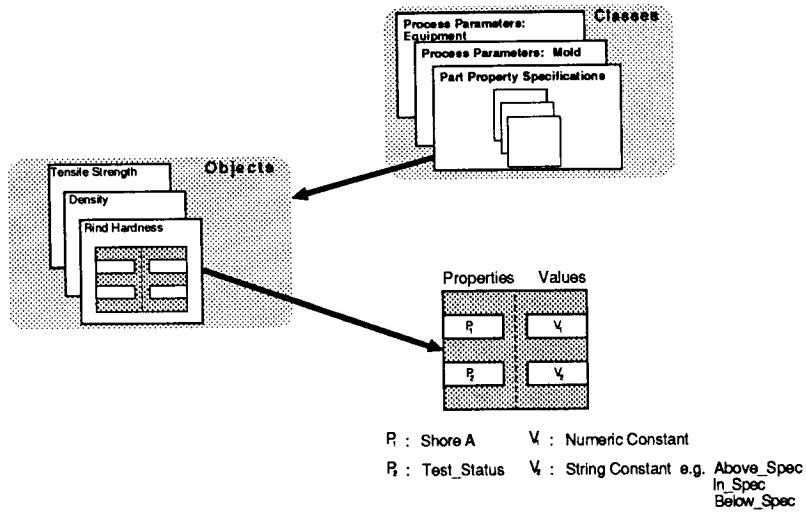


Figure 4. Class/Object Data Structure

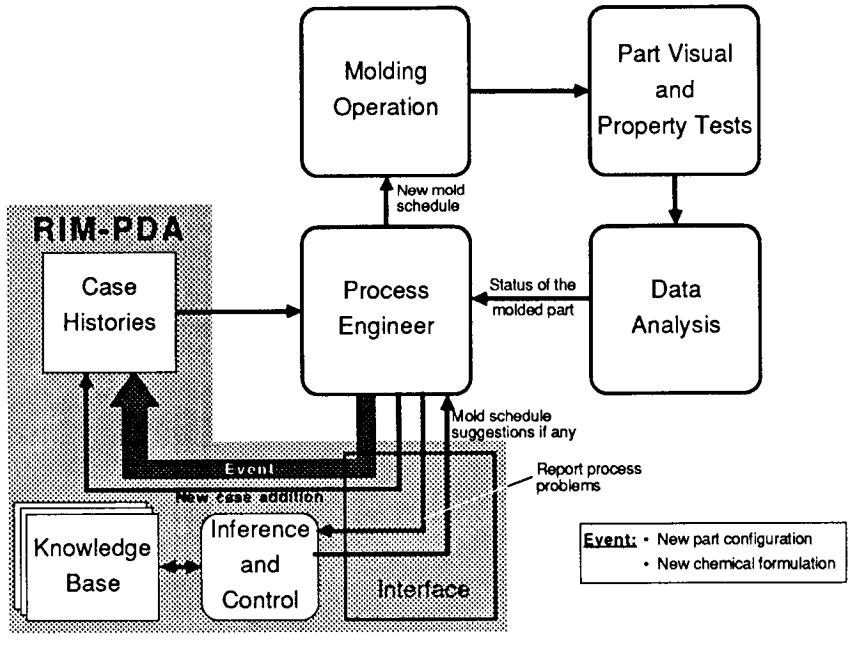


Figure 5. RIM-PDA Control Scheme

REFERENCES

1. [Bharwani] Bharwani, S.S. et al. (1986). "Refinement of Environmental Depth Maps Over Mutiple Frames," in *Proc. of IEEE Workshop on Motion: Representation and Analysis*, Charleston, S.C., May 1986.

2. [Blum] Blum, R.L. (1983). "Representation of Empirically Derived Causal Relationships," in *Proc. of Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, 8 - 12 Aug. 1983.

3. [Davis] Davis, R. et al. (1975). "Production Rules as a Representation for a Knowledge-Based Consultation Program," Stanford University, STAN-CS-75-519.

4. [Forbus] Forbus, K.D. (1984). "Qualitative Process Theory," in *Qualitative Reasoning about Physical Systems*, D.G. Bobrow, ed. MIT Press, Cambridge, MA, pp. 85-168.

5. [Kitzmiller] Kitzmiller, C.T. and Kowalik, J.S., (1986). "Symbolic and Numerical Computing in Knowledge-Based Systems," in *Coupling Symbolic and Numerical Computing in Expert Systems*, Amsterdam, The Netherlands.

6. [Shank] Shank, R. and Abelson, R. (1977). *Scripts, Plans, Goals and Understanding-An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum Associates.

7. [Stefik] Stefik, M.J. (1979). "An Examination of Frame-Structured Representation System," in *Proc. of International Joint Conference on Artificial Intelligence*, Tokyo, Japan, Aug. 1979, pp. 845 - 852.

SPACE SHUTTLE MAIN ENGINE ANOMALY DATA AND INDUCTIVE KNOWLEDGE BASED SYSTEMS: AUTOMATED CORPORATE EXPERTISE

N88-16398

Kenneth L. Modesitt
Rocketdyne Division, M/S AC06
Rockwell Corporation
6633 Canoga Avenue
Canoga Park, CA 91303
(818) 710-2077

Abstract

Since 1984, a low-level effort has been underway at Rocketdyne, manufacturer of the Space Shuttle Main Engine (SSME), to automate much of the analysis procedure conducted after test firings. Previously published articles in corporate publications, international conference proceedings, and the international trade press have contained the context of and justification for this effort [4, 8, 9], and technical issues regarding the integration of large data bases with the run-time component of the knowledge-base system [2]. In this paper, progress is reported in building the full system, known as SCOTTY, after a noted 23rd century rocket propulsion expert.

The progress is on two fronts. The first is an organizational or philosophical one: the automated analysis of SSME tests is a complex process, but only part of it typically involves an expert. That is, this expert knowledge-based system is called only when required; there are many more mundane tasks which may interface to an expert, but do not require heuristic technical expertise. Don't make your technical expert also serve as the program manager!

The second front of progress is a technical one. Since the very inception of the program, it has been strongly believed that the intrinsic nature of SSME test analysis and character of inductive-based expert system building tools (ESBTs) represent an excellent match of problem and tool. It was, in fact, the driving consideration for the 1984 recommendation given to management, along with the critical feature of being software-compatible with existing systems, i.e., the tool must be able to generate Fortran code. The intuition has been justified by the relative ease with which a significant source of corporate diagnostic and analytical expertise has been transformed to examples and thence to effective production rules. The source of this expertise is in completed SSME anomaly forms, one for each of the 1400+ tests conducted since 1975. The latter transformation from examples to production rules is accomplished automatically with a powerful inductive ESBT, ExTran 7 from Intelligent Terminals Ltd. [1], running on a Concurrent Computer Corporation 3260 super-minicomputer. The engineering staff responsible for building (and eventually maintaining) SCOTTY has consistently used examples as input -- a single rule has yet to be written. The knowledge-acquisition "bottleneck" is thus much wider than for most previously-reported expert systems.

The moderate expansion of the former low-level efforts in constructing an automated test analysis procedure for the SSME will be discussed in this paper. The topics will cover qualitative and quantitative details of the above organizational and technical issues, as well as various possible extensions being considered. These include:

- the integration of a large-scale relational data base system [3] and example generation at the knowledge acquisition component of ExTran 7,
- a graphics interface for experts and end-user engineers,
- increased efficiency from exploiting concurrency in problem and machine,
- potential extension of a tuned and limited subset of the system to flight engines,
- application of the system for training purposes of many newly-hired engineers,
- incorporation of design and monitoring tasks into an automated system,
- technology transfer to other engines and Rocketdyne programs,
- an anomaly description language, and
- the essential qualities of good software engineering practices for building expert knowledge-based systems.

Introduction

Every time a Space Shuttle Main Engine (SSME) is test fired, hundreds of measurements are taken from a wide variety of sensors. Many more values are also calculated. All of these data values, when combined with previous performance of the engine and its components, are used by the engineering staff at Rocketdyne to determine the future tests. These outcomes can vary from all requirements being met, to a few minor events, to a rare significant event. As the SSME is the world's most complex reusable liquid-fuel (oxygen and hydrogen) rocket engine, Rocketdyne and NASA, the customer, consistently insist that a thorough investigation of each test firing be performed by our most highly-trained staff. This emphasis on quality is heightened even further as a result of the Challenger tragedy, ensuring that the next scheduled shuttle flight, Discovery in June of 1988, will be the safest that is humanly possible. The recent increase in the SSME testing schedule, to about twelve per month, is witness of this concern.

To continue its virtually perfect record of supporting shuttle flights, Rocketdyne is always looking for ways, both technical and organizational, to improve the quality of our product while working within customer guidelines. One of the major methods involves making the most accurate diagnosis, analysis, and recommendation possible for the the next engine test or shuttle flight. To perform this task, reliance has been on maximal use of sophisticated tools and the expertise of an engineering staff. This staff has accumulated experience dating back to 1975 and covering 1400+ SSME firings, plus numerous other ones from the Apollo F-1 and J-2 engines to those on the Atlas.

In addition to gaining incalculable experience over the years, the engineering staff has also been gaining an increase in another quality -- age. Like most other aerospace companies, Rocketdyne has a significant gap between staff with 20-30 years of experience and those with 5-10 years. Although the young

engineers are bright and motivated, they are keenly aware they do not possess the wealth of background of our older senior staff. These staff engineers are approaching or at retirement age, but their replacements have considerably less rocket engine experience. Hence, Rocketdyne is confronted with a significant dilemma: how to improve the quality of SSME test analysis in the face of diminishing senior staff. Several options to solve this dilemma were discussed in [4]. It was decided to use a combination of staff, results from previous SSME tests, and automated tools to address the problem and begin to build a prototype for automated corporate expertise.

Rocketdyne is far from alone in being confronted with the above problems. Indeed, the corporation has ample "company" in deciding to use a type of automated tool know as expert systems, part of the artificial intelligence technology. In fact, there is considerable concern that, once again, this field is in danger of being "over-hyped" [6]. And the company is certainly not the first to decide to concentrate initially on a diagnosis type application, a type currently of considerable importance in industry despite being "old-hat" to the AI research community. So what is unique about SCOTTY, our automated system? There are two unusual aspects.

One such aspect is the incorporation of SCOTTY as "another", albeit advanced, software tool which must:

live in a distributed corporate environment,
talk to large data bases,
be maintained by existing engineering staff,
run with color graphics terminals,
execute on standard computers,
be amenable to parallel processing hardware, and
meet corporate-wide software engineering development and quality guidelines.

The other unusual aspect is a technical one which increases the ease with which SCOTTY can be constructed. By use of a type of ESBT known as inductive or example-based, the historical expertise now reposing in the anomaly sheets for the hundreds of SSME tests can be transformed into examples, and thence automatically into production rules. These rules will, in turn, drive SCOTTY during normal day-to-day operation in future years.

SCOTTY

History

In 1984, the author was hired by Rocketdyne to assist in the construction of an automated tool for SSME test analysis. Within two months, a proof-of-concept model for a High Pressure Oxidizer Turbo Pump (HPOTP) had been built. This involved recommendation of an inductive ESBT, Expert Ease by Intelligent Terminals, Ltd (ITL) in Glasgow, Scotland, and the first such PC-based ESBT commercially available. The tool was purchased and used, after minimal training time, by a mechanical engineer, to diagnose HPOTP anomalies, by specifying 42 examples and nine attributes. A 48 rule subsystem was automatically generated by Expert Ease. No rules were required of the engineer. This prototype and the problem context, rationale, and solution were described in an early paper [4]. A desirable tentative system configuration is shown in Figure 1.

ORIGINAL PAGE IS
OF POOR QUALITY

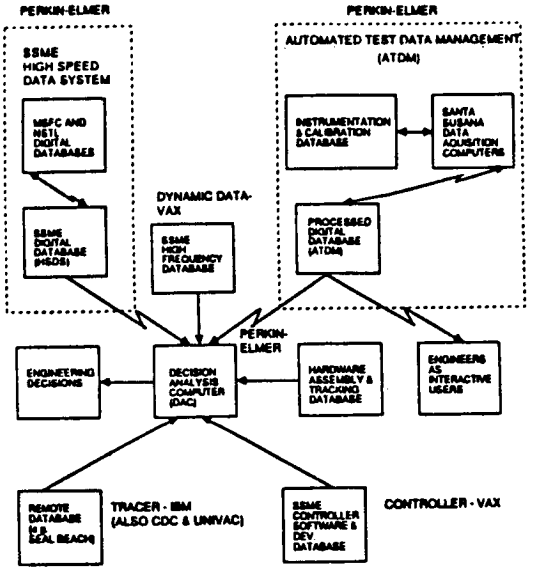


Figure 1. SCOTTY:
Final System Configuration

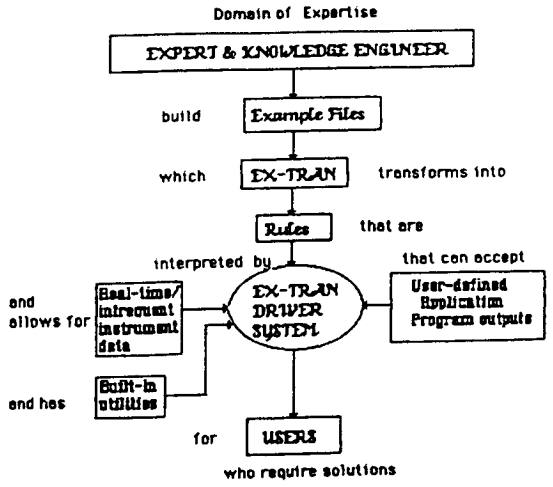


Figure 2. Building Expert Systems
with ExTran

During 1985 and 1986, the system (now named SCOTTY) underwent several extensions. From a tool viewpoint, a more powerful ESBT became available. ExTran 7, an industrial strength Fortran-based inductive ESBT from ITL which runs on a wide variety of machines from PCs to workstations to super-minis to mainframes, was recommended. A process for using ExTran is given in Figure 2. ITL ported the product to the available Concurrent Computer Corporation 3260 super-mini at minimal cost. The HPOTP examples were immediately transported to ExTran and the resulting module was now a true, albeit simple, knowledge base system (KBS) utilizing "Why", "How", and "What if" type questions, history files, external interfaces, and all the other features usually associated with a KBS.

Conceptually, SCOTTY was extended in several directions during this same time period. It was demonstrated that multiple problems could be run concurrently on the multiple processor Concurrent 3260. Graphics routines (PLOT-10 and GKS libraries) were tied to ExTran with a minimum interface. In-house statistical routines were easily linked to SCOTTY. Small Fortran routines were written to access SSME test files and output attribute values for input to SCOTTY sub-problems. Additional SSME component modules were specified. Figure 3 contains the early version of a structure chart. A major extension was the run-time interface between ExTran and the large data base managment system DMS/32 supplied by Concurrent (then known as Perkin-Elmer). These are all described extensively in a paper presented in 1986 [2].

ORIGINAL PAGE IS
OF POOR QUALITY

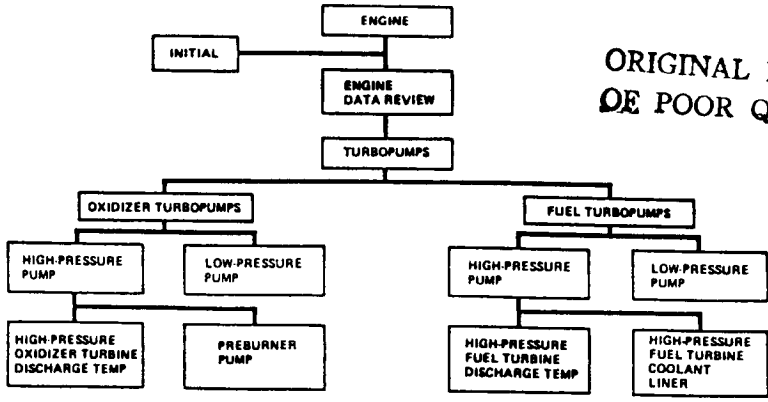


Figure 3. SCOTTY Structure Chart, Version .84

Current Status

SCOTTY now exists at a stage between a research prototype and a production model, using the taxonomy of Waterman [6]. It is not being used in production now -- additional resources would permit this to occur sooner. SCOTTY consists of far more than "just" an expert system, as is clearly shown in figure 4, but rather is one component in a fairly extensive software system. This reflects the strong belief that viable expert systems are most likely to succeed in a hybrid and integrated environment, where they must communicate easily with other standard existing and future sub-systems.

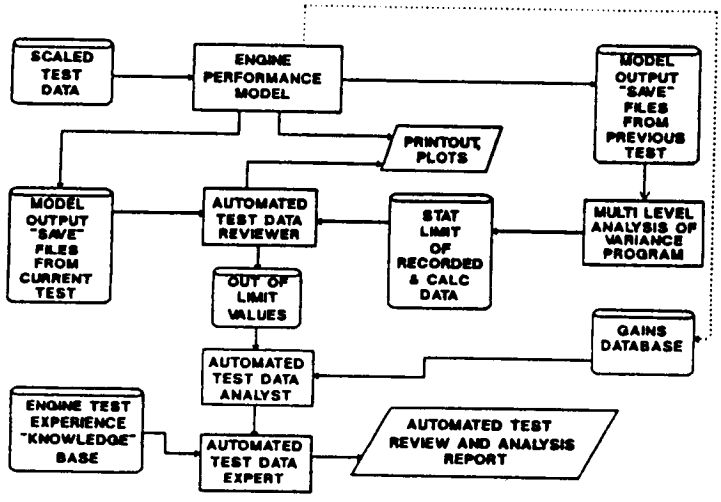


Figure 4. Context of SCOTTY (Automated Test Data Expert)

An updated structure chart reflects the understanding that SSME test analysis involves several levels of expertise, from relatively mechanical but comprehensive data review to component and system level diagnostic and analytical experts. See Figure 5. For many routine tests, the expert system is not required.

This is no stranger than the human equivalent case of calling on resources only when needed. The program manager decides when he/she must draw on the rare and expensive human expert. Do not make the mistake of having your senior technical specialist trying to "double" as program manager. These two entities have different types of expertise.

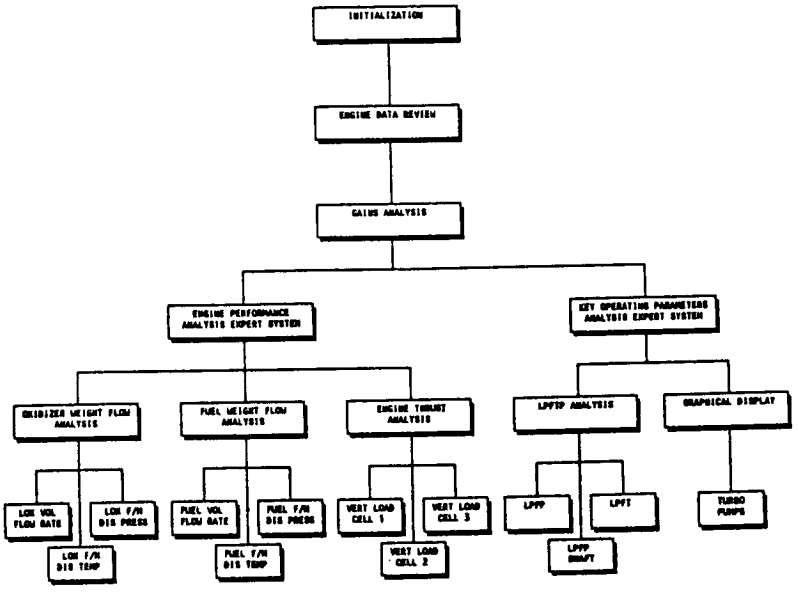


Figure 5. SCOTTY Structure Chart, Version .9X

SCOTTY, as of mid 1987, consists of 18 ExTran modules comprising 3200 lines of code (LOC) in Fortran. Supporting code required 5000 LOC. The ExTran generated code was derived automatically from approximately 260 examples. No rules have been written by hand to date -- all 700 rules were induced.

Extensions in Progress

Development is continuing on a number of fronts for SCOTTY. Three are highlighted here: graphics, anomaly data, and a new extension to ExTran.

As SCOTTY matures, the level of effort devoted to its development has increased. The first efforts were done solely on Rocketdyne internal R&D funds. Currently, funding efforts have been underway for some time with a customer for making the system a production one more quickly. Recently hired engineers and computer scientists have been active in extending the structure, leaving the knowledge content to acknowledged experts. A SSME instrumentation chart, now taped to the walls of hundreds of Rocketdyne engineering offices, is being converted to a dynamic color computer graphics form. See Figure 6 for a sample, purposefully simple, display of a Low Pressure Fuel TurboPump (LPFTP). This graphics subsystem will have capabilities to zoom, highlight problem areas (according to actual test data measurements), and depict flow. This is <u>not</u> CAD/CAM, although there are a few common themes, nor is it extensive CFD modeling

ORIGINAL PAGE IS
OF POOR QUALITY

of the National Aerospace Plane (NASP) using multi-million dollar CRAY 2s. It is a practical and feasible use of moderate color resolution on the readily available super-mini and terminals. Engineers on the floor, as would be expected, are very pleased to see in graphical form what they have hitherto had to dig out of static tables and plots.

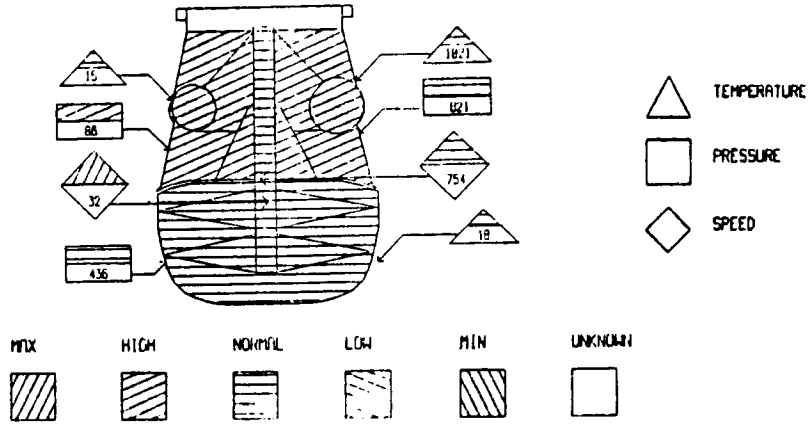


Figure 6. Simple Graphic Display for a LPFTP

Anomaly data is a key to the success of SCOTTY. It provides a starting point for converting much of the SSME testing expertise repository into machine readable form. Serious efforts are underway to use this source to augment the experience now encapsulated in the heads of senior engineering staff. Each anomaly sheet consists of three major fields: problem (symptoms), analysis (causes), and recommendations. See Figure 7 for a (dummy) sample. Zero or more anomalies are recorded for each test, usually very minor ones. By carefully reviewing each anomaly and any back-up plots/tables, it is possible to convert each one into an example format consisting of a set of attribute-values and decisions. Anomalies for the first several tests tried are converted rather slowly, as new attributes are frequently added. However, as experience in the conversion process is gained, and the rate of growth of new attributes slows, the rate of the anomaly to example format conversion increases significantly.

SSME ANOMALY SUMMARY

ENGINE

TEST

ANOMALY	ANALYSIS	ACTION TAKEN FOR NEXT TEST	UCR REQUIRED
FPB PC (PIDS 58, 158 AND 418) - FACILITY MEASUREMENT DEVIATES FROM CADS DURING MAINSTAGE AND OFFSET REMAINS AFTER C/O	• FACILITY AND CADS MEASUREMENTS SHOW NO OFFSET AT E/S • CADS MEASUREMENT IS NORMAL • OFFSET INCREASES DURING TEST • OFFSET PRESENT AFTER C/O • SENSOR CHECKOUT OK • PID IS INFORMATION ONLY • OK TO TEST	• CONTINUE TO MONITOR	NO

Figure 7. Sample Anomaly Data

The third extension underway is the intention to use a new feature of ExTran which is the result of a joint project between ITL and Concurrent with roots in the earlier work at Rocketdyne [2]. This feature extends the interface between ExTran and a data base system to the knowledge acquisition component of the former, as well as the run-time interface discussed in [2]. See Figures 8, 9, and 10. The effort of this joint project is known as Reliance-Expert, and is scheduled to undergo beta test at Rocketdyne. It would permit the anomaly data to be transformed to records in a relational DBMS. This would make this valuable data available for a wide variety of uses. One of the uses would be to serve as an "expert" for historical knowledge of SCOTTY, as it can now be transformed automatically into examples and then to rules. So, once again, the knowledge acquisition bottleneck becomes less and less of an issue, as it will be possible to go directly from anomaly records in a DBMS to production rules in an expert system.

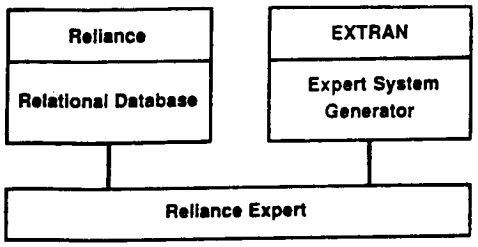


Figure 8. Reliance Expert Structure

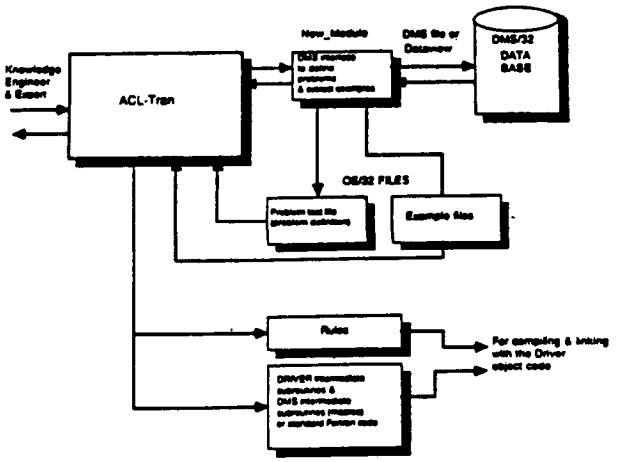
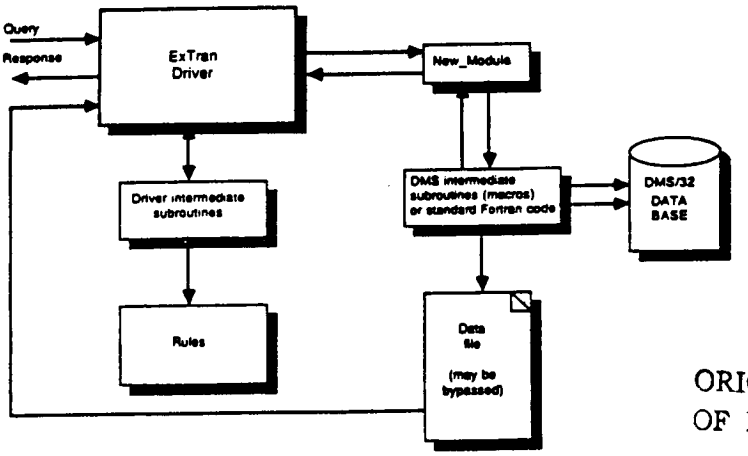


Figure 9. Reliance Expert Development Phase



ORIGINAL PAGE IS OF POOR QUALITY

Figure 10. Reliance Expert Run-time Phase

Future Development Issues

Further in the future are several concerns. There is an interest in each as a potential contributor to improving the quality of SSME test analysis. Obviously, Rocketdyne is keenly concerned also about technology transfer to other types of engines, in addition to the SSME. The company is deeply committed to the Advanced Launch System (ALS), Space Station power, National Aerpspace Plane (NASP), Orbital Transfer Vehicles (OTV), and other propulsion and energy systems.

These further-reaching concerns are concentrated both in application and technical areas. On the application side, Rocketdyne would like to investigate the potential of extending SCOTTY to handle a limited subset of the measurement data for flight engines. The incorporation of monitoring and design tasks is also of high interest. An obvious application is to enlarge the context of SCOTTY to include new hire training on SSME test analysis.

On the tool side, the issue of dealing with uncertain and/or noisy example data is significant. Real engineering problems involve uncertain and incomplete information. Indeed, a noted nuclear engineer, Dr. Billy Koen at the University of Texas in Austin, has gone so far as to define the engineering method as "the use of heuristics to cause the best change in a poorly understood or uncertain situation within the available resources" [5]. This feature exists on several commercial tools, but only in a pre-release form for ExTran, as of this date. The possibility of using abductive reasoning for diagnosis also appears to hold some promise [7].

Conclusions

Since 1984, a low-level effort has been underway at Rocketdyne, manufacturer of the Space Shuttle Main Engine (SSME), to automate much of the analysis procedure conducted after test firings. In this paper, we reported on progress in building the full system, known as SCOTTY, after a noted 23rd century rocket propulsion expert.

The progress is on two fronts. The first is an organizational one: the automated analysis of SSME tests is a complex process, but only part of it typically involves an expert. Don't make your technical expert also serve as the program manager!

The second front of progress is a technical one. Since the very inception of the program, it has been strongly believed that the intrinsic nature of SSME test analysis and character of inductive-based ESBTs represent an excellent match of problem and tool. The intuition has been justified by the relative ease with which a significant source of corporate diagnostic and analytical expertise has been transformed to examples and thence to effective production rules. The source of this expertise is in completed SSME anomaly forms, one for each of the 1400+ tests conducted since 1975. The latter transformation from examples to production rules is accomplished automatically with a powerful

inductive ESBT, ExTran 7, running on a Concurrent Computer Corporation 3260 super-minicomputer. The engineering staff responsible for building (and eventually maintaining) SCOTTY has consistently used examples as input -- a single rule has yet to be written. The knowledge-acquisition "bottleneck" is thus much wider than for most previously-reported expert systems.

The moderate expansion of the former low-level efforts in constructing an automated test analysis procedure for the SSME was discussed. The topics covered qualitative and quantitative details of the above organizational and technical issues, as well as various possible extensions being considered. These included:

the integration of a large-scale relational data base system and example generation at the knowledge acquisition component of ExTran 7, and

a graphics interface for experts and end-user engineers.

References:

1. A-Razzak, R., T. Hassan, A. Ahmed, ExTran 7.2 Users Manual, Intelligent Terminals Ltd., Glasgow, Scotland, 1986.
2. Asgari, D. and K. Modesitt, "Space Shuttle Main Engine Test Analysis: A Case Study for Inductive Knowledge-Based Systems Involving Very Large Data Bases," IEEE International Conference on Computer Software & Applications (COMPSAC), 1986, pp. 66-71.
3. Concurrent Computer Corporation, Reliance DBMS, 04-338 F01 M99 R08, Oceanport, NJ, 1987.
4. Daumann, A. and K. Modesitt, "Space Shuttle Main Engine Performance Analysis Using Knowledge-Based Systems," ASME International Conference on Computers in Engineering, 1985, pp. 55-62.
5. Koen, B., Definition of The Engineering Method. American Society for Engineering Education, Washington, D.C., 1985.
6. Modesitt, K., "Experts: Human and Otherwise," Proceedings of the Third Expert Systems International Conference, London, 1987, pp. 333-341.
7. Nau, D. and J. Reggia, "Relationships between Deductive and Abductive Inference in Knowledge-based Diagnostic Problem Solving," Proceedings of the First Expert Database Systems International Workshop, Benjamin-Cummings, 1986. pp. 549-558.
8. Sopp, G., "SCOTTY: Beaming Up a New Look at SSME Performance," Threshold, Rocketdyne Division, Rockwell International, Canoga Park CA, Number 2, 1987.
9. Tuckwell, R., "Scotty's Enterprise," Computer Systems, Bromley, U.K., August, 1987, pp. 18-20.
10. Waterman, D., A Guide to Expert Systems. Addison-Wesley, 1986.

N88-16399

A EXPERT SYSTEM TO ANALYZE HIGH FREQUENCY DEPENDENT DATA FOR THE SPACE SHUTTLE MAIN ENGINE TURBOPUMPS

PREPARED
BY

RAUL C. GARCIA JR.

MEMBER OF THE TECHNICAL STAFF
ROCKETDYNE
DIVISION OF
ROCKWELL INTERNATIONAL
CANOGA PARK, CALIFORNIA

FOR

THIRD ANNUAL CONFERENCE ON ARTIFICIAL
INTELLIGENCE FOR SPACE APPLICATIONS
NOVEMBER 2-3, 1987
HUNTSVILLE, ALABAMA

SEPTEMBER 15, 1987

ABSTRACT

A prototype expert system (named ADDAMX) is being developed using a rule induction expert system development tool, EX-TRAN 7.0. ADDAMX will analyze graphically represented rotordynamic high frequency dependent data from the low and high pressure liquid hydrogen (fuel) and liquid oxygen (LOX) turbopumps on the Space Shuttle Main Engine. It infers the operation of the turbopumps by using knowledge from an expert coded into rules to analyze Spectral Data. ADDAMX infers the turbopumps operation by identifying the speed frequencies and harmonics from each respective turbopump, the frequency feed through from one turbopump to another, the bearing generated frequencies from the pump and turbine end of the turbopumps and the pseudo and super pseudo 3N frequencies from the phase two High Pressure Fuel Turbopump (HPFTP).

1.0 INTRODUCTION

Analysis of rotordynamic high frequency dependent data is a team effort and time consuming manual procedure. The Automated Dynamic Data Analysis and Management system (ADDAM) will process and graphically represent the high frequency dependent data. This will significantly reducing the processing time; however, the analysis of the data will continue to be time consuming. This bottleneck is a result of the quantity of graphs and data routinely analyzed. To further complicate the dilemma, the knowledge to analyze the data reside in a few expert analysts; therefore, the routine and challenging analysis of data are further bottlenecked.

2.0 OBJECTIVE

The objective of the project is to develop a prototype expert system using a rule induction FORTRAN based expert system development tool called EX-TRAN 7.0. The Automated Dynamic Data Analysis and Management Expert System (ADDAMX) will analyze graphically represented high frequency dependent data, in the form of Power Spectral Density Plots (PSD), collected from strategic points of measurement on the Space Shuttle Main Engine (SSME) turbopumps. ADDAMX will identify the source of specific sinusoids which are indicative of the turbopumps operation using domain-specific knowledge, analysis techniques, rules of thumb acquired during interviews with the expert analyst and knowledge from documents approved by the expert analyst. The results of the analysis will be shared with the nonexpert or expert user (relative to the subject matter) in an effective manner.

3.0 KNOWLEDGE ACQUISITION

The theory of data acquisition, data representation, data interpretation and communication within the rotordynamic arena must be understood by the author (knowledge engineer) and the reader. Two methods to gain an understanding of the data were used; the first is reading two documents which give a good comprehensive overview of the data and the second is interviewing the expert. The first document is the "Rotordynamics High Frequency Dynamic Data Summary Book SSME Turbomachinery". This is an in-house proprietary document generated by experts in the field. The second document is Application Aote 243, "THE FUNDAMENTAL OF SIGNAL ANALYSIS", a document prepared by Hewlett Packard. Using both of these documents one can begin to understand data and the language used by the expert analyst.

The second part of understanding data included interviewing the expert. The task of analyzing data is divided amongst several people who work as a team. Identification of each team player's task is important. Several lengthy sessions with the expert were held to discuss the theory of data acquisition, data representation, data interpretation and the analyzing team's structure. The interview process usually consisted of a one hour session where the author asked questions of the expert. The response usually prompted additional questions relative to the subject. After the interview the acquired knowledge, "rules of thumb" and "tricks of the trade" were reduced to a written form, then presented to the expert at the next session for review and

corrections. This continuous transfer and check process assured an accurate transfer of knowledge.

The identification of the specific duties and responsibilities of the expert and the associates helped to divide the process into steps which would later help structure the expert system. Each associate was interviewed to obtain his knowledge for the particular task he performed. Keep in mind the associates can be considered experts in their particular task; however, relative to the project they are referred to as associates.

Occasional conflicts of theory or the process of analyzing data were resolved by asking the expert to clarify the theory or process to both the knowledge engineer and the associate. This way an open communication and consistent application of the knowledge was assured. The expert had the final say in the conflict.

3.1 DATA MANAGEMENT

High frequency data is recorded onto magnetic tapes during hot fire and flight SSME engines from strategically located accelerometers on the low and high pressure liquid hydrogen (fuel) and liquid oxygen (LOX) turbopumps, preburners and the preburner pump. Thrust level of the engine and the venting and/or pressurization of the fuel and LOX tanks is also recorded. The tapes are delivered to the appropriate real-time data analysis (RTDA) labs where data is digitized and then displayed on graphs used by the Data Analyst to infer the operation of the turbopumps. The ADDAM system is specially designed to display data in several types of graphs.

The team reviews the graphs to identify "expected events" indicating the nominal operation of the turbopumps. The team identifies events which need more analysis. These latter events are referred to as anomalies and may indicate non-nominal operation of a particular turbopump.

4.0 KNOWLEDGE REPRESENTATION

The creative aspect of representing the knowledge obtained from the expert and the associates proved to be the most challenging part of the project. Several methods and schemes were attempted until the present one was adopted.

The terms used in the sections to follow are obtained from the theory and nomenclature in the EX-TRAN 7.0 user manual, June 1984, documented by Mohammed A-razzak and Thamir Hassan, Intelligent Terminal LTD., U.K. [reference 3].

A main problem is the object in question or the problem to be solved. The main problem can be divided into subproblems each with its unique or common attributes. The attribute's values, numeric or symbolic, combined in certain ways (examples) will give the subproblems unique values which contribute to the solution of the main problem. The arguments determine the value of the attribute and are obtained by special subroutines which get the value from the ADDAM system.

4.1 EXPERT SYSTEM IMPLEMENTATION

ADDAMX is divided into 2 modules. The first is a "batch mode" module where analysis of the PSD plots is executed automatically. A PSD graph plots the power spectral density in G's squared per hertz versus frequency in hertz at a selected time slice. Events on the PSD plot are identified directly on the PSD plot hardcopy. No explanation is given of how the results are obtained (see figure 4.1). The second module is a "user interactive" module where the common expert system features of explanation of analysis behavior is possible. This module has received less priority then the batch module because of the need of the batch module. The concepts presented are implemented into both modules.

The main problem for the interactive module of ADDAMX is divided into eight subproblems (see figure 4.2). Attributes within subproblems can have several different values. The main problem for the batch module of ADDAMX uses five subproblems (see figure 4.3). These five subproblems use the same attributes as the above eight subproblems and do the same analysis.

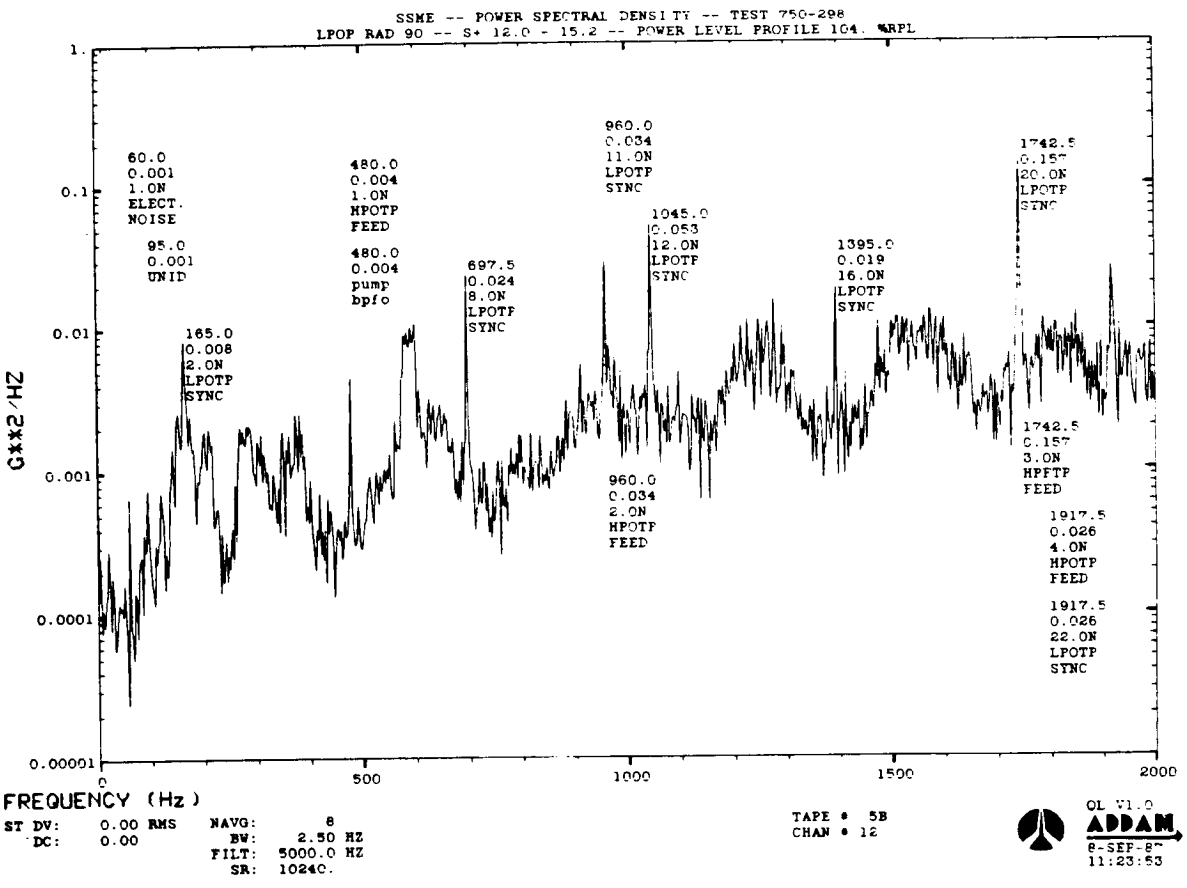


Figure 4.1: Batch module hardcopy output.

ORIGINAL PAGE IS
OE POOR QUALITY

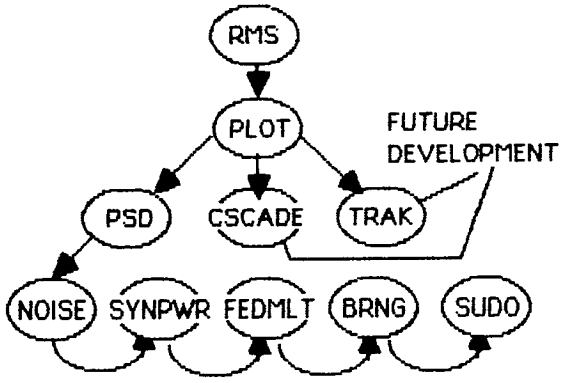


Figure 4.2: ADDAMX interactive subproblem flow chart.

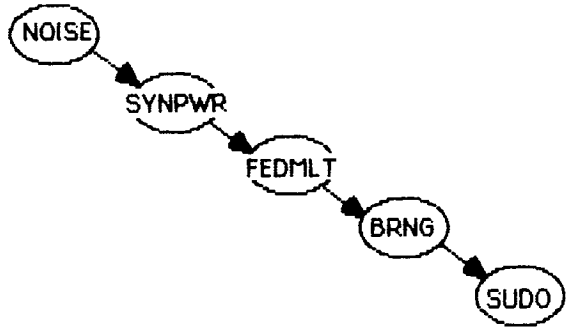


Figure 4.3: ADDAMX batch subproblem flow chart.

The ADDAMX interactive module uses two types of subproblems: "control subproblems" and "analysis subproblems". These names are unique to this project and may not be found in other literature. The control subproblems are "RMS", "PLOT", and "PSD". These subproblems control the execution of the other subproblems by directing the flow to the appropriate branch of the tree as shown in figure 4.2. The above subproblems do not directly analyze data to infer the turbopump's operation. For example, subproblem "PLOT" directs the flow of analysis to the appropriate knowledge base identified with the graph to be used to infer the turbopump operation. Each graph has its own knowledge base which deals with the graph's attributes and variables.

The analysis subproblems are "NOISE", "SYNPWR", "FEDMLT", "BRNG" and "SUDO". These subproblems analyze data directly to infer the turbopumps operation. FORTRAN IF-THEN rules are generated using an extension of Quinlan's ID3 algorithm (see Michalski, Carbonell and Mitchell, 1983) from examples obtained and derived from the knowledge aquired. The "noise" subproblem analyzes a sinusoid for the selected PSD measurement to determine if it is not indicative of a known non-turbopump frequency generating source; for example, a sixty hertz electrical cycle. Subproblem "SYNPWR" contains the knowledge for identifying a sinusoid as a speed synchronous or multiple of speed synchronous frequency for the appropriate turbopump. Subproblem "FEDMLT" identifies sinusoids which feed from one turbopump to another through the ducts, pipings and supports. Subproblem "BRNG" analyzes the sinusoid for possible origin from a turbopump pump or turbine bearing generating source. Subproblem "SUDO" analyzes the sinusoid in question to determine if it may be identified as a pseudo 3n or super pseudo 3n event.

If the sinusoid is not identified as one of the above five types of known sources of sinusoid generation then the attributes and sinusoid frequency values are written to a history file called

"HISTORY.DAT". As the system matures, more knowledge will be added to the knowledge base thus increasing the probability of identifying more expected and unknown sinusoids.

The analysts use table 4.0 as a guideline to help them identify operating windows for the turbopumps speed synchronous frequency. A window is a range above and below the mean value in which a known sinusoidal is expected to be found. The lower and upper bound for the window is obtained from a shaft speed versus power level graph. Because the new design turbopumps operate at lower speeds the window is not symmetrical about the mean; therefore, the lower bound is several hertz lower than expected. Furthermore, the lower and upper bound are not a statistical value but a guideline the analyst uses; therefore, ADDAMX, being an expert system, uses this guideline as such. Of course the integer multiples of the mean with the corresponding range shown on the table are used to identify the different multiples of speed synchronous frequency.

POWER LEVEL %	HPOTP -20 +15	HPFTP -20 +10	LPFTP -20 +15	LPOTP -10 +5
65	320	450	200	65
90	420	520	240	80
100	455	580	250	85
104	470	600	260	87
109	495	620	270	90

Table 4.0: Turbopump generated synchronous frequency (hz) SSME turbomachinery.

Table 4.1 is used to identify the bearing generated frequencies for both the pump and turbine end. Note the numbers are multiplied by the synchronous frequency for the respective pump to obtain the frequency window. "SUDO" uses data for the HPFTP 3 times speed synchronous speed plus or minus a specified number to create unique windows for the pseudo 3n and super pseudo 3n event (see figure 4.4). Subproblems "SYNPWR", "FEDMLT", "BRNG" and "SUDO" use the tabulated data to infer the turbopump operation.

Both the interactive and batch module use an analysis procedure. First a window is described using information in the above tables. This window is compared to the sinusoid in question. If the sinusoid falls within the window range then ADDAMX can hypothesize a possible frequency generated source as indicated by the respective known frequency window range. Verification of the hypothesis is done by comparing data collected from the surrounding accelerometers or other criteria to the hypothesis. A voting scheme is used in the verification step. Finally the results are printed to the screen or on to a hardcopy plot. Figure 4.5 displays a flow chart of the procedure.

	* LOX *				* FUEL *			
	HIGH PRESSURE		LOW PRESSURE		HIGH PRESSURE		LOW PRESSURE	
CLASS	TURBINE END	PUMP END	TURBINE END	PUMP END	TURBINE END	PUMP END	TURBINE END	PUMP END
BPFI	7.5(fsync)	7.5	8.0	7.6	8.0	8.0	8.0	7.6
BPFO	5.5	5.5	6.1	5.4	6.0	6.0	6.1	5.4
BSF	3.1	2.9	3.5	2.8	3.3	3.3	3.5	2.8
FTF	.42	.42	.43	.42	.43	.43	.43	.42

BPFI: Ball Pass Frequency Inner BPFO: Ball Pass Frequency Outer
BSF : Ball Spin Frequency FTF : Fundamental Train Frequency

Table 4.1: Bearing generated frequencies SSME turbomachinery

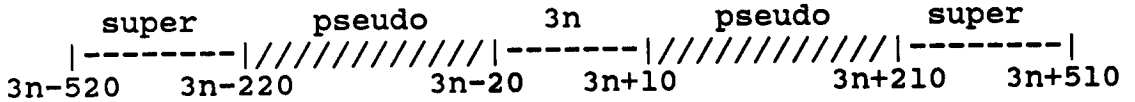


Figure 4.4: Pseudo and Super Pseudo 3n window range.

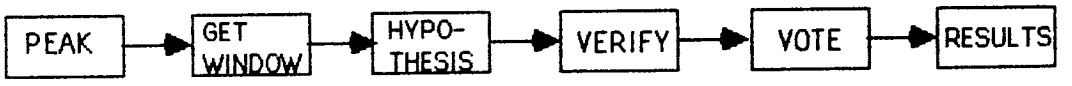


Figure 4.5: Analysis procedure.

5.0 CONCLUSION

The expert system ADDAMX identifies selected sinusoid frequencies from Spectral Data graphs as speed frequencies and harmonics from each respective turbopump, frequency feed through from one turbopump to another, frequencies generated by the turbopump bearings, pseudo and super pseudo 3N for the phase 2 HPFTP and finally electrical noise. ADDAMX does the analysis in an interactive or batch mode and the results can be displayed on the screen or hardcopy.

ADDAMX is in its infancy; however, it is helping to share the knowledge from the expert and associates with other groups. It is anticipated that the knowledge base will continue to grow to handle additional types of anomalies and increase in its capability through an organized effort from a dedicated department group.

BIBLIOGRAPHY

1. Robert Beatty and John Haworth, Rotodynamics High Frequency Dynamic Data Summary Book, SSME Turbomachinery, Rocketdyne, Canoga Park, 1978

2. Hewlett Packard, The Fundamentals of Signal Analysis, Application Note 243

3. Mohammed A-Razzak and Thamir Hassan, EX-TRAN 7.0 User Manual, Intelligent terminals limited, Glasgow GI 2AD, United Kingdom, 1984.

4. Fredrick Hayes-Roth, Donald A. Waterman and Douglas B. Lenat, Building Expert Systems, Addison-Weasley Publishing Company, INC., Reading, Massachusetts, 1983.

5. Patrick Henry Winston and Berthold Klaus Paul Horn, LISP, 2nd ed., Addison-Weasley Publishing Company, Reading, Massachusetts, 1984.

6. Patrick Henry Winston, Artificial Intelligence, 2nd ed., Addison-Weasley Publishing Company, Reading, Massachusetts, 1984.

7. Elaine Rich, Artificial Intelligence, Mcgraw-Hill Book Company, New York, New York, 1983.

N88-16400

The Use and Generation of Illustrative Examples in Computer-Based Instructional Systems

William John Selig
NASA/MSFC

and

James D. Johannes
The University of Alabama in Huntsville

ABSTRACT

Examples are both pervasive and necessary in the teaching of new material. One of the more common types is the illustrative example, used to clarify and instantiate general statements. The use of illustrative examples in computer-based instructional systems to date, when they have been used at all, has been generally limited to some form of 'canned' text. This method has the problem that as the system evolves, the canned supportive material does not necessarily follow along. This paper proposes a method whereby the underlying domain knowledge is represented such that illustrative examples may be generated on demand. This method has the advantage that the generated example can follow changes in the domain in addition to allowing automatic customization of the example to the individual.

1. Introduction

As the human race continues its venture into space, the supportive and operational systems necessary to accomplish this goal become increasingly more complex. It is becoming ever harder for an individual to grasp the overall function of the systems involved and thus it is becoming ever harder to effectively use these systems. Owing to both the growing complexity and size of our space effort, the need for efficient training systems is becoming critical. Computer based instructional systems, while still in various levels of development, hold out hope for just such efficiency of training due to their ability to tailor themselves to individuals' needs and their ability to time-share among many users [6,8]. In these instructional systems, examples will play an important role in increasing the efficiency of learning.

Examples have long been used in regular instructional materials to facilitate learning in a number of ways. Initial examples introduce material and pique user interest, thus

motivating the learning experience. Application examples anchor the learned material in a wider framework, thus increasing retention and understanding. Evidential examples and control examples are used to support or test various instructional statements. Finally, illustrative examples and counterexamples are used to both clarify and define the limits of applicability of general statements [2]. This paper is specifically concerned with the use and generation of these illustrative examples, hereafter referred to as just 'examples'.

Current approaches to the use of examples in computer-based instructional systems involve some form of presupplied text to determine what to say. These cover the range from 'canned' text to templates to feature scripts [1]. Alternatively, the system may contain an internal expert to generate an example of a 'better way', as in some coaching systems [9]. The problem with all of these systems, however, is that they depend upon someone deciding ahead of time what an appropriate example is for some given concept. In some domains, such as mathematics, this is adequate. In other domains, such as spacecraft or computer system operations, the ability to tailor the example to the user and his working environment is potentially more useful. This paper proposes just such a method.

This method of example generation uses a knowledge representation scheme based upon Sowa's conceptual graph theory. In this scheme there exists what Sowa calls a semantic net, similar to a cross-linked hierarchy, in which the basic interrelationships among the primitive concepts in the domain is captured. Domain knowledge about higher level concepts and actual entities is represented by conceptual graphs, similar to Schank's conceptual dependency theory representation, but with a potentially unlimited set of relations [5,7]. To generate an example from this domain representation, the concept to be exemplified is also represented as a conceptual graph with various attributes and parameters. This graph is then joined over the domain representation to fill in the missing pieces, either through full or partial matching on existing knowledge or generation based upon domain first principles.

The most applicable related research is that of Edwina Rissland on constrained example generation (CEG), in which an example is generated from an examples knowledge base using prespecified constraints. CEG lists three methods of 'generation'; retrieval, modification and construction. In retrieval and modification, pre-existing examples are used, either directly or with modification, that have been precompiled by the system builder. It is only in the construction phase that an example is actually generated from domain first principles, and even here the construction may instead be done by combining existing examples to create a larger example. The work on CEG is currently focused only on the retrieval and modification aspects of this method and has yet to address the construction aspect of generation from domain first principles [4]. This paper presents a method which includes that aspect.

2. Methodology

A typical situation where instructional systems would be useful in space related operations involves the learning of a command and control system. Learning these systems is similar to learning a computer operating system. There are numerous commands with various effects, possibly different in different contexts, and during the use of these commands the user must have an understanding of the overall view of things. Because of this similarity, and for practical testing reasons, this paper presents the proposed method in the domain of instruction about the UNIX operating system.

Appendix A shows a portion of the semantic net of basic domain primitive concepts. These form a framework for talking about types of objects. Note that some of these types, such as entity and information, are domain independent while others, such as command, are domain dependent.

Appendix B shows portions of the set of conceptual graphs comprising the extensional knowledge about the domain [3] and the user environment. It is there that methods would be represented for obtaining information about actual files and for interpreting that information, about a directory for example.

The best way to illustrate the proposed method is, of course, with some examples. In these examples the concept to be exemplified is first presented textually as might be composed by an instructional designer. Next is presented a conceptual graph for that concept, followed by a description of the processing involved and the resultant exemplifying conceptual graph.

Example 1

"The UNIX Programmer's Manual is kept on-line. You can use the 'man' command to print the manual pages for a system command."

```
COMMAND
    COMMAND-NAME: man
    COMMAND-ARGUMENT: *command-name
```

This conceptual graph (CG) would be matched on the command name field to the CG for the 'man' command (refer to Appendix B). Since no options are specified in the example CG, they would not be included in the match result. This resultant CG would specify that the command argument must be in the intersection of the set of manual titles and the set of command names. An entry from that intersection would be randomly chosen and inserted into the command argument field resulting in the following exemplifying graph:

```
COMMAND
    COMMAND-NAME: man
    COMMAND-ARGUMENT: sort
```

which an English generator would render as "man sort".

Example 2

The previous example was relatively simple. A more complicated example involves the explanation of the use of shell meta-characters.

"The * can be used in conjunction with ?, as in ??b*, to match multiple filenames."

```
SET
     SET-SIZE: >=2
     SET-ELEMENTS:
          FILE-NAME
               LENGTH: >=3
               PATTERN: ??b*
```

First we match this CG with the FILE-NAME CG in the semantic net. This sets the length field to 3-14. At this point we can either match against the user environment or generate filenames directly. If we choose to match against the user environment, we would call a routine to return all filenames in the user's current directory and attempt to match those against the requirements. If we matched enough of those filenames, we would return that set. Alternatively, generating the filenames would involve instantiating patterns that met both the length and pattern constraints. In either case an actor CG for matching would be invoked which knew about character patterns and meta-characters.

A generated result (compressed for brevity) might be

```
SET
     SET-SIZE: 4
     SET-ELEMENTS:
          FILE-NAME: { aab, 23bso, aabredor, debar }
```

3. Conclusions and Further Directions

This outlines the concept of system-generated examples and a method for achieving them. The present work has focused only on the representational and generational problems. Before generated examples can be fully used in computer-based instructional systems, some manner of generating English from the resultant conceptual graph is also needed. Additionally, it would be convenient to have a parser generate the initial conceptual graph from the English statement of the concept to be exemplified. However, these are separate and further areas for research. That examples are useful in regular instructional materials cannot be denied. Further work will allow them to benefit computer-based instructional systems as well.

REFERENCES

1) Bienkowski, M.A., R.E. Cullingford and M.W. Krueger, Generating Natural Language Explanations in a Computer-Aided Design System, Technical Report CS83-1, Laboratory of Computer Science Research, The University of Connecticut, 1983

2) Mandl, H., W. Schnotz and S. Tergan, On the Function of Examples in Instructional Texts, Presented at the 1984 AERA Annual Meeting

3) McGilton, H. and R. Morgan, Introducing the UNIX System, McGraw-Hill, 1983

4) Rissland, E.L., Constrained Example Generation, COINS Technical Report 81-24, University of Massachusetts at Amherst

5) Schank, R.G. and C.J Rieger III, Inference and the Computer Understanding of Natural Language, in Readings in Knowledge Representation, R.J. Brachman and H.J. Levesque (eds), Morgan Kaufman, 1985, pp. 119-139

6) Sinnot, L.T., Generative Computer-Assisted Instruction and Artificial Intelligence, Educational Testing Service, October, 1976

7) Sowa, J.F., Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984

8) Barr, A. and E.A. Feigenbaum (eds), The Handbook of Artificial Intelligence, V2, William Kaufman, 1982, pp. 225-8

9) Barr, A. and E.A. Feigenbaum (eds), The Handbook of Artificial Intelligence, V2, William Kaufman, 1982, pp. 254-60

APPENDIX A - Intensional Knowledge

This is a simplified representation showing only the concept types and not the relations linking the concepts involved. All relation types in this example are characteristics and are indicated by indentation.

```
ENTITY < (is a subtype of)  UST (the universal subtype)
     includes physical objects as well as abstractions

INFORMATION < UST          Anything that can be communicated


FILE-NAME < ENTITY, INFORMATION     The entity used to access a file
     LENGTH: 1-14
     PATTERN: { valid-characters }

COMMAND-NAME < FILE-NAME

COMMAND-DESCRIPTION < TEXT

COMMAND-OPTION < INFORMATION
     OPTION-LETTER
     OPTION-DESCRIPTION
     OPTION-ARGUMENT

COMMAND < ENTITY                          Performs an operation
     COMMAND-NAME
     COMMAND-OPTION
     COMMAND-ARGUMENT
     COMMAND-DESCRIPTION
```

APPENDIX B - Extensional Knowledge

```
COMMAND
     COMMAND-NAME:
          LENGTH: 3
          PATTERN: man
     COMMAND-DESCRIPTION: "finds information by keywords;  prints
                           selected manual pages"
     COMMAND-OPTION: (
          OPTION-LETTER: k
               -DESCRIPTION: "prints  a 1-line synopsis of  each
                              manual  section whose  listing
                              in   the   table   of   contents
                              contains one of the keywords"
               -ARGUMENT: { keywords },
          OPTION-LETTER: t
               -DESCRIPTION: "forces use of TROFF format"
                         )
     COMMAND-ARGUMENT: { manual-titles }

manual-titles: {at,awk,cat,cc, ... ls,man, ... sort,tail,wc }
```

N88-16401

INTERACTIVE KNOWLEDGE ACQUISITION TOOLS

Martin J. Dudziak and Jerald L. Feinstein
ICF/Phase Linear Systems, Inc.
9300 Lee Highway
Fairfax VA 22031-1207
(703) 934-3800

Abstract: This paper discusses the problems of designing practical tools to aid the knowledge engineer and general applications used in performing knowledge acquisition tasks. At issue for the knowledge engineer are several problems and misconceptions of knowledge engineering and knowledge-based systems development. The authors propose a strategy for removing some of those problems, presenting a particular approach we have developed for one class of knowledge acquisition problem characterized by situations where acquisition and transformation of domain expertise are often the bottleneck in systems development. The focus at ICF/Phase Linear has been upon the processing of text-based source materials through a software tool designed in-house, the Knowledge Acquisition Module (KAM). The authors go on to discuss how the tool and the underlying software engineering principles can be extended to provide a flexible set of tools that allow the application specialist to build highly-customized knowledge-based applications.

Introduction

There are some misconceptions or misrepresentations regarding what knowledge engineering can or should do. These confusions result in a failure to make the best use of computer technology and artificial intelligence-based techniques for building knowledge-based systems that are reliable and effective for real-world applications. The knowledge engineering process is typically characterized as follows:

(1) There is a body of expert knowledge "out there" which is in the minds of certain experts (or what they have produced -documents, automated systems, etc.).

(2) This knowledge can be codified or summarized into a formal representation which can then be used by an automated system.

(3) The knowledge engineer must "obtain" that body of knowledge and transform it into the ideal type of symbolic representation - discovering the ideal formalism is a goal that must be attained.

(4) The symbolic representation must be implemented into an expert system where there is a mapping of the symbolic representation into some form of code.

(5) Once built the expert system should perform its assigned tasks in a manner that is predictably similar to the way a human expert would carry out those tasks.

A fundamental misconception is that a comprehensive body of knowledge exists in the first place which can be codified into a formal representation. There is a tendency to think of knowledge as objects, facts as being entities that can be bounded and enclosed within the descriptive framework of a given type of formalism. There is also a tendency to think of the mapping problem (expert knowledge into symbolic form) as a task that has a singular and finite answer. However, putting automated systems aside and considering for a moment only human exchanges of information and learning, it is clear that acquisition and transfer of knowledge is not a linear sequence or an early codifiable phenomenon. The expert-novice interchange is highly iterative and interpolative. By this it is meant that the exchanges are more like conversations rather than data transfers as we normally think of them. [1] As with conversations, the implied background knowledge of both persons in the exchange becomes highly significant for the correct interpretation of what is spoken by both (all) participants.

There must be a high level of dialogue, particularly interrogation in both directions between expert and novice. This "handshaking", as it were, is what enables both participants to know that the other is understanding what is being communicated. Such questioning enables the novice to make clear what is understood and what is unclear and what is his or her context of understanding; it also empowers the expert with knowledge about the communication process so that he can emphasize or clarify certain facts, rules, and relationships. In a knowledge acquisition activity, particularly between engineer and expert, frequent questioning and clarification is the key to making sure that both are "speaking the same language." Of course, this often leads to a increased volume of written and verbal material to be analyzed and deciphered; thus the need for automating parts of those processes.

The knowledge engineer acts as both the go-between for an expert and an automated system and also as the designer of that computer-based product and must recognize this dual nature to this work. The knowledge engineer must take the lead in focusing the knowledge acquisition process so that it serves to not only provide substantive expressions of the expert's knowledge but information that will help in designing the most appropriate system structures for using that knowledge in the automated application. The knowledge engineer is responsible in a way unlike the typical apprentice to the expert in that what is relevant or useable information must be defined. Also the system design must be modified in response to new expert information that is gained through the interviews, dialogues and other acquisition activities.

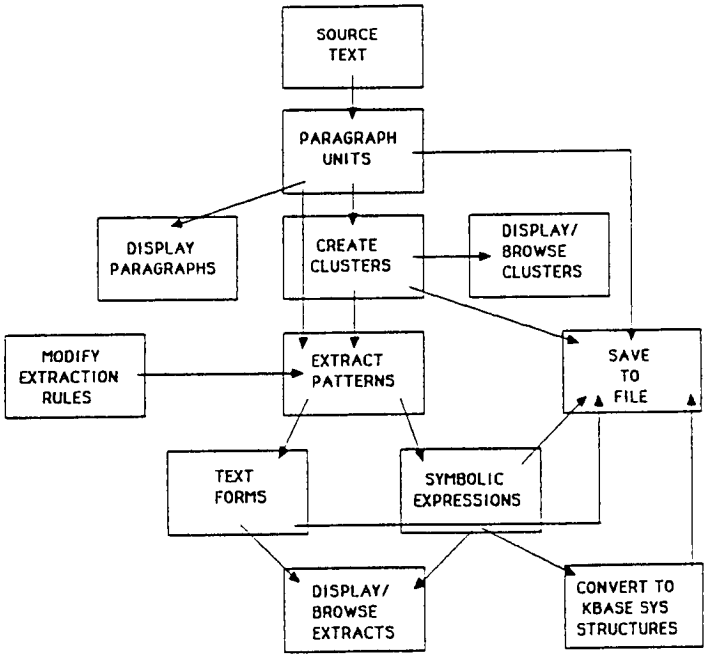
It is critical to keep the knowledge engineer active at every level of the acquisition process, from interfacing with the experts to formulating a computer-based representation for what was obtained. This provides the broad-context element of control which can take into account not only the expert source material currently being processed (e.g., codified) but also the knowledge which may not be coded into any automated system and which may not at the moment seem directly relevant but which can become relevant in the future. But if the knowledge engineer is to be actively involved in the entire acquisition-representation process, the tools that will help carry the load and to perform those tasks in a reasonable period of time must be available.

In brief, the knowledge engineer needs tools that can expedite these tasks but not perform them without active participation and control. These tools must enhance productivity and correctness without adding to the work load, rather than tools which replace entire segments of the knowledge acquisition process. If the knowledge engineer is taken out of the loop, so to speak, then the opportunity to bring in the broad-context of both acquired or implicit expert knowledge as well as general common-sense knowledge is reduced. Moreover, the task of identifying situations where a highly-automated module has generated something in need of correction or has omitted something, and the task of making those corrections or modifications "after the fact", may be so time-consuming and tedious that the value of the initial automation process is negated.

One insufficiently-addressed aspect of the knowledge engineering process which can be significantly improved and which has been the focus of ICF/Phase Linear automation efforts concerns the extraction of information imbedded in free-format source materials (e.g., texts, transcripts) and the transformation of such knowledge into useable formal representations. The latter may be in the syntax of the knowledge engineer's application system under development or in some intermediate form which the knowledge engineer may have adopted. There are problems not only in transforming loosely-formed knowledge into a codified symbolic representation but also in handling voluminous and diverse-format database records, text, interview transcripts, and other digitizable material. The problems in acquiring the knowledge properly also affect the selection and formalization of a sufficiently robust representational scheme to be used in the actual expert system, planner, scheduler or other application. The knowledge engineer needs to have fluent and easy access to the breadth and depth of relevant source material to effectively design data structures that will store facts, rules, relations and to design or select the reasoning mechanisms that will manipulate the knowledge bases.

In its project management and consulting work, ICF/Phase Linear staff have frequently found situations where the volume of interview transcripts, background texts, and reference materials,

particularly manuals and project specification documents, posed the major roadblock to establishing even an elementary knowledge base for an application. This has been particularly true for such engineering applications as autonomous underwater and aerospace vehicle control, mission planning, fault diagnosis and maintenance. The approach ICF/Phase Linear has been developing consists of building relatively simple software tools which to reduce the amount of material which the engineer must handle. Such tools also help to create intermediary data structures that can be directly applied toward the next phases of the knowledge engineering process such as incorporation into relations, facts, and rules in a knowledge base. The goal has been to allow the knowledge engineer to move quickly through large and difficult masses of data and to provide the ability to create new data structures that are more condensed, focused and easily managed, primarily through rapid browsing and editing. The result is that the engineer can be more involved in the full knowledge acquisition process without being overwhelmed by time-consuming operations.



KAM FUNCTIONAL STRUCTURE

KAM or the Knowledge Acquisition Module has been implemented as the kernel tool in a family of such expediter tools. As such, it is a concrete example of how some of the strategies outlined above can be feasibly implemented in a low-cost software package running on low-cost general-purpose hardware such as the PC family of microcomputers. The figure above illustrates the basic KAM functional structure.

In its first phase KAM has been built to handle source files of text data but it can be extended to work with non-text data as

well. Most applications for this type of program will involve text but in a variety of different formats besides standard paragraph-oriented text files. The primary features KAM provides are:

- The ability to rapidly form and browse through clusters or subsets of text arranged according to topics (the latter being specified using strings and keywords but also logical relationships between words, presence of synonyms and morphemes).

- The ability to extract elements of text on the sentential level which match pattern templates (rules that are defaults or established by the user) and to browse through these extracted patterns.

- The reconstruction of extracted text elements into both simple English-like and code-like user-specified representational forms which can be used directly or after editing in application systems.

- The ability to browse quickly among extracts and source text and to make edits to extracts which are automatically reflected in the other representational forms of the extracts.

- The ability to work with multiple source files and extract data sets and to combine data produce from different sources.

- The ability to automatically generate data structures from extracts according to pre-specified syntax rules such that the output data sets can be input into the knowledge bases of various existing applications.

KAM is currently implemented in LISP on a PC/AT. It has been developed as a prototype following an initial proof-of-concept version built on a Symbolics workstation, and further extensions and refinements of the system will be oriented toward deliverability on a variety of hardware besides the PC/AT.

While KAM is designed to be used in a standalone capacity for extracting and transforming text, it can also be incorporated, modular-fashion, into a more comprehensive workstation environment that provides the user with the capability of defining and using ad hoc "knowledge-object" definitions - software structures which specify different classes and types of data that the user may discover a need to use during the knowledge acquisition life cycle. An example of such a definition is one called a TopicDef; it is a frame-like structure that specifies the different rules and functions to be employed by KAM and related applications for determining whether or not a given piece of text should be considered as bearing reference to a topic or not. The TopicDef instructs KAM as to how the user conceives of the topic and how

the system should proceed in its examination of texts in order to judge its relevance or not.

Only brief mention has been made of the fact that there is a broad class of straightforward software tools for the knowledge acquisition process and that like KAM, they can be developed into standalone units or integrated to form a workstation environment. Such an environment is the application user's equivalent of a programmer's development environment as is found on a variety of machines, the most obvious perhaps being UNIX on conventional hardware and the Symbolics LISP machines. By providing more fluidity and convenience at the workstation, the knowledge engineer can address the acquisition problem and increase productivity in much the same fashion as the AI programmer can more effectively grapple with program design and prototyping issues through well-established features like incremental compilation, run-time debugging and editor-level evaluations. The behaviors of programming and knowledge acquisition are not that dissimilar. It seems appropriate to expand the tools which have proven successful in the programming arena toward time-consuming, productivity-draining problems in critical application areas like knowledge acquisition.

[1] A source for much of the theoretical foundation for concepts expressed in this paper is: Winograd, T. & Flores, F., "Understanding Computers and Cognition", Addison-Wesley, Reading MA, 1987

N88-16402

AUTOMATIC MATHEMATICAL MODELING for SPACE APPLICATION

Caroline K. Wang

Software and Data Management Division
Information and Electronic Systems Laboratory
Science and Engineering Directorate
Marshall Space Flight Center/ NASA
Huntsville, Alabama

I. ABSTRACT

This paper describes a methodology for Automatic Mathematical modeling. The major objective is to create a very friendly environment for engineers to design, maintain and verify their model and also automatically convert the Mathematical model into FORTRAN code for conventional computation.

A demonstration program was designed for modeling the Space Shuttle Main Engine Simulation Mathematic Model called Propulsion System Automatic Modeling (PSAM). PSAM is written in LISP and MACSYMA and runs on a Symbolics 367Ø Lisp Machine. PSAM provides a very friendly and well organized environment for engineers to build a knowledge base for base equations and general information. PSAM contains an initial set of component process elements for the Space Shuttle Main Engine Simulation and a questionnaire that allows the engineer to answer a set of questions to specify a particular model. PSAM is then able to automatically generate the model and FORTRAN code. A future goal is to download the FORTRAN code to the VAX/VMS system for conventional computation.

II. INTRODUCTION

Mathematical Modeling for Space Application Simulationproject is a very complicated process which includes Analysis, Design, and the Generation of complex equations. Generally the model will require several modification before it will match a real system. Historically the modifications have been time consuming and a fertile source of error.

The use of Artificial Intelligence techniques has shown that this process can be simplified. This paper describes a methodology for organizing and generating the model automatically.

III. EXPERT USER INTERFACE and AUTOMATIC KNOWLEDGE BASE GENERATION

An Artificial Intelligence system can provide a very friendly work environment for engineers. Through PSAM's questionnaire, the system collects the necessary information and generates the knowledge base automatically.

The knowledge base includes: (1) An information base in frames and
(2) Generic equations for the project.

IV. SOFTWARE TOOLS and AUTOMATIC EQUATION AND CODE GENERATION

There are a number of tools available for equation derivation and FORTRAN code generation. The one we choose to use is "MACSYMA". LISP was the language I used basically for user interface and knowledge base generation. I also used LISP for generating "MACSYMA" code. The MACSYMA code generates the equations automatically and at the same time FORTRANcode will also be generated automatically and saved into the disk file for future integration.

The future goal is to integrate the FORTRAN code we generated and download to the VAX/VMS system for conventional computation.

V. MAINTAINING AND VERIFYING

One of the most difficult problems in software today is the verification and maintenance of existing programs, especially programs built up our time with many programmers involved. This is particularly true for simulation programs. The traditional way of modifying simulation programs by rebuilding the model and recoding the model had a potential source of many errors.

The automatic system can eliminates most of these problems. The proce-dure become much simpler which makes it easy for the user to maintain and modify the model and program directly.

VI. EXAMPLE

A demonstration program was designed for modeling the Space Shuttle Main Engine Simulation Mathematic Model called Propulsion System Automatic Modeling (PSAM). PSAM is written in LISP and MACSYMA and runs on a Symbolics 367Ø LISP machine.

The design goals for PSAM were to develop automatic modeling skills for Propulsion System, and other scientific and Engineering applications. We used the old Engine Model for an example to study.

PSAM includes the following features:

(1). Friendly user interface.
(2). Automatic Knowledge Base generation and
(3). Automatic Equation and Coding generation.

The Space Shuttle Main Engine Simulation model was built up from the component process elements and their combination into the subprograms.

The component process elements are Pump,Hot gas turbine, Hydraulic turbine, Turbopump, Combustor, Valve, Incompressible propellant flow, Injector volume with priming for start, Hot gas heat transfer and Regen cooling flow. The subprograms are Fuel, Oxidizer and Hot Gas.

There are two types of information for a PSAM knowledge base. One is the component process elements generic equations and the other is the information base for the combination of the Space Shuttle Main Engine model subprograms and component process elements.

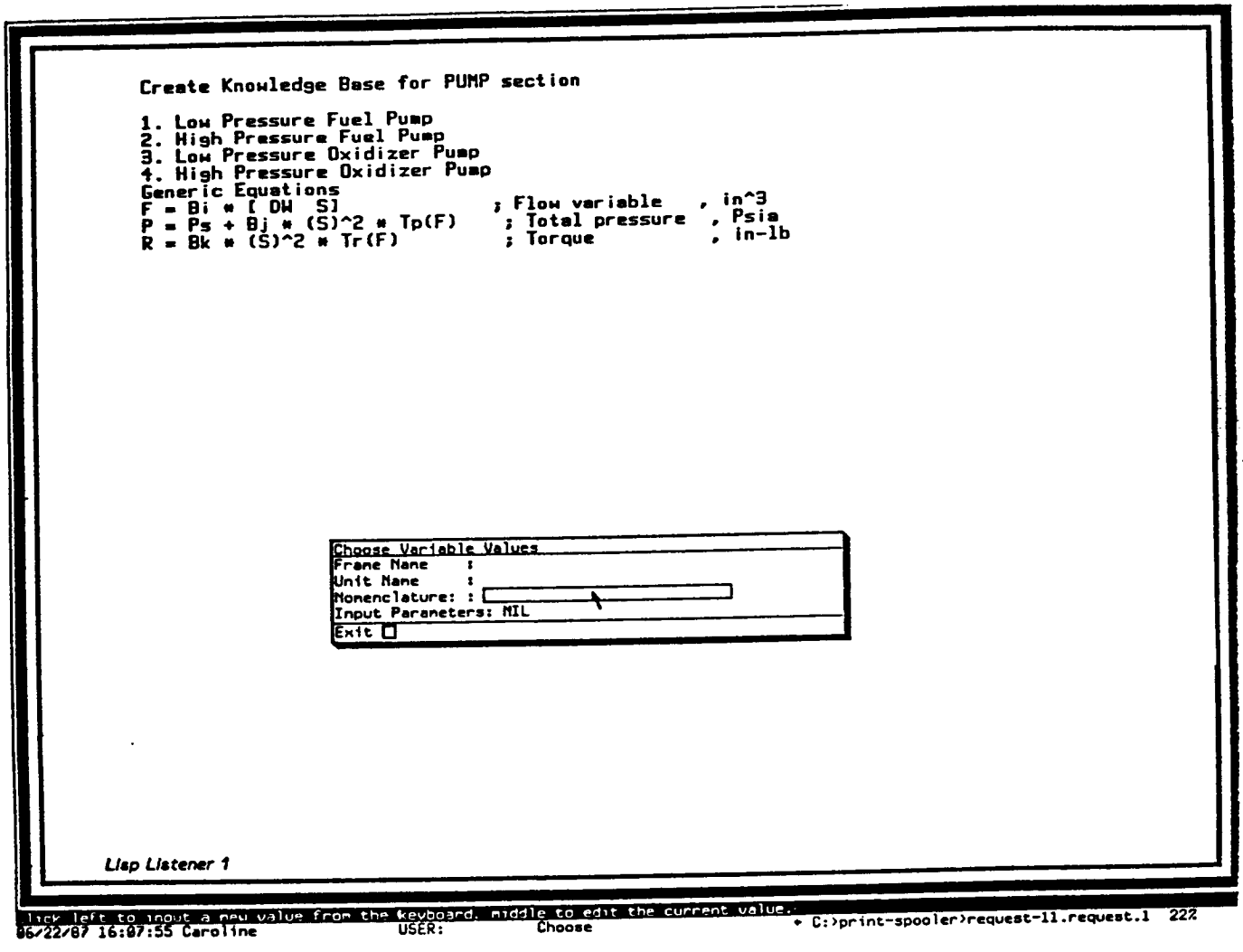
The Expert System collects the detailed requirements and generates

the sets of specific equations for the component process elements and subprograms.

PSAM has the ability to:

(1) Create or maintain the Knowledge base
(2) Load different knowledge base
(3) Automatically generate Equations
(4) Output generated Equation or FORTRAN code to Disk file or option for print out of the Laser printer.

Example for input information user interface for the component process element Pump knowledge base.



ORIGINAL PAGE IS
OF POOR QUALITY

```
(Deframe Ffp1
          (Class PUMP)
          (Unit "flow-variable")
          (Nomenclature "Low Pressure Fuel Pump Flow variable")
          (Input-parameter B11 DWfd1 Sf1))
(Deframe Ffp2
          (Class PUMP)
          (Unit "flow-variable")
          (Nomenclature "High Pressure Fuel Pump Flow Variable")
          (Input-parameter B18 DWfd2 Sf2))
(Deframe Fdp1
          (Class PUMP)
          (Unit "flow-variable")
          (Nomenclature "Low Pressure Oxidizer Pump Flow variable")
          (Input-parameter B27 DWmov+DWop3 So1))
(Deframe Fop2
          (Class PUMP)
          (Unit "flow-variable")
          (Nomenclature "High Pressure Oxidizer Pump Flow variable")
          (Input-parameter B39 DWmov+DWot1+DWop3 So2))
(Deframe Pfd1
          (Class PUMP)
          (Unit "total-pressure")
          (Nomenclature "Low Pressure Fuel Pump Discharge Total pressure")
          (Input-parameter Pfs B12 Sf1 Tpfp1))
(Deframe Pfd2
          (Class PUMP)
          (Unit "total-pressure")
          (Nomenclature "High Pressure Fuel Pump Total Pressure")
          (Input-parameter Pfd1 B19 Sf2 Tpfp2))
(Deframe Pod1
          (Class PUMP)
          (Unit "total-pressure")
          (Nomenclature "Low Pressure Oxidizer Pump Total Pressure")
          (Input-parameter Pos B28 SØ1 Tpop1))
(Deframe Pod2
          (Class PUMP)
          (Unit "total-pressure")
          (Nomenclature "High Pressure Oxidizer Pump Total Pressure")
          (Input-parameter Pod1 B4Ø So2 Tpop2))
(Deframe Rfp1
          (Class PUMP)
          (Unit "torque")
          (Nomenclature "Low Pressure Fuel Pump "torque"")
          (Input-parameter B13 Sf1 Trfp1))
(Deframe Rfp2
          (Class PUMP)
          (Unit "torque")
          (Nomenclature "High Pressure Fuel Pump "torque"")
          (Input-parameter B2Ø Sf2 Trfp2))
(Deframe Rop1
          (Class PUMP)
          (Unit "torque")
          (Nomenclature "Low Pressure Oxidizer Pump "torque"")
          (Input-parameter B34 So1 Trop1))
```

```
PSAM Generic equations:

"**********PUMP*************"$


"Pump Flow Variable"$

PFV(F,B,DW,S):=
Block ( F = B * (DW /S));

"Pump Total Pressure"$

PTP(PP,P,B,S,T):=
Block ( PP = P + B * (S)^2 * T);

"Pump Torque"$

PT(R,B,S,T):=
Block ( R = B * (S)^2 * T);


"**********TURBINE**********"$


"Turbine Torque"$

TT(R,B,P,T):=
Block ( R = B * P * T);

"Turbine Speed Parameter"$

TSP(M,B,S,T):=
Block ( M = B * S / (T^(1/2)));

"Turbine Weight Flowrate"$

TWF(DW,B,dP,r):=
Block ( DW = (B * (dP) * r)^(1/2));

"**********TURBOPUMP SPEED*******"$

"TurboPump Speed"$

TPS(S,B,Tq,SØ):=
Block ( S = B * 'Integrate ( Tq,t) + SØ);


"**********VALVE**********"$


"Valve Area"$

VA(A,Ab,T,Th,Thb):=
Block ( A = Ab * (T * Th / Thb));
```

```
"Valve Total Pressure Inlet Total"$

VTPIT(P,Pi,RF,DW,Rho):=
Block ( P = Pi - RF * (DW)^2 /Rho);

"Valve Total Pressure Inlet Static"$

VTPIS(PP,P,DW,A,Rho):=
Block ( PP = P - DW^2 /(772.8 * A^2 * Rho));

"Valve Delta Total Pressure"$

VDTP(DP,B,Rhob,Rho,DW,A,Ab,RF):=
Block ( DP = B * (Rhob/Rho)*(DW/(A/Ab))^2 - RF * DW /Rho);


"**********FLOW**********"$


"Fuel Flow"$

FF(DWf,Bi,P9,P,Bj,R,DWØ):=
Block ( DWf = B * 'integrate (((P9 - P - (Bi / R)) * DWf^2),t) + DWØ);

"Oxidizer Flow"$

OF(DWo,Bk,Ppos,P,Bl,DWØ,A,Ab,Bm,DWi):=
Block( DWo = Bk * 'integrate ((Ppos - p - Bl * (DWo /(A/Ab))^2 -
Bm*(DWi)^2),t) + DWØ);

"Variable in injector Priming function, Dimensionless"$

VIPF(E,Bh,DWo,DWi,EØ):=
Block( E = Bh * 'integrate ((DWo - DWi),t) + EØ);

"Weight Flowrate Injector"$

WFI(DWi,DWo,Eo):=
Block( DWi = DWo * Eo);


"**********COMBUSTER**********"$


"Combuster Total Pressure"$

CTP(P,Bi,DW1,DW2,Bj,DW3,PØ):=Block( P = Bi * 'integrate (( DW1 + DW2 - Bj *
DW3),t) + pØ);

"Combuster Fuel Weight Flowrate"$
```

```
CFWF(F,DW2,DW):=Block( F = DW2 / (DW1 + DW2));

"Combuster Temperature"$

CT(T,R,F,Bk,T9):=Block ( T = R(F) + Bk * T9);

"**********COOLING ELEMENT**********"$

"Cooling Element Total Pressure"$

CETP(P,B,DQw1,DQw2,H3,DWi,H,DW,PØ):=
Block( P = B * 'integrate((DQw1 + DQw2 + H3 * DWi - H * DW),t) + PØ);

"Cooling Element Specific Enthalpy"$

CESE(H,B,T):=Block( H = B * T);

"Cooling Element Weight Flowrate Main Chamber Heat Exchanger"$

CEWFM(DW,Bi,P,P5,Bj,R5,DW,DWØ):=
Block( DW = Bi * 'integrate ((P - P5 - (Bj / R5) * DW^2),t) +DWØ);

"Cooling Element Density"$

CED(R,B,DWi,DW,RØ):=
Block( R = B * 'integrate ((DWi - DW),t) + RØ);

"Cooling Element Temperature"$

CET(T,B,P,R):=
Block( T = B * ( P / R));

"Cooling Element Heat Transfer Rate Hot Gas Wall"$

CEHTRHGW(DQw1,B,T,Tw1,DWi):=
Block( DQw1 = B * (1.Ø + Ø.ØØ2 * T) * (DWi)^Ø.8);

"Cooling Element Heat Transfer Rate Ambient Thrust Chamber"$

CEHTRAT(DQw2,B,Tw1,T,DWi):=
Block( DQw2 = B * (1.Ø + Ø.ØØ2 * T) * (Tw2 - T) * DWi^Ø.8);

"Cooling Element Heat Transfer Rate TC"$

CEHTR(DQtc,B,Tc,Tw1,DWcn):=
Block( DQtc = B * (Tc - Tw1) * DWcn^Ø.8);

"Cooling Element Hot Gas Wall Temperature"$

CEHGWT(T,DQtc,DQw1,TØ):=
Block( T = B * 'integrate ((DQtc - DQw1) * temp(t),t) + TØ);
```

The Macsyma code that psam generates which will call the Generic equation function blocks and also generates the equations and FORTRAN code for the certain element.

```
fpump():=block(writefile("a1:>psam>equation>pump-equation."),
display(PT(ROP1,B34,SO1,TROP1)),
display(PT(RFP2,B2Ø,SF2,TRFP2)),
display(PT(RFP1,B13,SF1,TRFP1)),
display(PTP(POD2,POD1,B4Ø,SO2,TPOP2)),
display(PTP(POD1,POS,B28,SØ1,TPOP1)),
display(PTP(PFD2,PFD1,B19,SF2,TPFP2)),
display(PTP(PFD1,PFS,B12,SF1,TPFP1)),
display(PFV(FOP2,B39,DWMOV+DWOT1+DWOP3,SO2)),
display(PFV(FDP1,B27,DWMOV+DWOP3,SO1)),
display(PFV(FFP2,B18,DWFD2,SF2)),
display(PFV(FFP1,B11,DWFD1,SF1)),
closefile(),
writefile("a1:>psam>fortran>pump.for"),
fortran(PT(ROP1,B34,SO1,TROP1)),
fortran(PT(RFP2,B2Ø,SF2,TRFP2)),
fortran(PT(RFP1,B13,SF1,TRFP1)),
fortran(PTP(POD2,POD1,B4Ø,SO2,TPOP2)),
fortran(PTP(POD1,POS,B28,SØ1,TPOP1)),
fortran(PTP(PFD2,PFD1,B19,SF2,TPFP2)),
fortran(PTP(PFD1,PFS,B12,SF1,TPFP1)),
fortran(PFV(FOP2,B39,DWMOV+DWOT1+DWOP3,SO2)),
fortran(PFV(FDP1,B27,DWMOV+DWOP3,SO1)),
fortran(PFV(FFP2,B18,DWFD2,SF2)),
fortran(PFV(FFP1,B11,DWFD1,SF1)),
closefile());
```

$$ROP1 = B34\ SO1^2\ TROP1$$

$$RFP2 = B2Ø\ SF2^2\ TRFP2$$

$$RFP1 = B13\ SF1^2\ TRFP1$$

$$POD2 = B4Ø\ SO2^2\ TPOP2 + POD1$$

$$POD1 = B28\ SØ1^2\ TPOP1 + POS$$

$$PFD2 = B19\ SF2^2\ TPFP2 + PFD1$$

$$PFD1 = B12\ SF1^2\ TPFP1 + PFS$$

$$FOP2 = \frac{B39\ (DWOT1 + DWOP3 + DWMOV)}{SO2}$$

$$FDP1 = \frac{B27\ (DWOP3 + DWMOV)}{SO1}$$

$$FFP2 = \frac{B18\ DWFD2}{SF2}$$

$$FFP1 = \frac{B11\ DWFD1}{SF1}$$

```
ROP1 = B34*SO1**2*TROP1
RFP2 = B2Ø*SF2**2*TRFP2
RFP1 = B13*SF1**2*TRFP1
POD2 = B4Ø*SO2**2*TPOP2+POD1
POD1 = B28*SØ1**2*TPOP1+POS
PFD2 = B19*SF2**2*TPFP2+PFD1
PFD1 = B12*SF1**2*TPFP1+PFS
FOP2 = B39*(DWOT1+DWOP3+DWMOV)/SO2
FDP1 = B27*(DWOP3+DWMOV)/SO1
FFP2 = B18*DWFD2/SF2
FFP1 = B11*DWFD1/SF1
```

The sets or the FORTRAN code will then be integrated into a completed compiled SSME program and downloaded to VAX/VMS system for conventional computation.

N88-16403

LISP Based Simulation Generators for Modeling Complex Space Processes

Fan T. Tseng
Bernard J. Schroer
Wen-Shing Dwan
University of Alabama in Huntsville
Huntsville, AL 35899
(205) 895-6510

ABSTRACT

This paper presents the development of a simulation assistant for modeling discrete event processes. Included in this paper are an overview of the system, a description of the simulation generators, and a sample process generated using the simulation assistant.

INTRODUCTION

Numerous simulation languages exist for modeling discrete event processes and have now been ported to microcomputers (Pegden 1985 and Minuteman 1986). Graphic and animation capabilities have been added to many of these languages to assist the users build models and evaluate the simulation results.

However, with all these languages and added features, the user is still plagued with learning the simulation language which can be rather time consuming, especially if a complex system is being modeled. Furthermore, the time to construct and then to validate the simulation model is always greater than originally anticipated.

One approach to minimize the time requirement is to use pre-defined macros that describe various common processes or operations in a system. The literature recently has contained several approaches to developing and using these macros or simulation generators. For example, a simulation generator has been developed that translates the characteristics of a Flexible Manufacturing System (FMS) into a simulation model using SIMAN (Haddock 1987).

Another example is a ruled based expert system that assists the modeler in constructing simulation models (Khoshnevis 1986). The expert system automatically generates the corresponding SLAM simulation code. The icons used in this system are very similar to those in GPSS and SIMAN.

A Natural Language Interface (NLI) for an electronics assembly domain has also been developed that automatically generates SIMAN code from user defined text (Ford 1986). This NLI uses a Symbolics 3670 processor and presently has a limited dictionary.

A set of simulation generators has also been developed for simulating manufacturing systems (Schroer 1987). These generators are written in GPSS/PC and can be linked together through a GPSS main program.

SYSTEM OVERVIEW

Several simulation generators have been developed for modeling manufacturing processes (Tseng 1987). These simulators have greatly expedited the modeling process. The next step is to develop a more user friendly interface, or simulation assistant, between the modeler and the actual simulation language. Such a simulation assistant should reduce the user's need to know the details of the simulation language and reduce the time to construct and validate the model. The end result should be increased user productivity.

Figure 1 outlines a system schematic for the simulation assistant currently under development. The user sits at a Symbolics 3620 processor and, based on the prompts from the simulation assistant, defines the process or the model. The simulation assistant is written in Symbolics' Common Lisp.

Once the user has defined the process to be simulated, the simulation assistant automatically generates the corresponding GPSS simulation code of the process. The simulation assistant also calls on the library of simulation generators in generating the GPSS code.

The Symbolics 3620 is interfaced to a 3670 file server which is interfaced, via Ethernet, to a VAX 785. Resident of the VAX is the GPSS simulation system. The user then inputs the necessary run statements, the model is executed and the results printed. Currently, the VAX interface is not operational. Instead, the GPSS code that is generated on the Symbolics 3620 is re-entered into an IBM AT which has resident the GPSS system. The user then inputs the run commands on the AT to execute the model.

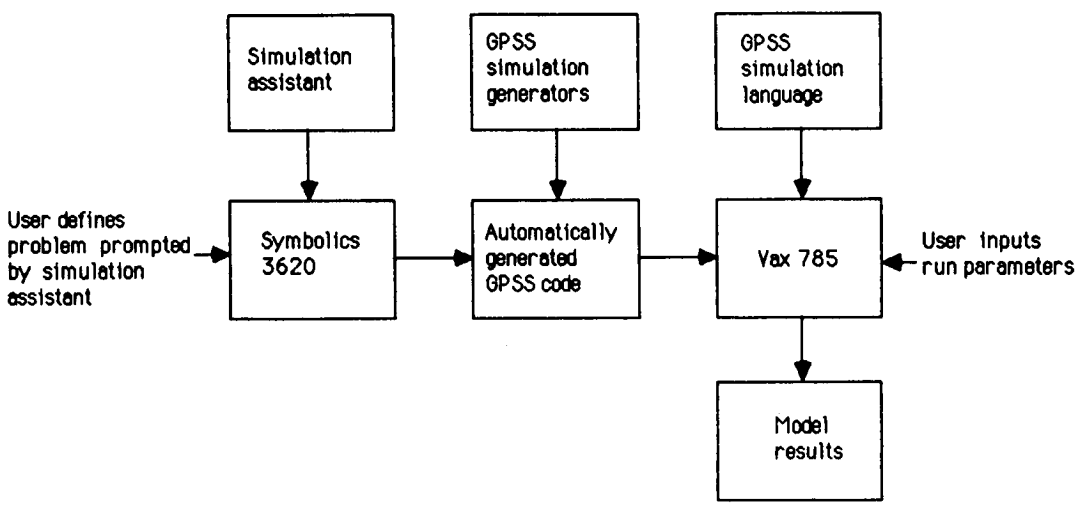


Figure 1. System overview

SIMULATION GENERATORS

Three simulation generators have been developed: an assembly station generator, a manufacturing cell generator, and an inventory transfer generator. Each generator consists of a GPSS macro or subroutine (Tseng 1987).

Figure 2 is a typical assembly station. The generator operates as follows. Items wait in a queue until the station becomes available. The item then seizes the station and waits until item A is available at stock point A. An amount of time is then simulated while item A is assembled to item B resulting in item C. If stock point A is empty, a signal is sent to the inventory transfer generator to move the empty cart to the manufacturing cell. The manufacturing cell then makes the required items for the cart. Another signal is sent to the inventory transfer generator to return the full cart back to the corresponding assembly station. This method of inventory control is commonly called the pull mode. If the assembly station is operated in the push inventory control mode, the items at stock point A are replenished at predefined intervals.

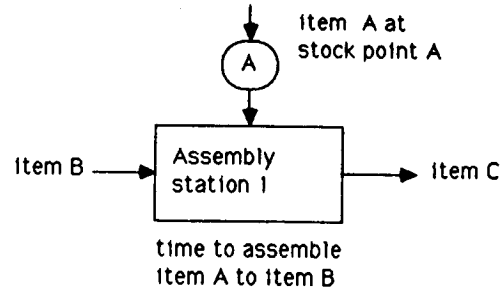


Figure 2. Typical assembly station

Figure 3 is a typical manufacturing cell. The generator operates as follows. Item D, or raw material D, is used to manufacture item E. Likewise, item F is used to make item G. An amount of time is then simulated to make an item. The finished items are stored in stock points E and G. Each stock point contains a defined inventory or carts with a fixed cart capacity. This output can go to either another manufacturing cell or an assembly station. The inventory at the stock points D and F can be raw material or items from another manufacturing cell.

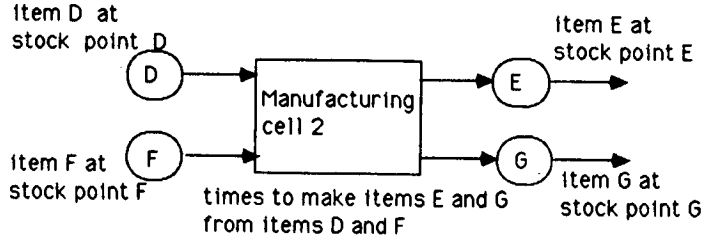


Figure 3. Typical manufacturing cell

SIMULATION ASSISTANT

An example of a typical process, which could translate to a space manufacturing task, is given in Figure 4 and consists of two assembly stations and one manufacturing cell. The following constraints are placed on the process: one inventory transfer generator for all item movement; a pull inventory control is imposed for items B and D; and a push inventory control is imposed for items A and C.

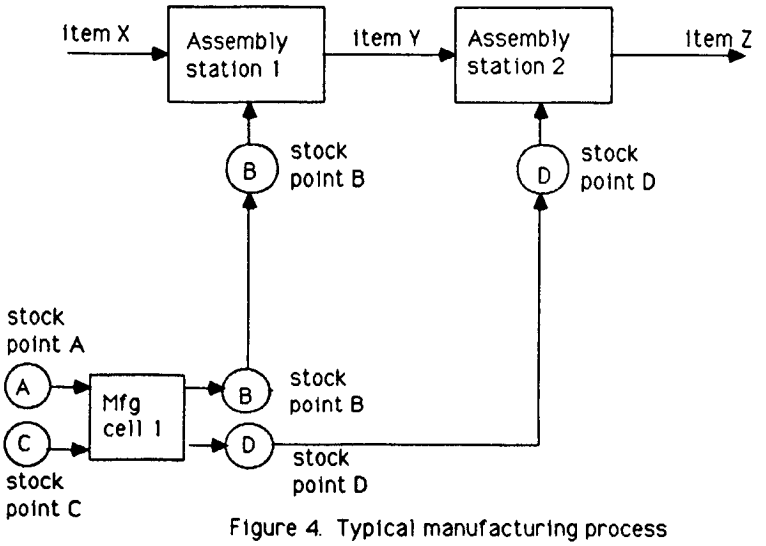


Figure 4. Typical manufacturing process

Figure 5 is a listing from the Symbolics 3620 screen showing a portion of the prompts from the simulation assistant and the corresponding user response. The screen consists of two windows. The right window contains various subwindows and options that the user selects via the mouse. Once the selection is made, the left window prompts the user for specific data that are used to construct a model of the process and to provide data for the simulation generators.

Several of the various subwindows are given in Figure 6. The first subwindow allows the user to select the type of distribution for the assembly time at a station. The current options are either exponential or normal. After the user selects the appropriate distribution, the user enters the mean time and standard deviation in the left window. The second subwindow allows the user to select the method of inventory, either push or pull. Once the method is selected, the user enters the amount of inventory at the various stock points.

CONCLUSIONS

In summary, a simulation assistant is an effective method of rapidly developing discrete event simulation models with minimum knowledge of the simulation language. An expanded set of simulation generators are necessary for modeling more complex processes. However, it is difficult to define these general purpose generators and to then link these generators to the simulation assistant. Considerable more work is required to make the simulation assistant more robust.

ORIGINAL PAGE IS
OF POOR QUALITY

```
Command: (sample-process)
Name of line: MAIN
Number of stations: 2
Mean arrival time: 100
Name of inventory part required at station 1: B
Mean assembly time: 100
Name of inventory part required at station 2: D
Mean assembly time: 100
Any more lines to create? (Y or N) No.
Maximum part B at assembly line [default 100]: 2
Maximum cart B at assembly line [default 100]: 2
Maximum part B at manufacturing cell [default 100]: 2
Maximum cart B at manufacturing cell [default 100]: 2
Mean manufacturing time: 10
Name of inventory transfer: FGIG1
Mean inventory transfer time: 2
Name of manufacturing cell: CELL1
Number of types of input items: 2
Name of input item: A
Name of input item: C
Maximum part D at assembly line [default 100]: 2
Maximum cart D at assembly line [default 100]: 2
Maximum part D at manufacturing cell [default 100]: 2
Maximum cart D at manufacturing cell [default 100]: 2
Mean manufacturing time: 10
Name of inventory transfer: FGIG1
Mean inventory transfer time: 2
Name of manufacturing cell: CELL1
Number of types of input items: 2
Name of input item: A
Name of input item: C
Maximum part A capacity at assembly line: [default 5000]: 100
Maximum part C capacity at assembly line: [default 5000]: 100
```

Figure 5. Simulation assistant display

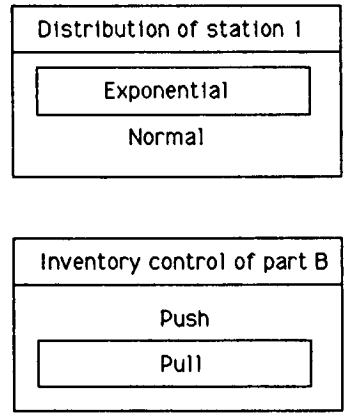


Figure 6. Various subwindows

REFERENCES

Ford, D. R. and B. J. Schroer. 1987. "An Expert Manufacturing Simulation System." Simulation, vol 48, no. 5.

Haddock, J. and Robert P. Davis. 1985. "Building a Simulation Generator for Manufacturing Cell Design and Control." Proceedings 1985 IIE Spring Conference, Los Angeles, CA.

Khoshnevis, B. and A. P. Chen. 1986. "An Expert Simulation Model Builder." Intelligent Simulation Environments, Society for Computer Simulation, vol 17, no. 1.

Mathewson, S. C. 1984. "The Application of Program Generator Software and Its extensions to Discrete Event Simulation Modeling." IIE Transactions, vol 16, no. 1.

Pegden, C. Dennis. 1985. Introduction to SIMAN. Systems Modeling Corp.

Schroer, B. J. and F. T. Tseng. 1987. "Modeling Complex Manufacturing Systems Using Simulation." Proceedings 1987 Winter Simulation Conference, December 1987, Atlanta, GA.

Tseng, F. T. and B. J. Schroer. 1987. "Use of Simulation Generators in Modeling Manufacturing Systems." Proceedings Southeastern Simulation Conference SESC 87, Society for Computer Simulation, October 1987, Huntsville, AL.

N88-16404

Knowledge-Based Simulation*

P.A. Newman
McDonnell Douglas Research Laboratories
McDonnell Douglas Corporation
St. Louis, MO

ABSTRACT

The complexity of space systems and increased costs for construction and operation make it necessary to both evaluate these systems in advance and to provide crew members with adequate advisory facilities in flight to determine the impact of changes in operational parameters. Currently, it is difficult to provide these facilities because of the need to weigh multiple objectives against one another when making system operation decisions.

Simulation has long been recognized as a useful tool for studying the behavior of complex systems which are conceptually difficult to understand and often mathematically intractable. Unfortunately, there are problems which limit its utility. Factors which influence its use include: the timeliness of results, cost, accuracy, lack of flexibility, and extensive programming requirements. However, combining simulation methodologies with A.I. techniques yields a practical approach to simulation known as knowledge-based simulation. The result is a tool which not only aids in model development but makes specific recommendations based on simulation analysis output.

This paper will describe an architecture for a knowledge-based simulator. The task of scheduling represents an area in which such a tool might be applied. More specifically, scheduling for crew and ground support activities for the shuttle and space station would benefit from the application of knowledge-based simulation. The knowledge-based simulator would allow the crew and support personnel to schedule and reschedule activities in a timely and flexible manner to examine and test possible plans.

INTRODUCTION

"Simulation is the process of designing a model of a real system and conducting experiments with this model for the purpose of either understanding the behavior of the system or of evaluating various strategies (within the limits imposed by a criterion or set of criteria) for the operation of the system" [6]. Systems which are characterized by complexity and task criticality require an accurate model of the system and its operation to allow the exploration and examination of system characteristics and to provide an opportunity for "fine-tuning" the proposed event set prior to execution. In addition, there must be adequate interaction facilities available during simulation execution to determine the impact of changes in important operational parameters. This is especially important when many interacting objectives must be considered in the decision-making process.

An approach to simulation is required that can accurately handle the complexity while retaining flexibility and timeliness. Combining simulation methodologies with knowledge-based techniques and object oriented approaches to modeling facilitates the automation of many of the manual functions associated with simulation; particularly modeling and analysis of results. An important attribute of this approach is the separation of data and procedures. Current simulation

* This work was supported by the McDonnell Douglas Independent Research and Development program.

PRECEDING PAGE BLANK NOT FILMED

languages embed data in procedures, thus limiting flexibility. The knowledge-based simulator is accessible interactively to allow the user to dynamically explore possibilities. Simulation becomes a more practical tool for "fine tuning" plans prior to implementation and performing "what if" exploration.

CONVENTIONAL SIMULATION

Simulation, in general, allows observation of a system's behavior through the controlled application of changes to a particular model. Simulators are tools that make it easier for humans to solve problems by providing information which describes the performance of a system under specific conditions. However, the results produced by current simulation techniques must be analyzed by the user. In this sense, conventional simulation systems are essentially descriptive rather than prescriptive tools that provide little insight or guidance to the inexperienced user. Their use can lead to inaccurate system assessments since the evaluation of what occurs in the simulation is the responsibility of the user.

The greatest advantage provided by simulation is that, by working with an idealized model of a system, it is possible to perform evaluations that could not be performed on the real system. Simulations are often used when an analytical solution either doesn't exist or is extremely complex; experimenting with the real system is too expensive, too time-consuming, too disruptive, or even destructive; or the actual system does not yet exist.

Current simulation techniques require the user to perform a manual set of steps. The user defines the model using a simulation language such as SLAMII or GPSS. It is necessary to verify the model prior to simulation. Analysis of the output is performed using operations research techniques and user experience. The user must then formulate conclusions which are justified by the analysis. The manual nature of these approaches results in a number of disadvantages. Constructing the initial model is often costly in terms of both people and computing requirements. Current techniques only address activities which have been explicitly specified in a model which may or may not accurately represent the system. This results in a serious lack of flexibility which is required for representing the dynamic systems. The accuracy and timeliness of both the input and the output data are often questionable when dynamic systems are modeled because of the time required for input and the difficulty in interpreting the output.

KNOWLEDGE-BASED SIMULATION

Knowledge-based techniques provide a basis for the development of a knowledge-based simulation tool to aid decision making in complex systems which cannot be adequately modeled or understood using mathematical techniques. Along with recommendations, knowledge-based simulation is able to provide the rationale to support a proposed action. The fact that the model knowledge, the operation logic, and the control components are separate provides modularity and facilitates the modification or addition of facts and rules which control the system. The use of a knowledge-based approach results in improved flexibility, improved user comprehension and greater accuracy and reliability of recommendations. The knowledge-based simulator has the ability to analyze developments over time, predict the future according to the assumptions implicit in the simulation model, analyze the results according to the rules and facts contained in the knowledge bases, and issue recommendations to the user.

Knowledge-based simulation applies the A.I. techniques of knowledge representation and reasoning to automate and expand the conventional simulation approach. These techniques expedite the development, verification, and modification of the simulation model by supporting direct interaction between the user and the simulation model. In addition, they can be combined with traditional operations research techniques to provide a facility which not only evaluates and

explains simulation output but which allows the user to explore alternatives from any point in the simulation. The knowledge-based approach to simulation uses an object-oriented representation of relationships between entities. It provides a more flexible model which is more amenable to change than are models programmed in more traditional simulation languages. [1,2,3,4,5]

ARCHITECTURE

When defining an architecture for a knowledge-based simulator, a number of areas must be addressed. These include the user interface, the model definition/editor, the simulation set-up, the simulation, the simulation analysis, intelligent exploration, and the associated knowledge bases. Figure 1 illustrates the basic components of a knowledge-based simulator.

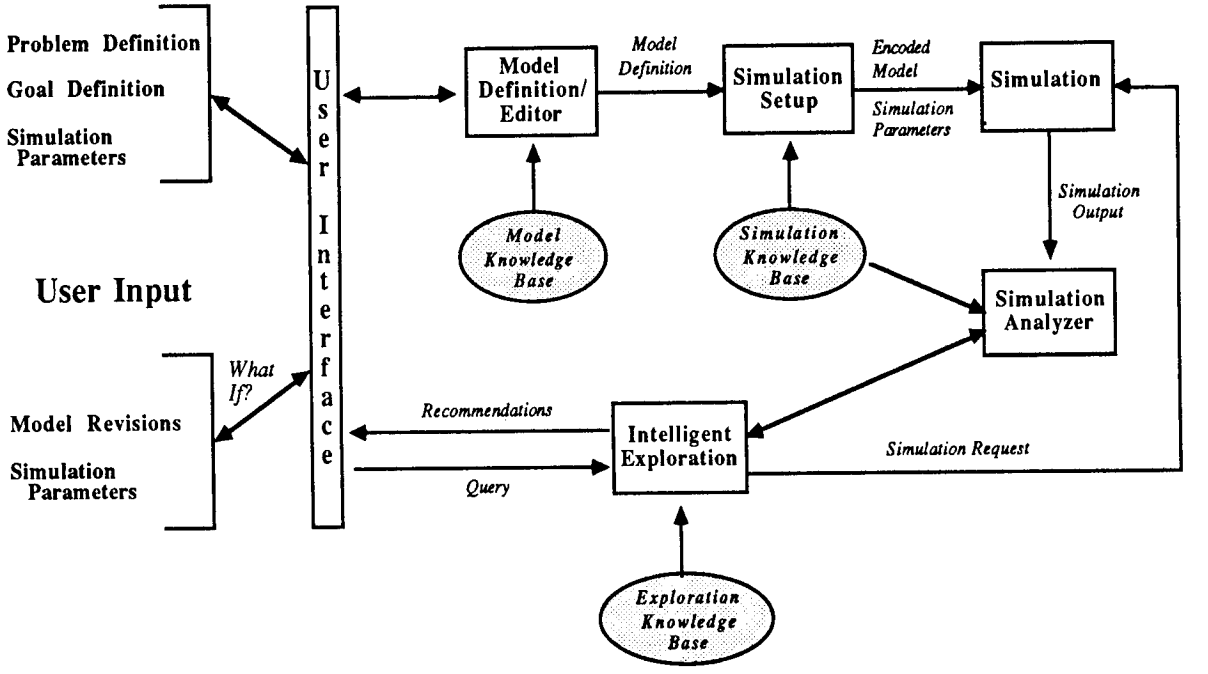


Figure 1. Architecture for a Knowledge-Based Simulator

User Interface

The User Interface is the user's link to the simulation environment. It must support the model building process, the output process, and the simulation evaluation process. When designing a user interface for any program, it is important to consider the user's background, the circumstances in which the program will be used, what inputs are required, what can be automated, what output information is important, and what form the information must take to be most easily understood.

Knowledge-based simulation specifically requires that the user be able to enter a problem description and goal statement as a set of constraints to the system. This input goes to the Model Definition/Editor which synthesizes a model for the simulation. The user also enters Simulation Parameters which include instructions for collecting simulation statistics.

The User Interface must handle the conclusions drawn from simulation analysis and produced by the Intelligent Exploration module. It must allow the user to query the Intelligent Exploration module for an explanation of these Recommendations. These queries may request that the simulation be modified and rerun starting at a specific point.

Model Definition/Editor

The model is synthesized from a combination of user input in the form of a Problem Definition and Goal Definition. The Model Definition/Editor facilitates revisions to the model by making it readily accessible to the user. It is also necessary for the model to be checked for validity, consistency and completeness prior to simulation. The latter is a difficult problem!

The Model Knowledge Base contains operational data such as system states and state transformation logic. This information specifies dynamic values, relationships among time dependent variables whose values change as a result of system operation, and operation procedures which move the system from state to state [1]. In addition, the Model Knowledge Base might include skeleton entities which can be instantiated to specifically identify attributes and behaviors which define the entity [3].

The model itself is composed of declarative objects and relations which match the user's concept of model organization. The model contains rules and facts about the system on both a global and local level. This knowledge defines system components, their attributes, and the environment. The environment definition includes organizational goals, system entities, entity relationships, and operational logic for dynamic interaction.

Simulation Set-up

The Simulation Set-up module takes the model and data collectors specified by the user and produces a set of instructions for the simulator. The output of this module initiates the simulation using the appropriate simulation parameters and instructions.

The setup and subsequent analysis access the Simulation Knowledge Base containing facts and rules pertaining to the simulation. This knowledge base contains procedures for performing the simulation and for evaluating results. A facility must be provided to allow the user to specify an existing procedure or to define new procedures [3]. This knowledge base should be extensible by the user.

Simulation

The simulation will initially be based on a discrete event approach. A clock will be advanced to the current time as significant events occur. Significant events are defined by the model. The simulation will be patterned after the object oriented techniques used in knowledge-based tools such as Simkit from Intellicorp, Inc.

Simulation Analysis

The data resulting from the simulation is analyzed according to a user specified goal. The user can indicate whether conventional operations research techniques or user defined methods should be used. An approach to this analysis begins with the representation of each goal as a set of constraints. Instruments for data collection provide input to the analysis. Following the simulation, each constraint is evaluated using a constraint utility index which could be created at the time the model is synthesized [3]. The effectiveness of a given event set can then be determined by considering the relative importance of each constraint in terms of the event set

being evaluated. The output of the analysis is sent to the Intelligent Exploration module for presentation to the user in the form of Recommendations based on the evaluation results.

Intelligent Exploration

A knowledge-based simulator should provide the user with recommendations for further improving simulation and analysis results. The user can then utilize this information to perform further simulations in an exploration or optimization mode.

Graphic interaction allows the user to select objects and explicitly edit their attributes and behaviors. The user should be able to query the states of visible objects, obtain graphical representations of relations among objects, and use the display to explain the behavior of the simulation wherever appropriate. This allows both intelligent explanation and intelligent exploration. Exploration allows the user to selectively modify a simulation, examine options from any point in the simulation, focus attention on selected aspects of the model, perform sensitivity analysis, and ask how particular results might be achieved [5]. To this end, the facility must be able to backtrack to a previous point in the simulation.

It is important that the user have confidence in the recommendations. Therefore, it must be possible to request further explanation. In fact, the primary task of explanation is to convince the user that a model is behaving reasonably and to show how the simulation arrived at a particular result. This can be done graphically or by means of a query.

CONCLUSIONS

Knowledge-based simulation will provide a flexible, interactive environment for the simulation of complex systems. The architecture described takes advantage of both conventional simulation approaches and knowledge-based techniques to define a knowledge-based simulator. This architecture can be extended to a variety of simulation applications.

REFERENCES

1. Bullers, W.I. Jr. and C.R. Schultz, *Production rule-based simulation for job shop scheduling*, **Proceedings of the 1986 AFIPS Conference**, 1986, pp. 718-723.

2. Moser, J., *Integration of artificial intelligence and simulation in a comprehensive decision-support system*, **Simulation**, Vol. 47, No. 6, December, 1986, pp. 223-229.

3. Reddy, Y.V. Ramana and M.S. Fox, *KBS: An Artificial Intelligence Approach to Flexible Simulation*, Technical Brief: Carnegie-Mellon University: The Robotics Institute, 1982.

4. Reddy, Y.V. Ramana, M.S. Fox, N. Husain, and M. McRoberts, *The Knowledge-based Simulation System*, **IEEE Software**, March, 1986, pp. 26-37.

5. Rothenberg, J., *Object-Oriented Simulation: Where Do We Go From Here?*, **Proceedings of the 1986 Winter Simulation Conference**, pp. 464-469.

6. Shannon, R. E., **Systems Simulation: The Art and Science**, Prentice-Hall, Englewood, N.J., 1975, p. 2.

N88-16405

RAPID PROTOTYPING AND AI PROGRAMMING ENVIRONMENTS APPLIED TO PAYLOAD MODELING

Richard S. Carnahan, Jr.
Andrew P. Mendler

Martin Marietta Astronautics Group
Box 179
Denver, CO 80201

ABSTRACT

This effort has focused on using Artificial Intelligence (AI) programming environments and rapid prototyping to aid in both space flight manned and unmanned payload simulation and training. Significant problems which have been addressed by this work are (1) the large amount of development time required to design and implement just one of these payload simulations and (2) the relative inflexibility of the resulting model to accept future modification. To date, results of this effort have suggested that both rapid prototyping and AI programming environments can significantly reduce development time and cost when applied to the domain of payload modeling for crew training and, while the focus of this work has been modeling/simulation development for training, the techniques employed are applicable to a variety of domains where models or simulations are required.

INTRODUCTION

This work was initiated as a response NASA Marshall Space Flight Center (MSFC) concerns regarding current modeling techniques applied to model/simulation development for Spacelab payload crew training and operations. Early discussions suggested that problems identified with current modeling processes would provide ideal test cases for the application of advanced software development techniques, focusing on the use of AI programming environments and rapid prototyping as capable of reducing the amount of time and cost required for model/simulation development, resulting in software systems which are easier to modify and extend. To date, results have suggested that rapid prototyping and AI programming environments show great potential to reduce development time and cost when applied to the domain of payload modeling for crew training. In addition, movement toward a greater level of system autonomy for both payload crew training and on-board payload operations will likely result in more efficient and cost effective Space Station payload operations.

PROBLEM AND OBJECTIVES

The current function of the Payload Crew Training Complex (PCTC) is to provide an integrated simulation of on-board Spacelab experiment operations for purposes of instruction in (1) procedures designed to bring the system to an operational or shut-down state as well as monitor its health and status while running, and (2) predefined fault handling command sequences. During an integrated simulation, the crew is required to respond to various scenarios for each of the experiments included in a particular mission; scenarios may include normal as well as faulted conditions. As the simulation proceeds, the simulation director, whose job it is to continually monitor the simulation while running, may dynamically modify a limited number of experiment parameters. It is important to realize that all crew procedures are rigidly defined and few, if any, deviations from those procedures are allowed. When following the defined command sequence does not bring the system to a nominal configuration, the crew requests assistance from ground support. Presently, there is little facility for allowing the crew to perform fault detection/isolation activities or reconfigure the system in the case of an anomoly; all these functions are the responsibility of the PI and other ground support personnel.

There are several problems with techniques traditionally used to develop models/simulations required for payload crew training: first, the process of modeling payload experiments and simulating associated procedures is slow and cumbersome. Second, due to several interacting factors, current models are extremely difficult to either modify or extend, and third, several different (and often incompatible) computers may be used to develop simulation/training models for different phases (e.g., design, flight operations), resulting in inconsistant software systems across the development life-cycle. In summary, the current simulation/model development process is inflexible and resistant to change. The notion of a simulation building/modeling environment which should be used across all life-cycle phases

PRECEDING PAGE BLANK NOT FILMED

is appealing, and presents an opportunity for examining the effectiveness of alternative/advanced software development techniques.

This effort examined the feasibility of applying rapid prototyping and AI programming environments to some of the problems outlined above. Particularly, it was important to assess how effective, in terms of reducing both cost and time incurred for model development, such techniques would be when used to model Spacelab payload experiments.

The objective of this activity has been to develop and evaluate a software/hardware environment for rapidly prototyping payload experiment simulations for training and operations. More specific technical objectives included using rapid prototyping techniques and AI programming environments to (1) model three Spacelab experiments (Wide Field Camera [WFC], Geophysical Fluid Flow Cell [GFFC] and Biorack) on the basis of documented experiment simulation modeling requirements, (2) model the functionality of a Dedicated Experiment Processor (DEP - a microprocessor provided by the experiment PI to increase the capability of the system), (3) simulate control/display panels for the GFFC and Biorack experiments, (4) simulate current payload crew-machine interfaces (both display and keyboard input) for the WFC and Biorack experiments, (5) design and implement prototype interactive crew-machine interfaces, (6) design and implement a prototype simulation director console for use in future payload simulation design and autonomous payload crew training and (7) examine the potential for integrating advanced software/hardware with existing PCTC software and hardware. It is important to understand that the techniques employed are applicable to a variety of domains where models or simulations are required. The remainder of this paper describes in more detail the approach taken and results obtained.

APPROACH

Hierarchical, object-oriented modeling/simulation is seen as an effective means of reducing inadequacies associated with current simulation techniques. Relationships between and among dynamic system components are often difficult to understand when traditional simulation methodology is used. Component behavior is difficult to trace since code which directly impacts behavior is spread throughout the simulation model and is not necessarily referenced to specific system components. For the same reasons, component interactions are even more difficult to comprehend. Assumptions about system performance are not made explicit but are masked by low-level mathematical/engineering information which can, again, be found throughout the model. In such cases, modifications to the model are difficult to make and resulting interactions are often untraceable [1].

The approach we have taken to model/simulation building is hierarchical and top-down; modeling is viewed as an iterative process in which the first model version represents a high level description of system functionality in terms of qualitative relationships between and among system components. The idea here is that the system may be abstractly described, and yet a 'working' model results. Indeed, general domain information related to the system under consideration may be used to set the framework for dynamic system component interactions and may also make reasoning about system performance more efficient (e.g., causal modeling, model-based reasoning). When tied to a graphic representation of the system, the model, at any level, can be tested to examine the accuracy of current system information. As knowledge about the system becomes more refined, the model can be easily modified or extended, at the appropriate level, to incorporate the new information. The complex and frequently untraceable interactions which often develop when modifications are made to traditionally developed models are minimized.

The combination of rapid prototyping and model hierarchies provides a more efficient process for model development, particularly in relation to training simulations. The design and implementation of models/simulations for training purposes need not, and indeed probably should not, be a separate activity from the design and implementation of models/simulations for operations or verification and validation. The approach employed here for model/simulation development uses rapid prototyping to facilitate both the requirements definition and implementation phases of hardware/software development as well as speed the process of code generation. Hierarchical model development results in using only those levels of model/simulation fidelity required by the training process. Higher fidelity versions of models/simulations can then be used for verification and validation or operations. In summary, only one software development effort occurs (two separate pieces of software are not required) resulting in one, common piece software system with different levels of abstraction. Hardware modifications which may impact simulations required for verification and validation or operations would only be incorporated in the training software if those modifications would affect the training process.

LISP AS A MODELING LANGUAGE - LISP appears to be an ideal choice for a rapid prototyping programming language and was the language chosen for this work. LISP's primary advantage lies in its ability to handle symbolic manipulation as well as numerical computation; an appropriate capability for the hierarchical modeling/rapid prototyping approach outlined above. Abstract system component functional relationships can be easily represented and the inherent flexibility of the language provides for rapid code extensions or modifications when more detailed system information becomes available. Another basic feature of LISP which supports the modeling approach we have taken is its ability to be run as either interpreted or compiled code. System modifications are rapidly accomplished since only the modified piece of code need be reevaluated. In addition, this facility allows for an incremental simulation capability in which simulations may be interrupted while running, parameters or other features modified/examined and the simulation restarted from that point; such a capability is highly useful both for system design as well as operations. Finally, in support of hybrid models, LISP allows function calls which access other programming languages; readily available low-level FORTRAN component models may be incorporated as part of the total system software.

PRODUCTION SYSTEMS AND MODELING - Production systems are useful for representing domain knowledge which can be stated as condition-action pairs and have been traditionally employed to encode an expert's knowledge about a particular domain for use in an expert system. However, their representation structure can also be applied to any modeling application where logic tables are used as part of the modeling scheme. Note that for these simulations, expert systems were not being created using production rules; the production system provided an efficient bookeeping technique, allowing for ease in system modifiability; potentially dangerous system interactions could be handled without great difficulty. Another benefit can be seen if one proposes the eventual incorporation of an expert system for purposes of experiment fault detection/isolation or intelligent training. Much of the system knowledge has already been encoded and extensions to the knowledge base are easily made.

Experiment modeling was completed using a three-level model hierarchy. Control passes from Level 1 (written in LISP), which has responsibility for overall program flow, to Level 2, the experiment logic (written in the production system language HAPS - Hierarchical Augmentable Production System) to Level 3, the math model (written in LISP). If a rule fires which requires the calculation of a parameter, the appropriate equation is called and the resulting value is passed to Level 1 or 2 depending upon the event. Modifications to the system can be made at any level without adversely affecting any other. It should be clear that neither production rules (representing logic tables) nor equations (representing low-level engineering information) were initially required to design and implement an abstract, high-level model of the experiment. All that was needed to begin was a general narrative statement outlining the overall system operational sequence; as necessary, logic tables and equations could be added later.

OBJECT-ORIENTED MODELING - The decision to use object-oriented programming as a primary technique for experiment modeling was grounded in (1) the type of application and (2) that part of the approach which required the design and implementation of dynamic, interactive graphics interfaces. Experiment system monitoring which requires procedural input from the crew appears to be ideally suited to the message-passing capabilities which object-oriented programming provides (flavors were used for this work). In general, the notion of representing a dynamic system as a group of objects (components) and their associated behaviors is intuitive and provides a structural framework from which inferences concerning system behavior can be made. Message-passing among objects emulates potential system interactions since a single message can initiate any number of complex behaviors or other messages. It is important to understand that objects need not be limited to hardware component representation, but can be used to mirror other features of total simulation/model implementation such as process control functions and their interactions. In addition, objects may be combined to form other objects, a facility which is particularly useful when representing complex, dynamic systems [3].

The crew interface for experiment system monitoring was designed using the Phoenix Graphics Editor (also flavors-based) which supports rapid prototyping of color, interactive graphics interfaces for a variety of applications. Using Phoenix, objects can be graphically displayed and their behaviors either externally or internally defined. Graphic objects can be made interactive so that the user can directly manipulate the color screen; when the user interacts with an object on the screen, interactive behaviors associated with that object are triggered. Objects provide a flexible and efficient means for prototyping user interfaces in dynamic systems. Interfaces for both the prototype simulation director console and experiments were designed and implemented in significantly less time than it would have taken without the benefit of object-oriented programming.

One other significant benefit of object-oriented programming, crucial to the ease and rapidity with which systems are developed and modified, needs to be mentioned: the notion of inheritance. Object hierarchies provide a natural analogy to the modeling approach used for the present work. Higher-level objects correspond to higher-level qualitative system information, whereas lower-level objects are more closely tied to lower-level quantitative information. Object hierarchies can be defined such that modifications made to behaviors or attributes of an object will not affect any superordinate object. On the other hand, object modifications will be inherited by any subordinate object (specific exceptions to this rule are possible). In addition, multiple inheritance (inherited information from multiple objects) is useful when it is important to view an object from different perspectives [2].

INTELLIGENT MAN-MACHINE INTERFACES FOR TRAINING - Although not the primary focus of the initial effort, part of the work accomplished to date has been the design and implementation of a prototype intelligent simulation director console. As outlined above, the present situation in the PCTC requires that the simulation director console be continually manned while a simulation is progressing. Such a mode of operation is becoming prohibitively expensive. In addition, Space Station requirements that training software be transportable will compel the development of software which is significantly more robust and capable of running more autonomously. The prototype Space Station simulation director console was implemented to allow some idea of the potential intelligent support might provide. The focus of the present prototype was in the area of simulation design, and rudimentary interface functionality includes the capability to load various experiment software systems, choose simulation scenarios based on the experiments loaded, change parameters for those experiments loaded, create experiment timelines for the simulation and dynamically add events to appropriate timelines by directly interacting with the screen.

The Phoenix Graphics Editor (developed by MMAG) was used to prototype initial versions of advanced crew interfaces for purposes of training and operations. Initial interface designs may be rapidly implemented (1-2 days) and are easily associated with an underlying system or procedural model. System operations or perturbations are reflected by visual changes in graphic component representations. The Phoenix system allows windows created using the editor to be combined with any kind of window available on the Symbolics machine, providing flexibility in overall interface design and functionality.

It would appear that the potential for applying intelligent crew interface support features is significant. In an operational state advice concerning procedures to be taken in a given system state might be made available to the crew. In a training scenario, advice would be more tutorial in nature. Other intelligent support features (e.g., automated procedural sequences, automatic report generation) could also be added. Future work will examine these important issues.

RESULTS AND CONCLUSIONS

The two major problems, the large amount of time required for model development and the resistance of finished models to modification, have both been successfully addressed. To date, comparisons of advanced and traditional modeling approaches suggest software development time and resulting cost are significantly less when the advanced approach is used. The WFC experiment model/simulation was completed in 3.0 man-months (NASA's estimate was 28.0 man-months), while undergoing several major modifications. Modeling the GFFC experiment, including the DEP and simulating the control/display panel, was completed in 1.5 man-months; it is estimated that the GFFC experiment represented a more difficult case than the WFC due to the inclusion of the DEP and the control/display panel. DEP modeling was not a significant issue since it could be subsumed in the overall control structure. Indeed, GFFC DEP functionality was limited by NASA to handling inputs from the control/display panel and processing a master fault condition. Modeling the Biorack experiment, including the simulation of all current DDUs (Data Display Unit) and control/display panels, was accomplished in 2.0 man-months. In terms of overall system functionality, models developed using the advanced approach have been externally validated and appear to evidence the fidelity expected from experiment models developed using present techniques.

Using Phoenix as a rapid prototyping tool for the design and implementation of interactive crew interfaces proved to be extremely beneficial. Current payload crew interfaces for the WFC and Biorack experiments, as well as a prototype interactive interface for the WFC experiment, were rapidly implemented (2 man-weeks). In addition, control/display panels for the GFFC and Biorack experiments were easily simulated (3 man-days). It is important to realize that rapid implementation of interfaces facilitates immediate testing of the experiment model under

development. Additions/modifications to the WFC, GFFC and Biorack models could be dynamically tested and feedback provided concerning model accuracy by visually inspecting system performance. The usefulness of Phoenix for person-machine interface design is apparent since modifications to any interface were easily made and their impact immediately assessed. Using HAPS to represent experiment logic tables provided an analogous rapid prototyping environment for the implementation and modification of certain model information. Also, the data represented by HAPS rules are the beginning of a knowledge base which most certainly would be required in the event an expert system for experiment fault detection/isolation and system reconfiguration or intelligent training was needed. As indicated above, using HAPS allows for rapid future model/system modification and enhancement.

Future work will focus on two major areas: (1) continued development of the simulation director console which will automate a majority of tasks manually performed by the current simulation director, and (2) integration of advanced software/hardware with the system configuration already in place at the MSFC PCTC. Both of the these areas are considered crucial to future Spacelab and Space Station payload crew training. For Spacelab, work has already begun which examines the feasibility of integrating advanced software/hardware with existing PCTC software/hardware. What is proposed is control, through our software/hardware, the training simulation in the PCTC while at the same time continuing to use present displays and hardware for purposes of crew training continuity. For Space Station, the development of autonomous training simulation control, the capability for intelligent support in simulation design and the design of more effective crew interfaces are all areas which need to be examined.

The work described in this paper suggests several conclusions: first, the use of rapid prototyping and AI programming environments is an efficient and cost effective means of accomplishing payload modeling for purposes of crew training. Hierarchical, top-down modeling in combination with object-oriented programming appears to provide a flexible structure for rapid model design, implementation, modification and enhancement. Using this approach implies that (1) high-level, abstract models could be used for training prior to the availability of higher-fidelity models, (2) lower-fidelity models may be all that are required for some training applications and (3) if needed, low-level engineering math models may be added later; additions which could provide in-flight operational software. Second, better crew interfaces would reduce crew training time by easing memory load and providing more efficient system monitoring capabilities. Intelligent crew interface support could provide needed advice in the event of an anomolous condition, in either training or operational settings, and could carry out more routine procedural sequences now manually accomplished. Third, there appears to be significant potential for intelligent support in training simulation design and operation, for both Spacelab and Space Station.

ACKNOWLEDGMENTS

The authors would like to thank Bert Dolerhie at MSFC for serving as the NASA technical liaison for our work. Bert provided everything we required in a timely fashion and gave us some excellent insight into the workings of the PCTC as well as very useful comments on an early draft of this paper. We would also like to express our appreciation to Harvey Golden at MSFC who frequently took time out of his busy schedule to meet with us to discuss programmatic features of the work. Finally, Salem Suleiman was instrumental in a large portion of the actual coding and used HAPS to great effect; his efforts are greatly appreciated.

REFERENCES

1. McArthur, D. J., Klahr, P., & Narain, S. (1986). ***ROSS: An Object-oriented Language for Constructing Simulations.*** In P. Klahr, & D. Waterman (Eds.), Expert Systems: Techniques, Tools, and Applications (pp. 70-91). Reading, MA:Addison-Wesley.

2. Stefik, M., & Bobrow, D. G. (1986). ***Object-oriented Programming: Themes and Variations.*** AI Magazine, 6(4), 40-62.

3. Weinreb, D., & Moon, D. (1980). ***Flavors: Message Passing in the Lisp Machine.*** AI Memo No. 602, MIT Artificial Intelligence Laboratory.

C-4

N88-16406

ORBITAL NAVIGATION, DOCKING, AND OBSTACLE AVOIDANCE AS A FORM OF THREE DIMENSIONAL MODEL-BASED IMAGE UNDERSTANDING

J. Beyer
C. Jacobus
B. Mitchell

ABSTRACT

Range imagery from a laser scanner developed at ERIM can be used to provide sufficient information for docking and obstacle avoidance procedures to be performed automatically. Three dimensional model-based computer vision algorithms in development at ERIM can perform these tasks even with targets which may not be cooperative (that is, objects without special targets or markers to provide unambiguous location points). Roll, Pitch, and Yaw of vehicle can be taken into account as image scanning takes place, so that these can be corrected when the image is converted from egocentric to world coordinates. Other attributes of the sensor, such as the registered reflectence and texture channels, provide additional data sources for algorithm robustness. Temporal fusion of sensor images can take place in the work coordinate domain, allowing for the building of complex maps in 3-space.

PRECEDING PAGE BLANK NOT FILMED

N88-16407

COMPUTING 3-D STRUCTURE OF RIGID OBJECTS USING STEREO AND MOTION

Thinh V. Nguyen, PhD.
MULTISIGNAL TECHNOLOGY CORPORATION
4662 Katella Ave., Suite H, Los Alamitos, CA 90720

ABSTRACT : *This paper presents our work performed as a step toward an intellegent automatic machine vision system for 3-D imaging. The problem considered here is the quantitative 3-D reconstruction of rigid objects. Motion and stereo are the two cues to be utilized in our system.*

The system basically consists of 3 processes: (i) the low-level process to extract image features, namely the corner points, (ii) the middle-level process to establish the correspondence in two modalities: stereo (spatial) and motion (temporal) and (iii) the high-level process to compute the 3-D coordinates of the corner points by integrating the spatial and temporal correspondences. Once the final correspondence is obtained, the corner points in the 3-D space can be determined easily as the intersection of two lines connecting the light sources to the corresponding points in the left and right views.

I. INTRODUCTION

Our problem is stated as follows:

Given 2-D images from a frame sequence of a moving rigid object taken by stereo cameras, perform a quantitative 3-D reconstruction on the object, i.e. determine the 3-D coordinates of the object control points.

The main goal of our research is to develop a strategy to combine information extracted from two independent sources: stereo and motion. Each source carries different cue on the object 3-D structure. Our work exploits the complementary nature of these two sources to reconstruct the object.

II. OUR APPROACH

Our approach is a hierarchical one. We break the solution into three distinct processes: low-level process, middle-level process and high-level process.

1) **Low level process:** *Feature extraction*

The low-level process segments the images into relevant features. These features will be used as the image descriptors for subsequent processing.

In the problem we are considering, there are images of rigid objects with regular geometrical shapes, i.e. shapes with well defined lines, curves and corners. The features to be extracted, therefore, should be related to these characteristics. One type of feature that is particularly useful for the processing of rigid objects is the corner point.

A corner point in an image is defined as a point which is an edge point and has significant change in the edge direction. Several researchers have proposed many

PRECEDING PAGE BLANK NOT FILMED

corner detector algorithms. After several experiments, we found that the Zuniga-Haralick [3] corner detector performed reasonably well for a variety of scenes.

2) **Middle level process:** *Correspondence*

The input to our correspondence process is the list of corner points detected in the segmentation (feature extraction) process. The spatial correspondence due to stereo will be carried out for two views (left and right) at each frame instant. The temporal correspondence due to motion will be carried out for two consecutive frames of the same view. Therefore, each correspondence process will involve four images grouped in four pairs. We solve the spatial correspondence by the epipolar line technique and the temporal correspondence by the relaxation matching method.

The epipolar line technique requires that the geometry of the two stereo cameras is known. For each point in the right image (the use of the right image is purely arbitrary), the equation of the epipolar line (in the left image) is computed. All points that lie in the neighborhood of this line are obtained as the spatially corresponding points.

The temporal correspondence is determined by a cooperative relaxation matching algorithm similar to the matching technique by Barnard and Thompson [1]. Our matching algorithm differs than that of Barnard and Thompson in two aspects: (i) the initial similarity measure is based on neighborhood interdistances and (ii) the matching is carried out in two directions (forward and reverse), only those points that are matched consistently in both directions are kept. This method eliminates the need of selecting proper probability threshold and also reduces false matches.

The details of these algorithms are described in [2].

3) **High level process:** *Integration*

The main task in the high level process is the resolution step. The resolution step uses the consistency principle. Correspondence must be consistent for both stereo and motion. In other words, if two points P_L and P_R of the left and righ image at frame i are two spatially corresponding points, then at frame j, Q_L and Q_R must also be spatially corresponding where Q_L and Q_R are the two temporally corresponding points of P_L and P_R respectively.

The spatial and temporal matchings will form loops. A loop is a closed matching sequence. Each loop will essentially pass through four points in the four images as illustrated in Figure 1. There are two kinds of loops:

. Shared loops: Loops which share same point(s).

. Single loops: Loops which do not share any points with any other loops

Conceptually, single loops are usually stable loops which represent correct matching sequences of all the four points in four images. There are cases, however, that single loops pass through incorrect corresponding points due to error in temporal correspondence or noisy conditions. We, therefore, propose the use of single loops only as a guide to search for correct loops, single or shared.

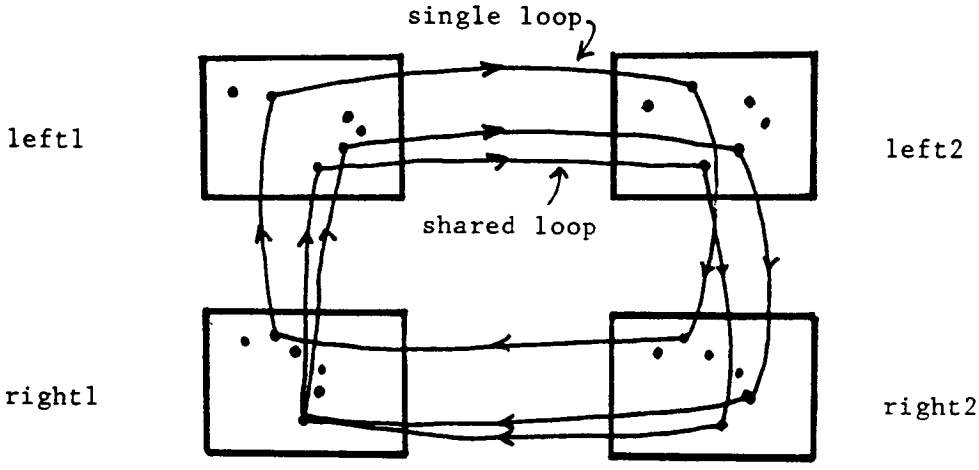


Figure 1 : Example to show some loops

Let : right1,left1 denote the two stereo views of the first frame.
right2,left2 denote the two stereo views of the second frame.
N denote the total number of points in the right1 view.

Our resolution algorithm then consists of the following steps :

Step 1: Determine single loops - For each point in the right1 image, find all closed loops going in the clockwise direction (the selection of clockwise direction is arbitrary). Select only single loops from these closed loops. Repeat for all N points in the right1 image. Let K denote the number of single loops.

Step 2: Determine initial object points - For each stereo pair in each single loop, compute the 3-D coordinates of the points in space. The result of this step is a set of 3-D coordinates of K points in frames 1 and 2. These K points will serve as a basis for further refinement.

Step 3: Find all possible matches - For each point in the right1 image, find all the possible matches in other images.

Step 4: Sort out potential candidates - From all possible loops for each point in the right1 image, select the best L loops (L is a small number relative to N) . The criterion for determining "best" loop is the *deviation from rigidity*. This deviation from rigidity is determined by computing the difference of the sums of distance errors from the two 3-D points in two consecutive frames. The distance is from the point to the K points found in step 2.

Step 5: Applying rules to select the best pairs - From the L best candidate pair found in step 4, we then apply the following heuristic rules to select our final best pairs because the smallest deviation of rigidity does not necessarily lead to correct loop due to potential errors and mismatches.

. Rule 1: If the smallest error is three times smaller than the second smallest error then the pair having the smallest error is the final best pair.

. Rule 2: If any single loop in step 2 is one of the candidate pairs, and if the relative difference between the corresponding error and the smallest error is less than 0.2, and if this corresponding error is less than a threshold then the single loop in question is the final best loop.

Step 3, 4 and 5 are repeated for all points in the right1 image. From these loops, we then remove redundant loops because some of these loops may be the same loop by keeping only the loops which have the first point in the right1 image the same as the corresponding starting point of that loop.

From the final best loops, the 3-D coordinates of the object point can be determined easily as the intersection of the two lines connecting the two focus points of cameras and the two stereo points on the image planes. Let (x,y,z) and (x',y',z') denote the coordinates of two temporally corresponding points , the object motion can then be modeled as an affine transform . The affine transformation is a mapping to transform the three coordinates (x,y,z) to a new set of coordinates (x',y',z'). The overall effect of rotations and translations results in the following equations :

$$\begin{aligned} x' &= a_0x + a_1y + a_2z + a_3 \\ y' &= b_0x + b_1y + b_2z + b_3 \\ z' &= c_0x + c_1y + c_2z + c_3 \end{aligned} \qquad (1)$$

When the number of object control points is sufficiently large (greater than 5 for example), we can use the computed coordinates to estimate the a_i,b_i parameters ($i = 0,1,2,3$) in the equations (1) using the least squares method. These parameters can then be used to predict object coordinates so that other tasks (e.g. tracking) can be performed.

III. RESULTS

We used simulated images to test our technique. The simulated images were random dots in space. We arbitrarily placed these dots on the surface of an ellipsoid. For the results reported here the lengths of the ellipsoidal axes are 500 mm, 400 mm and 500 mm in the x,y, and z directions respectively.

The computed 3-D coordinates are then compared to the true 3-D coordinates. The error is computed as the distance between the computed point and the true point. We counted the number of points that have errors in 3 groups. Group 1 consists of points that have small error (from 0 to 30 mm) . Group 2 consists of points that have medium error (from 30 mm to 75 mm). Group 3 consists of points that have large error (greater than 75 mm). The number of generated points is 10. Uniform random noises are added to displace these points. These points are also randomly deleted in both image planes. For 10 runs, the total number of points in both frames is 200.

The camera focus length is 16 mm. The two cameras are displaced by 500 mm, 0 mm and 50 mm in the x,y and z directions respectively. The right camera is rotated 10^o in the y direction . The object is placed at a distance of 2800 mm in the z direction . Motion parameters of the object are : translation (-50 mm, 0 mm, 50 mm) , rotation (5^o ,5^o , 5^o) in the x,y and z directions respectively.

The result of one typical simulation is shown in Table 1.

No. of deletions	Noise strength (pixels)	Group1	Group 2	Group 3
(0,0)	0	131	0	1
(0,0)	1	131	0	1
(0,0)	2	103	26	3
(0,0)	3	71	36	16
(0,0)	4	52	38	26
(0,0)	5	37	53	32
(1,1)	0	118	0	1
(1,1)	1	118	0	1
(1,1)	2	95	18	2
(1,1)	3	64	31	13
(1,1)	4	45	35	12
(1,1)	5	38	41	20
(2,2)	0	103	1	2
(2,2)	1	103	1	2
(2,2)	2	82	16	4
(2,2)	3	60	35	10
(2,2)	4	34	43	17
(2,2)	5	29	36	22

Table 1: Number of computed points in 3 groups.

IV. CONCLUSION

This paper describes our work in determining the 3-D structure of rigid objects using stereo and motion. The image features are the corner points. Correspondences of image features are carried out for stereo (spatial correspondence) and motion (temporal correspondence) using two independent methods. The two types of correspondence are then integrated to produce the final correspondence which provides the 3-D coordinates of the image features. Our integration technique exploits the rigidity constraint and heuristic rules. Results obtained by simulation show that our technique works reasonably well even under several noise sources.

ACKNOWLEDGEMENTS

The work described in this paper is supported by NASA Marshall Space Flight Center under Contract No. NAS8-37308. The guidance and support of NASA personnel, in particular, Mr. Glenn Craig, are sincerely acknowledged.

REFERENCES

[1] Barnard, S.T. and Thompson, W.B., "Disparity Analysis of Images," *IEEE Trans. on Patt. Anal. and Mach. Intell.*, Vol. PAMI-2, No. 4, July 1980, pp. 333-340.

[2] Nguyen, T.V., "Computing Range and 3-D Structure of Rigid Objects using Stereo and Motion," MULTISIGNAL TECHNOLOGY CORPORATION, *Technical Report TR-87.001*, July 1987.

[3] Zuniga, O.A. and Haralick, R.M., " Corner Detection Using the Facet Model, *Proc. IEEE Computer Society - Computer Vision and Image Processing*, 1983, pp. 30-37.

N88-16408

REAL TIME AI EXPERT SYSTEM FOR ROBOTIC APPLICATIONS

John F. Follin

ABSTRACT

GA Technologies has developed and constructed a computer controlled multi-robot process cell to demonstrate advanced technologies for the demilitarization of obsolete chemical munitions. This cell contains two robots, an advanced machine vision system, and a variety of sensors (force, range finding, and tactile). Autonomous operation of the cell under computer control has been previously demonstrated and reported.

Although expert systems are not new to the artificial intelligence world, systems that are easy to use (programmers and operators), work within a process control system, control multi-robots and vision systems in real time, and are very flexible and hard to come by. Here at GA we have developed an expert system based in Data General's Common Lisp. The purpose of this system is to demonstrate that once the expert system is operating with rules the system can carry out operations that have not been preprogrammed. These operations, or goals, can be introduced into the system and the artificial intelligence software will solve the goals and generate a solution or solutions. The system will execute these solutions using a variety of hardware equipment. The presentation will discuss the development and operation of our expert system.

GA has, using internal funding, incorporated an Artificial Intelligence processor to direct the control of the process cell. The system uses an Expert System that was developed in a Common Lisp environment which can solve a variety of problems using a rule based system. Rules and goals for various processes to be demonstrated were input to the system and control of the robotics cell through Artificial Intelligence was achieved. Any rules that were modified or created during the solving of system goals were stored for later recall; in efect, the system can learn new rules. The expert system is interfaced through touch screens, voice recognition, and voice synthesizers for easier man-machine interfacing. A special feature of interest is the use of sophisticated computer graphics for LISP system development, testing and execution monitoring.

This presentation describes the methods through which the vision system and other sensory inputs were used by the AI processing system to provide the

PRECEDING PAGE BLANK NOT FILMED

information required to direct the robots to complete the desired task. The presentation discusses the mechanisms that the expert system uses to solve problems (goals), the different rule data base, and the methods for adapting this control system to any device which can be controlled or programmed through a high level computer interface.

Various applications and system demonstrations (some pertaining to space) have been performed using the above equipment and will be discussed.

N88-16409

SOLID MODELLING FOR THE MANIPULATIVE ROBOT ARM (POWER) AND ADAPTIVE VISION CONTROL FOR SPACE STATION MISSIONS*

V. Harrand
A. Choudry
Center for Applied Optics
University of Alabama in Huntsville
Huntsville, AL 35899

ABSTRACT

We have studied the structure of a flexible arm derived from concatenation of the Stewart-Table-based links. Solid modelling provides not only a realistic simulation but is also essential for studying vision algorithms. These algorithms could be used for the adaptive control of the Arm, using the well known algorithms such as shape from shading, edge detection, orientation, etc. Details of solid modelling and its relation to vision based adaptive control will be discussed.

*Work supported in part by NASA Grant #NAGW-847 and State of Alabama Research Council.

N88-16410

ITERATIVE-DEEPENING HEURISTIC SEARCH FOR OPTIMAL AND SEMI-OPTIMAL RESOURCE ALLOCATION

Susan M. Bridges
James D. Johannnes
Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

This paper examines the use of iterative-deepening A* (IDA*) in solving a certain class of resource allocation problems. IDA* stores a number of nodes proportional to the depth of the solution in the search tree rather than the number of nodes generated as in A*. A* and IDA* solutions to the resource allocation problem were implemented with empirical results showing that the low effective branching factor associated with real-valued cost estimates causes an unacceptably large number of nodes to be generated by IDA*. A modification of IDA* that can be used for semi-optimization, called IDA*$_\varepsilon$, was developed and is shown to be an ε-admissible tree search algorithm. A comparison of the performance of A*, IDA*, and IDA*$_\varepsilon$ for resource allocation demonstrates the effectiveness of IDA*$_\varepsilon$ in reducing the computational requirements of the problem.

1. INTRODUCTION

Mission planning for space applications must address many resource allocation problems of various types. This paper will examine a class of resource allocation problems in which resources have a given effectiveness in reducing the value of tasks and more than one resource may be used for a particular task. The problem is to find an optimal allocation of resources to tasks that maximizes gain, i.e. the total reduction in task value minus the cost of the resources allocated.

Slagle and Hamburger [6] report the use of the A* heuristic search algorithm in solving this resource allocation problem. In recent work by Korf [1,2], a variation on A*, called iterative-deepening A* (abbreviated IDA*) has been shown to be optimal in terms of space and time requirements among heuristic best-first tree searches. This paper reports the investigation of IDA* for solving this type of problem. Difficulties that arise when the cost function used with IDA* is real-valued rather than integer are discussed, and the development of modifications of IDA* that can be used for semi-optimization is reported.

2. BACKGROUND

The following description of this type of non-linear resource allocation problem is modeled after that of Slagle and Hamburger [6]. The problem is to assign a set of resources to a set of tasks where each resouce has a given cost and each task a given value. Additionally, for each resource-task pair, an effectiveness value in the range 0.0

PRECEDING PAGE BLANK NOT FILMED

to 1.0 is given that represents the expected portion by which the task value will be reduced if the resource is assigned to the task. For each resource, there are n + 1 choices for assignment where n is the number of tasks, i.e. the resource may be assigned to any of the n tasks or not used at all. The choice of not using a resource is assumed to have a gain of 0 and so would be chosen over assignments resulting in a negative gain. Given a choice between not using a particular resource or using a resource-task assignment with a gain of 0, the choice of not using the resource will be chosen. Several resources can be assigned to the same task with the assumption that they do not interact.

3. A* VERSUS IDA*

Slagle and Hamburger [6] describe the use of the A* algorithm to find an optimal plan for assignment of resources to tasks. A* has been shown to always yield an optimal solution when the heuristic cost function used is consistently "optimistic" [3], i.e. for minimization problems it always underestimates the cost of the solution and for maximization problems it always overestimates the value of the solution. Heuristics with this property are called "admissible" . In general, the A* algorithm has storage requirements proportional to the number of nodes that will be expanded. Korf [1,2] has shown that a modification of A*, depth-first iterative-deepening A* (IDA*), has space requirements proportional to the depth of the solution node in the search tree. In addition, IDA* always finds an optimal solution in a manner similar to A* and asymtotically expands the same number of nodes as A*. In the section below, we describe the implementations and results of experiments comparing the efficiency of the A* and IDA* algorithms for resource allocation.

For both algorithms, the search tree is organized such that each level in the tree represents the allocation of a particular resource. Thus, the depth of the tree, d, is equal to the number of resources, and the branching factor of the tree is (n + 1) giving a total of $(n + 1)^d$ nodes in the complete search tree. Each node in the tree represents a partial plan for allocation of resources with the root node representing the plan of not using any of the resources. The evaluation function for each node is the sum of the gain (g) achieved by the partial plan, and an estimate of the gain achievable by allocation of the remaining resouces (h). The heuristic used for the calculation of h in this implementation is similar to one of the heuristics described by Slagle and Hamburger [6]. For each resource remaining to be allocated, the maximum gain achievable by that resource with the task values of the current node is calculated. Interaction among resources is ignored. The value of h is the sum of these maximum gains or the sum of the current task values, whichever is lower. Clearly, the cost function is both monotone and admissible since the gain calculated is always optimistic and becomes more accurate (lower) as more resources are added to the partial plan.

In the A* solution to this problem, a priority queue (OPEN list) is maintained that contains all nodes that have been generated but not expanded. These nodes are ordered in descending order by estimated gain. Nodes are successively removed from the queue and expanded until a solution node is found (a node where the actual gain is equal to the estimated gain). This problem is unusual in that every node in the search

tree represents a solution (an allocation of resources to tasks), so any child node that has an estimated gain less than an actual gain already found can be pruned and never reconsidered.

The IDA* algorithm does not use an OPEN list. Instead, a threshhold value is maintained for each iteration and any node that has an estimated gain less than the threshhold value is not generated on that iteration. The initial value for the threshhold is the gain estimate for the root node. A simple recursive depth-first search is done of all nodes with gain estimates greater than or equal to the cutoff. In addition, the value of the highest rejected node is recorded. The search terminates when a node is found with an actual gain equal to its estimated gain. If a solution has not been found at the conclusion of the depth-first search with a given cutoff, the value of the threshhold is changed to the highest rejected value and the search is done again. The iteration process is repeated until a solution is found and the solution found is guaranteed to be optimal since all nodes with higher possible gains will have been examined on previous iterations.

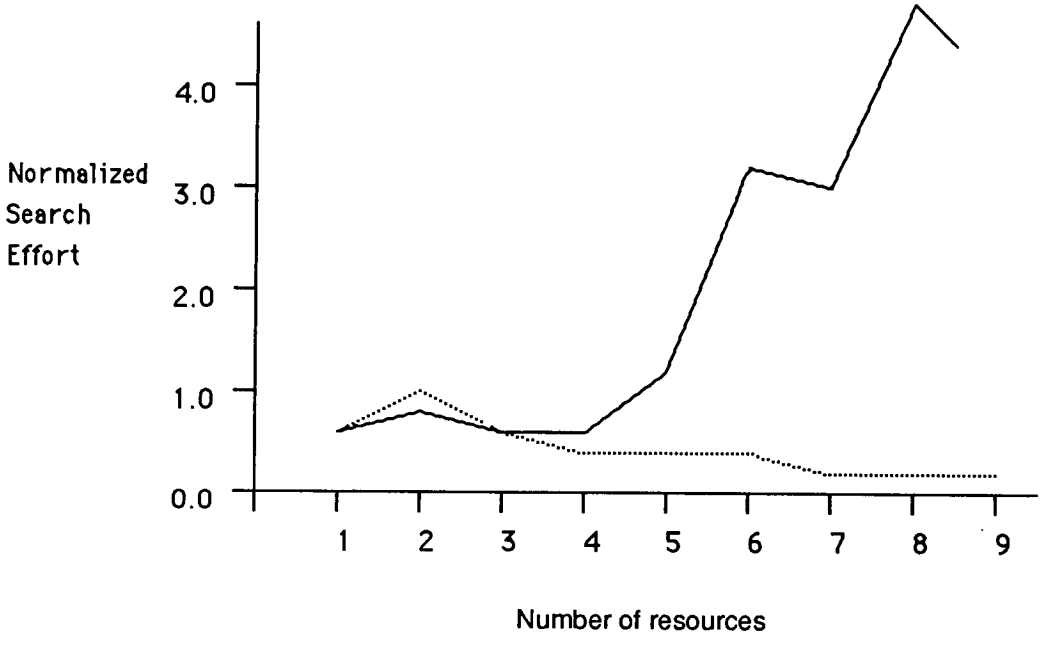


Figure 1. Performance comparison of A*, IDA*, and IDA*$_\varepsilon$ on the same data sets. In this experiment, the number of resources was equal to the number of tasks. Task values, resource costs, and effectiveness values were randomly chosen from uniform distributions. The solid line represents the mean ratio of the number of nodes expanded using IDA* compared to A*. The dotted line represents the mean ratio of the number of nodes expanded using IDA*$_\varepsilon$ compared to A*.

The solid curve in Figure 1 shows the results of an experiment comparing the use of A* and IDA* on the same randomly generated data sets. The curve represents the mean ratio of the number of nodes expanded using IDA* to that expanded using A*. Although IDA* always stores far fewer nodes than A*, the number of nodes generated by IDA* grows at a much faster rate than for A*. Both algorithms quickly overwhelm

computational resources with A* consuming all available space and IDA* requiring an unreasonable amount of time for relatively small size problems. Other experiments have shown that some improvement in performance is achieved by considering the resources in order of maximum possible gain, but the improvement is not sufficient to avoid the computational limits encountered.

Korf [2] has shown that for tree search problems with a branching factor, b, of magnitude greater than 1, IDA* opens asymtotically the same number of nodes as A*. The order of the algorithm is b^d, where d is the number of iterations and b is the ratio of the number of nodes opened during the current iteration to the the number of nodes opened on the previous iteration. The constant coeffficient is $(1 - 1/b)^{-2}$ as the search depth goes to infinity. Korf [1] pointed out that for branching factors close to 1, the constant coefficient approaches infinity as the search depth goes to infinity. Unfortunately, in our problem and, in fact, in many domains with real-valued evaluation functions, the branching factor is often close to 1 with the algorithm opening only one additional node on each iteration. The semi-optimization algorithm described in the next section was developed in an attempt to find a modification of IDA* that has better performance characteristics for problems with real-valued cost functions.

4. SEMI-OPTIMIZATION WITH IDA*

Pearl [4] describes several speedup versions of A* (called A^*_ε) that can be used for semi-optimization. Algorithms that guarantee that the cost of the solution (for minimization problems) will not exceed the cost of an optimal solution by a factor of more than $1+\varepsilon$ are called ε-admissible. We describe an ε-admissible modification of IDA* (IDA^*_ε) which exhibits the same characteristics of increased performance as ε-admissible versions of A* but with much reduced storage requirements. For purposes of this discussion, we will describe the algorithm as used for minimization of cost problems as is traditional for A*.

IDA^*_ε works much like IDA* except that the threshhold for the initial iteration is set to $1+\varepsilon$ times the cost of the root node and on successive iterations it is set to $1+\varepsilon$ times the cost of the lowest rejected node from the previous iteration. As with IDA*, on each iteration a simple recursive depth-first search is done of all nodes with cost estimates less than or equal to the threshhold. Note that the storage requirements for IDA^*_ε are proportional to the depth of the solution in the search tree and are handled automatically via the runtime stack. The search terminates when a goal node is chosen for expansion. A relatively straight-forward modification of Pearl's proof for the ε-admissibility of A* can be used to prove the ε-admissibility of IDA^*_ε.

The only modification necessary to convert the IDA* resource allocation program to

IDA^*_ε was to change the calculation of the threshhold. As guaranteed by the algorithm, the IDA^*_ε program found allocation plans with gains within 10%(ε) of the optimal gains for plans found by A* and IDA*. Since the number of iterations of IDA^*_ε can never exceed $1/\varepsilon$, this algorithm avoids the explosive growth in the number of nodes generated that occurs with IDA*. Figure 1 shows a comparison of the mean number of nodes generated by IDA* and IDA^*_ε with both normalized to A*. IDA^*_ε also opens significantly fewer nodes than A*. This indicates that A* and IDA* spend a great deal of time discriminating among solutions of approximately the same value. Whereas A* has storage requirements proportional to the number of nodes generated, IDA* and IDA^*_ε both have storage requirements proportional to the depth of the search tree (the number of resources in this case). Thus, IDA^*_ε uses far less time than IDA* and far less space than A* while guaranteeing a solution with a cost within a factor of $1 + \varepsilon$ of the optimal solutions found by these two algorithms. This reduction in time and space complexity allowed the IDA^*_ε program to find semi-optimal solutions to problems that could not be solved using A* or IDA* within the time and space limits imposed.

5. CONCLUSIONS

Resource allocation problems of many types will continue to be of vital importance in mission planning. We have demonstrated that when IDA* is applied to one type of resource allocation problem, it uses far less storage than A* but opens far more nodes and thus has an unacceptable time complexity. This is shown to be due, at least in part, to the low-valued effective branching factor that is a characteristic of problems with real-valued cost functions. The semi-optimal, ε-admissible IDA^*_ε search algorithm that we described was shown to open fewer nodes than both A* and IDA* with storage complexity proportional to the depth of the search tree.

6. REFERENCES

1. Korf, R. E. Depth-first iterative-deepening: an optimal admissible tree search. *Artificial Intelligence,* 27, 1985, pp. 97-109.
2. Korf, R. E. Iterative-deepening-A*: an optimal admissible tree search. *Proceedings Ninth International Joint Conference on Artificial Intelligence,* Los Angeles, CA, 1985, pp.1034-1035.
3. Nilsson, N. J. *Problem solving Methods in Artificial Intelligence.* McGraw-Hill, New York, 1971.
4. Pearl, J. *Heuristics,* Addison Wesley, Reading Mass. 1985.
5. Pearl, J. and J. H. Kim. Studies in semi-admissible heuristics, *IEEE Transactions on Pattern Analysis and Machine Intelligence,* 4, 1982 , pp. 392-399.
6. Slagle, J. R., and H. Hamburger. An expert system for a resource allocation problem. *Communications of the ACM,* 28, 1985, pp. 994-1004.

N88-16411

Similarity Networks as a Knowledge Representation for Space Applications

David Bailey, Donna Thompson and Jerald Feinstein
ICF/Phase Linear Systems, Incorporated
9300 Lee Highway
Fairfax, VA 22031-1207
(703) 934-3800

ABSTRACT: Similarity networks are a powerful form of knowledge representation that are useful for many artificial intelligence applications. Similarity networks are used in applications ranging from information analysis and case-based reasoning to machine-learning and linking symbolic to neural processing. Strengths of similarity networks include simple construction, intuitive object storage, and flexible retrieval techniques that facilitate inferencing; therefore similarity networks provide great potential for space applications.

INTRODUCTION

Space exploration depends upon computers to aid in such tasks as navigational control, mechanical and electrical systems monitoring, and flight tracking. As equipment used in space becomes more complex, the role of computers becomes vital in the areas of design, monitoring, control, and maintenance. To keep the pace with this complexity, computer hardware has developed faster processers using RISC architectures and parallel processing. Computer software must now become more intelligent as well as more abundant. Intelligent software is needed to enhance the capabilities of limited personnel, whether they be crew members or design teams. Potential areas for increased use of intelligent software include system design, decision support, simulation, and information retrieval.

The intelligent software necessary to facilitate the various tasks mentioned above utilizes artificial intelligence (AI) techniques. Artificial intelligence applications are based upon some type of mapping between concepts in the physical world and abstract software datatypes. This mapping is known as knowledge representation. Choosing the proper knowledge representation is vital to the success of an artificial intelligence application. A good knowledge representation has the following properties:

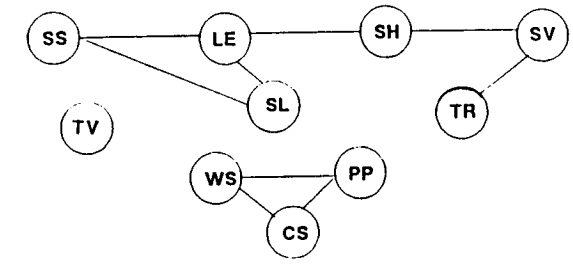
- Makes important things explicit,
- Exposes constraints,
- Is complete and concise, and
- Is easy to use.

Similarity networks are a powerful form of knowledge representation that are well-suited to many artificial intelligence applications. This is especially true in space applications due to the ill-defined nature of search spaces and formerly intractable problems facing aerospace and astronautics engineers. Created to assist machine learning programs, similarity networks may also be used to analyze information, reason from experience, and support various other AI techniques.

PRECEDING PAGE BLANK NOT FILMED

SIMILARITY NETWORKS DEFINED

Similarity networks are a knowledge representation technique that stores and links objects based upon their similarity to each other. Similarity networks are composed of clusters of objects that are connected via weighted links. As the networks become more complex, hierarchies of clusters and networks are formed. The objects represent the physical or abstract concepts that are being stored in the network. An object may be as simple as a letter of the alphabet or as complex as an electrical circuit design. The weighted links connect any two objects and designate the degree of similarity between the objects.



SV = Saturn V Rocket
SH = Space Shuttle
SL = Space Lab
LE = LEM
SS = Space Station
TR = Triton
WS = Weather Satellite
PP = Planetary Probe
CS = Comm. Satellite
TV = Terrain Vehicle

Figure 1.

An example of a similarity network, shown in Figure 1, describes the relationships between and among various classes of space vehicles. Notice that functionality characteristics such as propulsion and data gathering as well as physical attributes such as dimensions and mass help to form the clusters within this network. For example, a planetary probe is linked to a weather satellite in part because they form similar exploratory tasks.

Similarity networks were first described in Patrick Winston's thesis "Learning Structural Descriptions from Examples" [Winston 75]. They are later mentioned in Minsky's paper "A Framework for Representing Knowledge" [Winston 75]. The first implementation of a similarity network was the result of thesis work done by David Bailey [Bailey 86]. Mr. Bailey experimented in methods of constructing, searching, representing, and evaluating similarity networks. Using this experience, ICF/Phase Linear Systems has been researching the use of similarity networks in artificial intelligence for a broad range of applications in industry, government, the military, and in space.

CONSTRUCTING A SIMILARITY NETWORK

The first step in constructing a similarity network is choosing and describing the objects to be stored. Objects are chosen based upon the type of application to be built. For example, to build software that reasons from experience, descriptions of situations and outcomes are used to build the network. As a second example, if a tool identification

program is desired, descriptions of various types of tools and their purposes need to be stored.

The physical, conceptual, or abstract objects to be stored in the network must then be described. This may be accomplished with object-attribute-value triplets, property lists, natural language texts, or other knowledge representations depending upon the type of matcher used to compare the objects.

The matcher compares each object to all of the others and returns the number of features in common and the number of features unique to each object. These numbers are then put into an equation that determines the similarity score for the objects. If the score exceeds the minimum threshold established by the system designer, the objects are linked in the resulting similarity network.

Several choices are made in constructing a similarity network, including the proper similarity equation to use, the weights to use within the equation, the description of the objects, and the threshold at which links are to be formed. The quality of the network can be determined using heuristics that examine the clusters within a network, as well as the intuition of the network developer. An advantage of the similarity network approach is that it has a built-in form of sensitivity analysis for a final evaluation of the network.

An iterative construction process produces many views of the network. This ensures that specific knowledge representation requirements and objectives are met. In addition, new relationships between the objects that were previously undiscovered now emerge as the network construction parameters are varied. By creating an awareness of new relationships, similarity networks provide invaluable assistance in exploring complex, qualitative, and ill-defined problem spaces.

SIMILARITY NETWORKS IN AI APPLICATIONS

Similarity networks are an effective knowledge representation for many AI applications. Two types of applications particularly well suited to similarity networks are information analysis and case-based reasoning.

Information Analysis

Information analysis applications process new data or take a fresh look at existing data. Similarity networks facilitate several information analysis applications including:

- Object identification
- Resource substitution
- Perspective changing
- Knowledge acquisition.

Each of these applications is described in more detail in the paragraphs below. To illustrate some practical uses of these applications, an example from a spacecraft electrical systems design scenario will accompany each description.

An object identification application takes as input the description of an object and produces as output the name or category of that object. Identification applications search the similarity network for an object that matches the input description. If there is not an exact match the system finds the closest match. This results in three classifications, the most similar object, the category (or prototypical member of the category), and the match score. The match score provides a measure of quality - or similarity - for the object returned. As an example, a spacecraft electrical systems design assistant with a built-in identification application might be used in conjunction with a visual scanner to search for sodder bridges or other problems within a circuit board in a control panel or other instrumentation.

Similarity networks also provide a good knowledge representation for resource substitution application. A resource substitution application increases efficiency by promoting the creative use of materials. For example, if wire wrap were needed to connect two components but was unavailable, a resource substitution application might recommend the use of a sodder bridge based upon the similarity between the two object's functionality. In the same way, unobvious substitutions can be made. Again from our electrical systems design example, an unobvious substitution might be for the application to suggest using the heat sink of a neighboring electrical system to be a heat source of the environmental system. This is also an example of a change in perspective which is discussed next.

A change in perspective provides a different look at the objects in the similarity network. The change in perspective allows different properties of network objects to be ranked as more important in certain situations. In the heat sink to heat source example above, the heat sink is viewed as a resource and not as a waste. This provides an efficient solution to the problem of supplying heat.

The final information analysis example is knowledge acquisition. Given situations as objects, certain types of induction may be used to produce rules from recurring situations. This technique may be used either on previously acquired information or dynamically in conjunction with an expert system that learns as it goes. From the electrical systems design example, recurring use of capacitors to act as surge protectors might prompt the system to form a rule that "if a surge protector is needed, then use capacitors".

Case-based Reasoning Applications

The second major category of application utilizing similarity networks as a form of knowledge representation is case-based reasoning or reasoning by example. A case-based reasoner performs knowledge-based functions somewhat like those of an expert system. For example, a case-based reasoner built upon a similarity network might perform the task of control monitoring in a life support system.

A case-based reasoner would operate similar to the identification application described above using situations as objects. An exact match

would produce the outcome stored with the situation in the similarity network. This outcome would tell the operator what to expect in the given situation. Inexact matches produce results qualified by their similarity scores. The case-based reasoner could also be used to hypothesize on speculative information. If systems operators wanted to determine how high a reading could climb before approaching a dangerous level, they could enter potential readings to monitor the reasoner's reactions.

Case-based reasoners have several advantages over production rule forms of expert systems. First, case-based reasoners are easy to build. Sample situations and outcomes are entered directly into the similarity network. Knowledge engineers are not required to supervise the acquisition of information. Second, the initial information is retained by the reasoner, making it possible to return to the initial data to test assumptions. Finally, the case-based reasoner - using the match score - knows when it does not have an appropriate solution.

Similarity networks provide an effective knowledge representation for other types of artificial intelligence applications such as machine learning, analogical reasoning, classification, and machine vision. More traditional forms of computing that require information storage and retrieval may also benefit from the power and promise of similarity networks.

SIMILARITY NETWORKS IN SPACE APPLICATIONS

The information analysis and case-based reasoning applications discussed above are directly relevant to space applications. Information analysis systems can be used for electrical and environmental systems design, foreign terrain exploration, manufacturing quality control, sensor data identification, and systems configuration support to name a few. Case based reasoners can be applied to systems design, control, and monitoring, physical security advising, and flight tracking.

The following is a sample of the type of information that could be obtained from a similarity network based application in exploratory scanning:

Person: What is the fuzzy, round object located at the lower right portion of the screen?

Computer: I don't know. It is metallic. (75% match score)

Person: It has an unusually high level of radioactivity. Does that help to identify it?

Computer: Changing perspective. It may be the result of the destruction of a nuclear-power device. By the shape, it appears to be a cooling rod. (60%)

Person: How can we retrieve, analyze, and store the object safely?

Computer: Matches radioactive transport situation in case histories (100% match). Retrieve with a robotic arm. Pack with aqueous

transport solution in lead containers.

This example is hypothetical, but it is indicative of the types of systems that can be built with similarity networks.

CURRENT IMPLEMENTATIONS

Similarity networks are currently implemented on two systems. A research version is running on a Symbolics LISP machine at the MIT AI Laboratory. The second is an applications-oriented version at ICF/Phase Linear Systems. The ICF/Phase Linear system is currently being used for three different projects. In the first project we are attempting to link symbolic processing with neural networks. In the second we are developing a business tracking system that monitors successful small businesses over time. The third project is a legal assistant that works from case history data to help solve crimes and predict terrorist attacks.

CONCLUSIONS

Similarity networks are a powerful form of knowledge representation that can be used for a wide variety of artificial intelligence applications. Certain types of applications, such as information analysis and case-based reasoning, are a particularly well-suited for similarity networks. These artificial intelligence programs are applicable to space applications in such areas as systems design, control, and maintenance, sensor input identification, exploration, and knowledge-based navigation of autonomous systems. Current implementations of similarity networks indicate that they provide a good knowledge representation for flexible, interactive artificial intelligence applications.

REFERENCES

Bailey, D.L. Similarity Networks as a Means of Indexing and Retrieving Descriptions. Bachelor's Thesis for the Dept. of Electrical Engineering and Computer Science at the Massachusetts Institute of Technology. 1986.

Winston, P.H. The Psychology of Computer Vision. McGraw-Hill Book Company. 1975.

N88-16412

Discovery and Problem Solving: Triangulation as a Weak Heuristic

Daniel Rochowiak
Philosophy
University of Alabama in Huntsville

Abstract

Recently the artificial intelligence community has turned its attention to the process of discovery and found that the history of science is a fertile source for what Darden has called 'compiled hindsight.' Such hindsight generates weak heuristics for discovery that do not guarantee that discoveries will be made but do have proven worth in leading to discoveries. Triangulation is one such heuristic that is grounded in historical hindsight. I will explore this heuristic within the general framework of the BACON, GLAUBER, STAHL, DALTON and SUTTON programs. In triangulation different bases of information are compared in an effort to identify gaps between the bases. Thus, assuming that the bases of information are relevantly related, the gaps that are identified should be good locations for discovery and robust hypotheses.

Introduction

Part of the fascination of the history of science rests in the accounts of scientific discovery. The exuberant, triumphant stories of scientific discovery give shape to our vision of scientific inquiry and substance to the high status we accord it. However, as one begins to think about, analyze and conceptualize the process of scientific inquiry, clouds of suspicion gather. The triumphant stories are often stories of insight, imagination, luck or other characteristics that seem opposed to the idea that scientific inquiry is orderly, methodic and logical. Are scientific discoveries works of genius unfettered by the dictates of logic and the constraints of empirical research? Are scientific discoveries the results of good fortune and not careful methodic analysis? If so — if discovery requires genius or good fortune — and if one holds that scientific research is the paradigm of methodic, critical, logical reasoning, then it appears that discovery is not really a part of scientific research at all and is certainly not it most distinguishing feature.

The tensions and oppositions of the foregoing considerations can lead to a strict separation of the contexts of discovery and justification. Quickly put, the distinction is a distinction between processes that give rise to new ideas and theories and processes that test proposed ideas and theories. Since it is only in testing ideas and theories that the dictates of logic and the rules of empirical adequacy are appropriate, it is justification, and not discovery, that is the hallmark of scientific inquiry.

If one accepts that the hallmark of scientific inquiry is its base in logic

and standards of empirical adequacy, then if one wishes to claim that scientific discovery is equally a part of and characteristic of that sort of inquiry, then it must be made clear that it is possible for there to be a logic of discovery and attempt to produce that logic.

Heuristics and the process of discovery

Heuristics are procedures that generate desired results some of the time. Unlike algorithms, heuristics do not guarantee that a correct result will be produced in finite time and space. More precisely, an algorithm is a procedure composed of a finite, stepwise sequence of instructions such that (1) given the initially required information the procedure will be completed in finite space and time, (2) no additional information or creativity is required to carry out the instructions, and (3) the result of the procedure is a correct result. This definition is clearly normative owing to condition 3. This condition is added to prohibit consideration of trivially algorithmic procedures. Heuristics differ from algorithms in that the three condition do not apply in a categorical manner. Although heuristics may often terminate with correct results in finite time and space using only specified information, they need not always do so. The strength of a heuristic is a function of its previous success and the conditions under which it is used. Heuristics embody *compiled hindsight* [1]. A heuristic is a strong heuristic when it has been highly successful in the past and is being applied under appropriate conditions.

Discovery is a process that consists of generation and evaluation [4]. In generation new ideas, hypotheses or theories are articulated or constructed. In evaluation these hypotheses, ideas or theories are tested for plausibility. Intuitively, plausibility differs from justification in that theses judged plausible need not be justified but all justified theses must be plausible. The judgments of novelty in generation and plausibility in evaluation are context dependent. For a certain body of information a thesis may be novel and plausible, while for a different body of information it may fail to be either. Understanding discovery as a process combining generation and evaluation allows one to interpret scientific inquiry as movement from the novel and plausible to the routine and justified.

If the notions of 'logic' and 'discovery' are understood to encompass heuristics as well as algorithms and processes of generation and evaluation as well as moments of insight, then it seems reasonable to believe that there are logics of discovery. The reasonableness of this belief does not entail that there is some unique logic of discovery. Rather it allows that there may be several.

The artificial intelligence community has generated several programs that embody logics of discovery. Langley et al. [3] have examined four

particular families of programs: BACON, GLAUBER, STAHL and DALTON. The BACON programs can be understood as generating plausible quantitative laws. By generating a complete experimental combination of the values of dependent and independent variables, and by attempting to extract constants and mathematical relations, the program searches both the space of data and the space of laws in an effort to find laws that accurately summarize the data. The primary heuristics of the BACON programs concern the identification of constants and linear relations. The GLAUBER program can be understood as generating qualitative laws that relate classes of facts. Its primary heuristics concern the formation of classes that best summarize the relations between the predicates, attributes and values of the data and specify the quantifiers (universal or particular) that generate law-like claims. The STAHL program attempts to specify the components of a compound by examining facts about reactions and the substances present in them. The primary heuristics of this program concern reduction, substitution and the identification of components and compounds as being the same. The DALTON program begins with data concerning reactions and the components of compounds and attempts to formulate a model that explains the reaction. Its primary heuristics concern the number of occurrences of atoms and compounds and principles of conservation across reactions. Darden and Rada [2] have devised the SUTTON program to capture the discovery of the chromosome theory of heredity. Its primary heuristics concern part-whole relations, identity and causal propagation.

Discovery programs that concern scientific reasoning clearly profit from the compiled hindsight that can be extracted from the history of science. Procedures that have proved valuable in the past can be converted into heuristics that may be of value in the present. Each of the foregoing discovery programs embodies procedures abstracted from the history of science which are reformulated in terms of the such well understood strategies as 'generate and test,' 'hill climbing' and 'means-ends analysis.'

Triangulation

The heuristics used in discovery programs are neither sufficiently general to be used in all cases nor sufficiently mechanical to guarantee results. Discovery heuristics are context sensitive; their strength varies according to the context. Another heuristic that can be extracted from the history of science turns these difficulties into virtues. *Triangulation* allows for the the comparison and evaluation of different bases of information with the goal of generating more coherent and robust accounts of those bases. [5,6,7].

The heuristic of triangulation can be formulated as a group of related rules concerning generation and evaluation.

1) *If* there is a pattern in domain A that closely matches a pattern in domain B and the pattern of domain B is plausible,
   then use the structure of the pattern in B to generate new patterns in A.
2) *If* the domain A does not have a clearly defined pattern and there is some domain B that contains concepts that closely match those in A,
   then use the structure of the pattern in B to generate new patterns in A.
3) *If* a result in domain A is generated in accord with 1 or 2, and the result closely matches a result in domain B,
   then accept the result as plausible.
4) *If* the plausible results of A closely match the plausible results of B in both structure and concepts,
   then unify the domains and evaluate all of the patterns of A and B in the new domain.
5) *If* plausible results are generated in a domain formed in accord with 4,
   then attempt to justify the results.
6) *If* there are patterns that do not hold for domains formed in accord with 4,
   then identify the conditions under which the patterns do not hold, make these conditions the antecedents of material conditionals and evaluate.

Triangulation is clearly a weak heuristic. There is no guarantee that the process will be successful in entering into the context of justification. Triangulation can generate implausible results and it can generate results that may be erroneous. However, triangulation makes good use of the results of other heuristics used in differing contexts, and attempts to bridge the gaps that could be created by applying other heuristics in a particular domain without considering the results in other domains. It does so by generating hypotheses in accord with both structural and conceptual analogies (rules 1, 2 and 3) derived from other contexts. Further, triangulation amplifies the coherence of results by generating a unified domain and generating new conditionalized hypotheses (rules 4, 5 and 6). The hypotheses generated in this manner serve to address two criteria of plausibility not directly addressed by other heuristics. First scientific hypotheses are often deemed plausible on the basis of analogies to patterns in other more well understood domains. Second scientific hypotheses often gain plausibility by unifying domains even when the unification generates patterns that are more restrictive.

The heuristic of triangulation can be extended to provide a gateway to reasonings that are even more extensible. In the foregoing rules only the relation of unification has been considered. Other relations are possible. Two domains may retain their autonomy and still be relevantly related. Neighboring domains may force constraints on what is to be considered a plausible hypothesis in a particular domain, or a new plausible and

justified result in a particular domain may force alteration in the plausible patterns of other domains. By extending the heuristic of triangulation to include such a gateway, discovery processes that are neither data driven nor theory driven may be investigated.

Conclusion

It is reasonable to consider the context of discovery to be amenable to rational analysis provided that the notion of logic is extended to include heuristics and the notion of discovery is extended to include processes of generation and evaluation. These extensions allow for the possibility of a logic of discovery, but do not demonstrate that there is such. One way in which it can be demonstrated that there is a logic of discovery is by constructing programs that generate discoveries. Such programs have been constructed. However, the heuristics of these programs focus primarily upon the data in a single domain. The heuristic of triangulation uses the patterns and results of one domain to generate and evaluate the results of another domain. This heuristic focuses on the scientific values of analogical support and increased coherence, and makes possible a gateway to other forms of extensible reasoning. Thus, triangulation should prove to be a valuable addition to the treasury of heuristics of discovery.

References

[1] Darden, Lindley. "Viewing the history of science as compiled hindsight," *Institute for Advanced Computer Studies (University of Maryland)*, UMIACS-TR-86-20/CS-TR-1715. Forthcoming in AI Magazine.

[2] Darden, Lindley and Roy Rada. "Hypothesis formation using part-whole interrelations," *Institute for Advanced Computer Studies (University of Maryland)*, UMIACS-TR-87-17/CS-TR-1832

[3] Langley, Pat, Herbert Simon, Gary Bradshaw and Jan Zytkow, *Scientific Discovery: Computational Explorations of the Creative Processes*, Cambridge, Mass.: The MIT Press, 1987.

[4] Schafffner, Kenneth. "Discovery in the biomedical sciences: logic or irrational intuition" in *Scientific Discovery: Case Studies*, Thomas Nickles (ed.), Boston: D. Reidel, 1980, pp.171-206.

[5] Rochowiak, Daniel. "Darwin's psychological theorizing: triangulating on habit," *Studies in History and Philosophy of Science*, forthcoming.

[6] Rochowiak, Daniel. "Extensibility and completeness: an essay on scientific reasoning," under review.

[7] Rochowiak, Daniel. "Expertise and reasoning with possibility," NASA Artificial Intelligence Conference, November 1986.

N88-16413

Commonality Analysis as a Knowledge Acquisition Problem

Dorian P. Yeager
The University of Alabama
College of Engineering
Department of Computer Science
Tuscaloosa, Alabama 35487

Abstract. Commonality Analysis is a systematic attempt to reduce costs in a large scale engineering project by discontinuing development of certain components during the design phase. Each discontinued component is replaced by another component which has sufficient functionality to be considered an appropriate substitute. The replacement strategy is driven by economic considerations. The tool currently in use by NASA to guide the commonalty analysis process, known as the System Commonality Analysis Tool (SCAT), is based on an oversimplified model of the problem and incorporates no knowledge acquisition component. In fact, the process of arriving at a compromise between functionality and economy is quite complex, with many opportunities for the application of expert knowledge. Such knowledge is of two types: (1) general knowledge expressable as heuristics and mathematical laws potentially applicable to any set of components, and (2) specific knowledge about the way in which elements of a given set of components interrelate. Examples of both types of knowledge are presented, and a framework is proposed for integrating the knowledge into a more general and useable tool.

Introduction. Component part standardization has been used as a means of increasing volume and reducing the cost of manufacturing goods since the industrial revolution. The major cost saving was due to mass production, which dramatically reduced the cost of producing each unit. A side benefit was that items manufactured in this way were cheaper and easier to repair, because replacement parts were plentiful and reliable. Commonality is a similar technique, applied at a higher level. Commonality analysis attempts to standardize components on a system-wide basis, or across multiple systems in a large engineering effort. The components involved are more complex, serving multiple functions. For example, Boeing Corporation has saved millions of dollars in development, production, and maintenance costs, as well as in pilot training, by employing identical cockpits in the Boeing 757 and 767 aircraft. The earlier in a large engineering effort that the principle of commonality is employed, the greater the potential cost-saving benefits.

In a general sense, commonality analysis refers to an objective evaluation of a large and complex project at a fairly early stage in its design with the goal of finding opportunities to apply the principle of commonality. Much of what can be called commonality analysis is highly creative and has no fixed methodology. Howev-

PRECEDING PAGE BLANK NOT FILMED

er, there is one activity that appears to run through all such analyses as a unifying thread: the direct comparison of two or more competing designs, to ascertain the feasibility of eliminating some of those designs. Often the functionality of an item can be extended in such a way that it may serve other purposes while continuing to function in the original fashion as well. It may also be possible to use multiple copies of one item in place of another. In still other cases it may be possible to make a simple substitution, eliminating an item whose functionality can be completely assumed by another.

Current Software Solutions. Only one type of comparison lends itself easily to automation via software, and that is the type of comparison which strives to evaluate the advisability of substituting one or more copies of one item for another, without examining the possibility of redesigning or extending the functionality of any items. In this case it is often sufficient to simply evaluate a cost function. In the case of a two-way comparison, say between item a and item b, the cost function must be evaluated for three situations: that in which a and b are uniquely implemented, that in which a substitutes for b, and that in which b substitutes for a. The three numbers are compared, and the lower cost wins. This simple strategy is the basis for a software tool currently in use by NASA, called the System Commonality Analysis Tool, or SCAT (See [1]). SCAT evaluates a set of n objects by computing n+1 costs: the so-called "unique option", plus the n possible substitution strategies.

Of course, cost functions must be given sufficient data on each item in order to give realistic predictions of comparative costs. The gathering and management of that data is another need which indicates a software solution. The SCAT program incorporates data management facilities. In fact, SCAT is written as a front end to a commercial database management system (DBMS). SCAT obtains the data it needs for its cost analyses from files created with the DBMS functions.

SCAT operates as follows. Design data on hardware and/or software components are captured as records in commonality databases. Each such database is created and maintained by a database administrator familiar with the project. A separate record is made for each item which may be a candidate for comparative cost analysis. The attributes of a record must always include those required by the SCAT cost function. To insure this, the database administrator is constrained to create the database via the SCAT front end, which automatically supplies the needed attributes with each new database. However, there is no requirement that all items entered into a database have identical, or even very similar, functional characteristics. Nor is there any capability within SCAT to search for sets of items with related functional characteristics. For its comparative cost analyses, SCAT relies on the database administrator to communicate to it precisely the subset of n items which it is to evaluate. This is done with standard database subsetting operations, communicated via a

series of menus which painstakingly prompt for the necessary information to construct a relational expression to be used as a query. For those with more relational database experience, direct access to the DBMS proper is provided.

Once the subset database is identified which is to be subjected to analysis, the SCAT user may request a cost analysis on that subset. SCAT then assumes that the items in the subset have identical functionality, and provides the requested n+1 cost figures, sorted in increasing order. The final assumption is that the most "cost effective" alternative will be a viable alternative.

A More General Formulation. Let α be the relation, defined on the set of all records in a given commonality database, as $a\alpha b$ if and only if a is a feasible substitute for b. We call α the feasibility relation on that set of records. The properties of the relation α depend entirely on the characteristics of the given database. α may or may not be symmetric, antisymmetric, or transitive. As a convention, we take $a\alpha a$ to be true for all items a in the database (that is, α is always reflexive). By rights, it is the connected components of the relation α that ought to be subjected to analysis. In other words, if a record x is in a given subset being subjected to analysis, we would wish that all records y be also in the subset, where there is a series of records $x_1, x_2, \ldots, x_m$, for which $x_1 = x$, $x_m = y$, and for each $i = 1, 2, \ldots, m-1$, either $x_i \alpha x_{i+1}$ or $x_{i+1} \alpha x_i$.

Let us assume that we have isolated one of these subsets, say A, and that it is in fact a connected component of α. The form of the relation on set A may be arbitrarily complex. Let us consider the simple case of a two-element set, the two elements a and b we referred to in our discussion of SCAT, above. If both $a\alpha b$ and $b\alpha a$ are true, then all three SCAT options make sense, and we choose the least costly. If only one of them is true, for example if $a\alpha b$ and not $b\alpha a$, then we may or may not choose to replace b by a, even though α permits us to do so. It may be more cost-effective to produce the two items separately. However, if we run a SCAT analysis on the set, the recommendation may be to substitute b for a, even though that is not a viable alternative. SCAT's recommendations must be filtered through a human expert, who knows which solutions make sense and which do not. Now let us add a third element, c, to the set. An interesting fact here is that the most economical alternative may be to substitute a for c and produce b uniquely. This may be true because of the form of the relation α. For example, it may be that the only two non-reflexive relationships are $a\alpha c$ and $b\alpha c$. However, depending on which cost function one uses, such a twofold strategy may be called for even if α freely allows substitutions of all kinds in the set {a,b,c}.

The most general substitution strategy is represented by a pair (π,T), where π is a partition of the set A and T is a set of representatives of π. In the example above, the partition is π =

{{a,c},{b}} and the set of representatives is T = {a,b}. If $\pi = \{K_1, K_2, \ldots, K_m\}$, and $T = \{t_1, t_2, \ldots, t_m\}$, then it must be true that for each i = 1, 2, ..., m, $t_i \alpha x$ for all x in K_i. For this reason we call (π,T) an α-partition.

A SCAT-type solution can now be seen as a special case of this general form. It is the case where the partition π and the set T have only one element each. That is, $\pi = \{A\}$ and $T = \{t\}$ for some element t of A.

The Need for a New Methodology. Clearly, techniques for generating the more general form of solution described above will be much more complex than the simple SCAT strategy. An initial collection of knowledge about α-partitions and cost functions on α-partitions is available in the form of a series of propositions contained in a paper [2] submitted by the author to the journal, Operations Research. Several of these propositions suggest algorithms which may be applied to provide a sub-optimal solution, which may then be refined by heuristic techniques. Because of the very general nature of the problem, there probably is no deterministic algorithm which will yield an optimal solution in every case, and each case must be examined in light of its own properties. An eclectic solution strategy is called for. Logic programming is the obvious tool for investigating such solution strategies because of the natural way in which propositional knowledge may be encoded.

Capturing the Feasibility Relation. The perfecting of a generalized solution strategy for commonality analysis is an intriguing problem, but there is a companion problem which is just as intriguing. To be able to say that widget a is a feasible substitute for widget b clearly requires expert knowledge about widgets. To search through a database of hundreds of widget designs and produce a set of twelve which are closely related to the extent that a SCAT-type analysis may be performed on that set also requires a certain level of expertise. Is there any hope that this process may yield to a software solution? If so, then a knowledge base component is necessary.

It is possible to capture the knowledge about α and store it as an integral part of the commonality database itself. Clearly, there must be a close physical association between the data and the knowledge whereby the relation α on that data may be constructed. We propose, then, that every commonality database be accompanied by a companion knowledge base. The construction and maintenance of the knowledge base would be the responsibility of the database administrator.

Let us examine how the knowledge might be encoded. In the SCAT environment, the user is encouraged to find a set of items for analysis by sorting on various attributes and scanning the sorted list for potentially common sets of items. When such a group appears, the user may communicate to SCAT the set he or she is interested in by means of a relational expression which

identifies the desired set. If the decisions concerning how to sort and group the data are made in advance, the entire process of selecting a subset for analysis can be carried out in a single automated operation.

But let us not confine ourselves to SCAT-type methodology. What we are trying to do is to capture the feasibility relation α. Any information about α will be useful, even if it consists only of a single pair (a,b) of records. The forms taken by the knowledge will be varied. The following list covers some of those forms.

Type of Information	Parameters Needed
pair	<record key>_1, <record key>_2
sort	<attribute>,<direction>,...
group	<attribute>,<range of values> <attribute>,<relation> <attribute>,<tolerance>
relational expressions	No specific form for parameters. May use a specially designed prefix or postfix coding scheme.

Conclusions. The Commonality Analysis problem requires expert knowledge at all phases of the solution process. The creation of databases, the maintenance of data and knowledge about the data, the selection of commonality alternatives, and the application of solution strategies may all profit from software solutions that incorporate knowledge. The report [3] referenced below presents an overall strategy for the incorporation of knowledge.

Acknowledgements. The author expresses his gratitude to NASA and the NASA/ASEE Summer Faculty Fellowship program, which sponsored his initial research into this topic. Special thanks go to Dale Thomas for his enthusiasm, encouragement, and ideas.

References.

1. MSFC. Commonality Analysis Study, User Manual for the System Commonality Analysis Tool (SCAT), D483-10064, March 1987. Contract NAS8-36413, NASA George C. Marshall Space Flight Center, Alabama.

2. Yeager, D. P. A Formulation of the Commonality Analysis Problem and Some Partial Solutions, submitted to Operations Research, 1987.

3. Yeager, D. P. Expert System Development for Commonality Analysis in Space Programs, in final report of the 1987 NASA/ASEE Summer Faculty Fellowship Program, NASA George C. Marshall Space Flight Center, pp. XXXV-i through XXXV-25.

N88-16414

QUALITATIVE MODELS FOR PLANNING: A GENTLE INTRODUCTION

James D. Johannes
James R. Carnes

Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama 35899
Phone (205) 895-6255/6088

ABSTRACT

Qualitative Modeling is the study of how the physical world behaves. These physical models accept partial descriptions of the world and output the possible changes. Current systems assume that the model is static, and that physical entities do not effect change into the world. For instance a certain qualitative systems can diagnose faulty electrical circuits, but cannot design plans to rewire circuits to change their behavior. This paper describes an approach to planning in physical domains and a working implementation which integrates qualitative models with a temporal interval-based planner. The planner constructs plans involving physical quantities and their behavioral descriptions.

1. INTRODUCTION

This paper describes a system for the representation and solution of dynamic planning problems. The representation of physical entities in qualitative models provides an excellent forum for storing definitional and behavioral information about a particular domain [2,8,10]. This knowledge is easily coupled, in the model, with a temporal component permitting the representation of entity behavior and interaction over time. The temporal capability is accomplished with the introduction of a time interval into the qualitative knowledge model. Inference mechanisms that usually run with the detailed qualitative information are now time-qualified, adding another "dimension" to the knowledge model [6,10]. The prototype system developed to illustrate these concepts is described, and, finally, directions for future work are outlined.

2. QUALITATIVE MODELS

Qualitative models are aptly described by Williams[10] as a physical system with initial conditions whose analysis typically involves (1) a description of the temporal behavior of the

PRECEDING PAGE BLANK NOT FILMED

system's state variables, in terms of a particular qualitative representation, and (2) an explanation how this behavior came about.

In more detail, a physical system consists of a set of state variables (e.g., force and acceleration) and a system of equations, parameterized by time, which describe the interactions between these variables (e.g., f(t) = ma(t)). A qualitative representation partitions the range of values for a particular quantity into a set of interesting regions (e.g., positive, negative, or zero). The particular representation selected depends on properties of the domain and the goals of the analysis.

The behavior of the system can be viewed in terms of a qualitative state diagram, where each state describes the qualitative value of every state variable in the system. The behavior of the system over time can be viewed as a particular path through this state diagram. Each state along this path represents an interval of time over which the system's state variables maintain their values. The duration of this interval is dictated by principles involving continuity and rates of change.[4,6,10]

The system changes state whenever any state variable changes its qualitative value. The values in the next state are then determined by (1) identifying those quantities which cannot change value (e.g., if q is positive in a particular state and its derivative is positive or zero then it will remain positive in the next state), and (2) propagating the effects of those quantities that are known to change. The qualitative reasoning system also keeps track of the reason for every deduction in (1) or (2), using the record, among other things, to generate explanations (e.g., "an increase in force causes the mass to accelerate").[5,6,10]

Components in a qualitative model may be expressed as an object containing five elements[3,9]: individual objects involved in the process, preconditions (outside of the object knowledge) on the behavior, quantity conditions (inequalities), relations asserted as object behavior, and influences the behavior has on quantities.

3. TEMPORAL INTERVALS

The planning system maintains a list of entity qualities or properties qualified by intervals over which they hold. The planner uses a time logic[1] to maintain the temporal relationships between intervals. Table 1 shows the possible values for different relations.

In operator and rules defined within the system, intervals are represented by symbols starting with '$', while variables are represented by symbols starting with '?'. The temporal relation between two intervals is expressed as a disjunction and written

in a list (e.g., (:< :>) is used to mean "is before or after"). Different properties or facts about an object are paired with the same interval over which they hold. Thus, (ON A TABLE) $INTERVAL1 denotes a fact (ON A TABLE) which holds over the time interval $INTERVAL1.

Value	Description	Inverse	Description
:<	before	:>	after
:M	meets	:MI	met by
:O	overlaps	:OI	overlapped by
:S	starts	:SI	started by
:F	finishs	:FI	finished by
:D	during	:DI	encloses
:=	equals	:=	equals

TABLE 1: Interval Relations and Inverses[1].

An operator defines an action the object can perform to change its properties in the world. The planning system uses a model of action[1] with temporally qualified expressions describing operator preconditions and effects. For instance, Figure 1 defines an operator PICKUP which can be applied to an object if it is clear and resting on something. PICKUP's effects clear the object's old location. The constraints field is used to restrict the temporal relations among facts matching with the preconditions and effects.

```
OPERATOR: pickup
  PRECONDITIONS: (clear ?object) $clear-object
                 (on ?object $surface) $on
  EFFECTS: (pickup ?object ?surface) $pickup
           (clear ?surface) $clear-surface
  CONSTRAINTS: $clear-object (:M) $pickup
               $on (:O) $pickup
               $on (:M) $clear-surface
```

FIGURE 1: Definition of Operator PICKUP[3].

A rule models temporal laws of the domain. The planning system uses rules as backward chaining operators for solving goals, as well as forward chaining, temporally constrained inference rules. Thus, with the object behavior modeled as rules, the system can both plan their action and infer their results.

Rule definitions are similar to operator definitions, with antecedents behaving like operator preconditions, consequents behaving like operator effects, and consequence constraints behaving like operator constraints. The additional field, temporal conditions, places preconditions on the temporal relations among facts matching the antecedents. The time logic supports temporal intersections, allowing a rule to be inhibited until

antecedents are known to intersect (meaning their relation is a subset of (:S :SI :F :FI :D :DI :O :OI :=)) and to assert consequents over their intersection. Figure 2 demonstrates these features. Given (ON A B) and (ON B C) whose intervals intersect, (OVER A C) is asserted during their intersection.[3,9]

```
RULE
  Antecedents: (ON ?x ?y) $on-xy
               (ON ?y ?z) $on-yz
  Temporal Conditions:
    Exists (INTERSECTION $on-xy $on-yz)
      called $intersection
  Consequents: (OVER ?x ?z) $intersection
  Consequent Constraints:
```

Figure 2: A Temporally Qualified Inference Rule[3].

4. APPLICATION

The constructs of the previous section are part of a simple temporal planning example to operate on selected objects. Plans can be generated to "pick-up" A, B, or C. This example is, however, very different from the typical "blocks world" environment in two respects, each object have been physically described and temporally qualified. The physical description portrayed the objects ability to placed over one another and even how many may be stack above each particular object. This aspect of the system has not been exploited in order to concentrate on temporal planning which give the intervals in which each object resides over another object. Figure 3 shows the script generated by the planner to meet a goal in this simple system.

```
Goal: PICK-UP A t1
Solution: Apply the rule to see if any objects are over A at t1.
  Goal: PICK-UP B
  Solution: Apply the rule to see if any objects are over B at t1.
    Goal PICK-UP C
    Solution: Apply the rule to see if any objects are over C at t1.
    Action: pick-up C
  Action: pick-up B
Action: pick-up A
```

FIGURE 3: Output for Goal to Pick Up Object A.

Qualitative models such as those described in this paper will be necessary for detailed planning operations onboard space station and for many other space applications. Planning systems using temporally-qualified structural and behavioral knowledge will be able to plan the independent actions of IVA or EVA robots, is needed to function in a dynamic, time-varying,

environment[7]. Qualitative systems will also be able to generate complex plans for multiple experiment packages, using knowledge of core subsystem properties to keep operations within established constraints.

5. CONCLUSIONS AND FUTURE DIRECTIONS

Qualitative modeling represents the physical and temporal information required for dynamic planning applications. Characteristic and behavioral information describe an entity in term of properties which are used in different time qualified rules and operators to generate plans to achieve desired goals. A simple example of temporal reasoning about physical objects was presented, but examples of more sophisticated thermal and electrical systems of entities have been accomplished.

Work of this nature is helpful in developing and evaluating representations for qualitative modeling and planning, that is controlling the effects of time, space, and general properties of physical objects [6,7]. Current efforts in the design of the space systems are requiring the capture detailed knowledge of system design[7], so new space systems may incorporate advanced knowledge-based applications, such as planning systems driven by qualitative models. Accordingly, this research will continue to explore detailed representations within domain and world models, and investigate different planning strategies for reasoning and control.

REFERENCES

1. Allen, J.F., Maintaining Knowledge About Temporal Intervals, Communications of the ACM, Vol. 26, No. 11, 1983, pp. 832-843.

2. Brachman, R.J., R.E. Fikes, and H.J. Levesque, KRYPTON: A Functional Approach to Knowledge Representation, IEEE Computer, Vol. 16(10), 1983, pp. 67-73.

3. Hogge J.C., Compiling Plan Operators form Domains Expressed in Qualitative Process Theory, AAAI-87, Proceedings of the Sixth National Conference on Artificial Intelligence, August 13-17, 1987, pp. 229-233.

4. Johannes, J.D., Representation and Use of Judgmental Knowledge, Proceedings IEEE COMPSAC, Chicago, Illinois, November 1983, pp. 156-161.

5. Johannes, J.D., Judgmental Knowledge Representation for Problem Solving and Expert Systems, IEEE COMPDEC: Computer Data Engineering Conference, Los Angeles, California, April 1984, pp 286-292.

6. Morgenstern, L. A First Order Theory of Planning, Knowledge, and Action, Proceedings of the Conference on Theoretical Aspects of Reasoning About Knowledge, March 19-22, 1986, pp. 99-114.

7. NASA TM 89190, Advancing Automation and Robotics Technology for the Space Station, Advanced Technology Advisory Committee (ATAC), Progress Report Three, October 1, 1986, pp. 27-28.

8. Purves, R.B., J.R. Carnes, and D.E. Cutts, Foundation: Transforming Data Bases into Knowledge Bases, Proceedings of the Conference on Artificial Intelligence for Space Applications, November, 1987.

9. Wilensky, R.S., Planning and Understanding, Addison-Wesley, 1983.

10. Williams, B.C., Doing Time: Putting Qualitative Reasoning on Firmer Ground, AAAI-86, Proceedings of the Fifth National Conference on Artificial Intelligence, August 11-15, 1986, pp. 105-112.

N88-16415

Operational Aspects of a Spacecraft Planning/Scheduling Expert System

David R. McLean, Ronald G. Littlefield, and David S. Beyer
Bendix Field Engineering Corporation
10210 Greenbelt Road
Lanham-Seabrook, MD 20706

ABSTRACT

This paper describes various operational aspects of the ERBS-TDRSS Contact Planning System. The ERBS-TDRSS Contact Planning System is an expert system which has been used operationally since June 1987 by the Earth Radiation Budget Satellite (ERBS) Flight Operations Team (FOT) at Goddard Space Flight Center to build weekly schedules of requests for service from the Tracking and Data Relay Satellite System (TDRSS). The ERBS-TDRSS Contact Planning System, which is written entirely in the C language and runs on an IBM PC-AT, reads ERBS orbit prediction data from a 9-track tape and builds a 1-week schedule of requests for TDRSS service in three minutes, with additional time required to print the schedule that has been built. By comparison, the ERBS FOT scheduling expert previously had to spend over six hours per week to do this same task. Several enhancements have been made to the system since it has become operational. In addition, modifications have been made to the knowledge bases of the system to accomodate changing operational scenarios and mission constraints imposed by the TDRSS Network Control Center (NCC). Contributing to the operational success of this planning/scheduling expert system are: (1) the resource window generator which limits search through the orbit data, (2) the custom-built strategies interpreter which applies various strategies to try to schedule primary and alternative events, (3) the custom-built inference engine (TIE1) which automatically checks each event proposed by the strategies interpreter against mission-specific scheduling constraints, (4) the custom-built user interface to the system, and (5) the report generator which produces TDRSS schedule requests in the format required by the NCC.

INTRODUCTION

This paper begins by describing the basic operation of the ERBS-TDRSS Contact Planning System (ERBS-TDRSS CPS). Next, significant enhancements to the ERBS-TDRSS CPS and changes in its operational characteristics are discussed. Finally, some conclusions based on several months of operational experience are presented.

A schematic diagram of the ERBS-TDRSS CPS is shown in Figure 1.

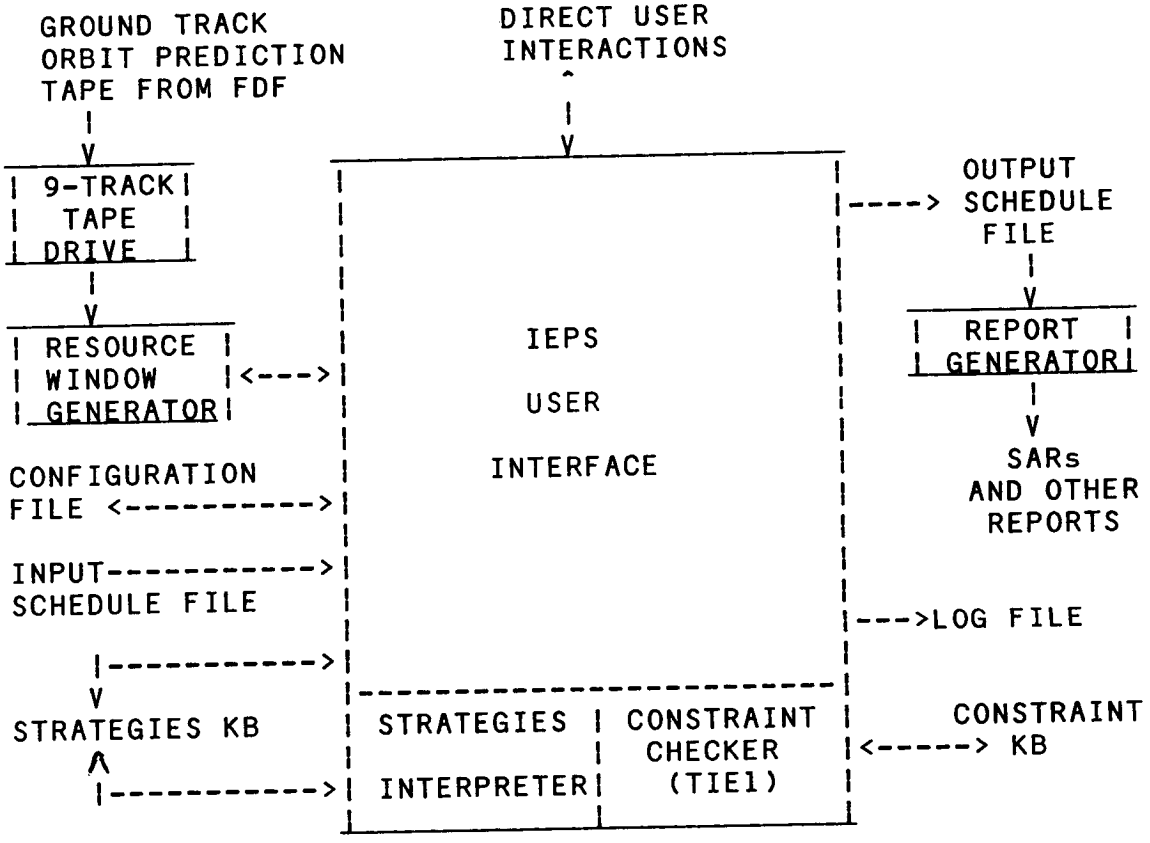


Figure 1: Schematic Diagram of the ERBS-TDRSS Contact Planning System.

In its operation, the ERBS-TDRSS CPS first directs a tape drive to read selected orbital information from a 9-track Ground Track Orbit Prediction data tape obtained from the GSFC Flight Dynamics Facility. Orbital information from the tape is written to the PC that runs the ERBS-TDRSS CPS. This data includes such items as antenna viewing angles and start and stop dates and times for each orbit and daylight period.

Next, resource window generator routines on the PC apply scheduling heuristics to the orbital data to identify resource windows. This process improves overall system efficiency by reducing the amount of data which the strategies interpreter must search through to find TDRSS contact periods.

As the ERBS-TDRSS CPS is run in its automatic scheduling mode, resource window information is used along with system configuration information from a configuration file, scheduling strategies from the strategies KB, and scheduling constraints from the constraint KB to build a 1-week schedule of TDRSS requests. As the strategies interpreter applies primary and alternative strategies to find valid candidates for TDRSS

contacts, tentative event information is passed to the constraint checker, the Transportable Inference Engine (TIE1). TIE1 checks this event information against mission-specific scheduling constraints in the constraint KB to determine whether the candidate event should be scheduled. If the primary event strategy is rejected by the constraint checker, the strategies interpreter automatically tries alternative scheduling strategies and passes updated event information back to the constraint checker. This process is repeated for each event until it is scheduled or the strategies list is exhausted.

After a 1-week schedule has been produced in the automatic scheduling mode, the ERBS-TDRSS CPS can be used in the interactive scheduling mode. In this mode, the Interactive Experimenter Planning System (IEPS) user interface provides graphical displays of event timelines and user-friendly editing capabilities.

During interactive scheduling mode operations, the constraint checker is automatically invoked to help the ERBS FOT planner add or delete individual events in accordance with the scheduling constraints in the constraint KB. When there is a scheduling conflict, TIE1 produces a diagnostic message to explain why the event cannot be scheduled. By entering a password, the ERBS FOT planner can override the constraint checker and force the system to schedule the event.

Finally, report generation routines in the ERBS-TDRSS CPS are used to produce reports including: selected data tape records, resource window listings, TDRSS contact request worksheets, and specially formatted Schedule Add Requests (SARs). The SARs that are produced are sent (via a Mission Planning Terminal) to the NCC.

Three papers [1], [2], and [3] that provide additional information about the ERBS-TDRSS Contact Planning System are listed in the reference section.

OPERATIONAL EXPERIENCES

When the ERBS-TDRSS CPS began operating, Ground Track Orbit Prediction data from a 9-track tape was read into an IBM mainframe in the Command Management Facility and then transferred (via a modem) to the PC in the ERBS FOT area. It took about 20 to 30 minutes to transmit the data required for a 1-week schedule. To enhance the ERBS-TDRSS CPS, a tape drive with flexible software was added. This tape drive allowed orbital data for one week to be read from a 9-track tape in less than a minute. In addition, selected orbital data tape records and resource window listings could be quickly verified by the ERBS FOT planner.

After the ERBS-TDRSS CPS operations began, it became evident that the ERBS FOT planner was not comfortable with the multitude of DOS operating system commands required to run the ERBS-TDRSS CPS. To make the control of the ERBS-TDRSS CPS more user friendly, a commercial product was used for adding a high level menu-driven user interface to the system. This interface includes options for: loading the data tape, selecting the scheduling mode (automatic or interactive), entering start dates, selecting antenna service, running the scheduling process, and printing reports. In addition, since the orientation of the spacecraft relative to its direction of travel affects its ability to view TDRSS (and the spacecraft is occasionally "flipped" end-for-end), a menu prompts the ERBS FOT planner at the start of each scheduling session to specify the orientation of the spacecraft and when the spacecraft is scheduled to be flipped.

In response to changing scheduling requirements imposed on the ERBS Flight Operations Team, significant alterations were made to the knowledge bases of the ERBS-TDRSS CPS. For example, several new types of Tape Recorder Dump (TRU Dump) and Stratospheric Aerosol and Gas Experiment (SAGE) events (along with their scheduling strategies and constraints) were added to the knowledge bases. Also, the durations of some tape recorder dump events were changed and separate knowledge bases for scheduling SAGE events during periods of "full-sun" were added. These knowledge base changes provided new scheduling capabilities for the ERBS-TDRSS CPS.

CONCLUSIONS

Although the ERBS-TDRSS CPS has only been operational for several months, significant changes have been made to the system. Some of these changes, such as the addition of the tape drive and the menu system, were made to increase the performance of the system and to make the system more user friendly.

Other changes were made to support new scheduling requirements imposed on the ERBS FOT and resulted in new and different operating capabilities for the system. It is expected that maintenance expertise to support changing scheduling requirements will be required for the life of the mission. For example, when TDRS-West becomes operational, the ERBS-TDRSS CPS will probably have to be modified.

An important aspect of maintaining the ERBS-TDRSS CPS has been the ability to add new scheduling capabilities by editing its knowledge bases. For example, rules and frames in the strategies and constraint knowledge bases were quickly and easily modified to provide capabilities for scheduling several new types of events, each with its own antenna configurations.

Modifications such as these would likely have been much more difficult and time consuming to make in a system with a more conventional software architecture.

Another important factor in maintaining the ERBS-TDRSS CPS was having access to the source code. For example, changes to the source code were required in order to enhance the resource window generator. Most commercial expert system shells do not include access to the source code.

Finally, having a portable, C-based system that runs on conventional hardware, such as IBM PC-ATs under DOS or MC/68020 workstations under UNIX, has been very helpful. Maintenance and development work has been done mainly on the 68020-based workstation utilizing its powerful development environment. The updated source code was then ported to the ERBS-TDRSS CPS PC and recompiled.

ACKNOWLEDGEMENTS

The authors would like to thank Patricia Lightfoot, William Macoughtry, and Carolyn Dent of The Spacecraft Control Programs Branch at NASA-GSFC and Ellen Stolarik of Bendix Field Engineering Corporation for their many contributions to the Interactive Experimenter Planning System task. In addition, the authors would like to thank Doris Tallman of the ERBS Flight Operations Team at NASA-GSFC for her expert help. This work was supported by NASA contract NAS5-27772.

REFERENCES

[1] McLean, David R., "The Design and Application of a Transportable Inference Engine (TIE1)," PROCEEDINGS OF THE 1987 GODDARD CONFERENCE ON SPACE APPLICATIONS OF ARTIFICIAL INTELLIGENCE (AI) AND ROBOTICS, Goddard Space Flight Center, Greenbelt, MD., May 15, 1986.

[2] McLean, David R., Littlefield, Ronald G. and Beyer, David S., "An Expert System for Scheduling Requests for Communications Links between TDRSS and ERBS," PROCEEDINGS OF THE 1987 GODDARD CONFERENCE ON SPACE APPLICATIONS OF ARTIFICIAL INTELLIGENCE (AI) AND ROBOTICS, Goddard Space Flight Center, Greenbelt, MD, May 13-14, 1987.

[3] McLean, David R., Littlefield, Ronald G., and Macoughtry, William O., "Defining and Representing Events in a Satellite Scheduling System: the IEPS (Interactive Experimenter Planning System) Approach," 1987 INTERNATIONAL TELEMETERING CONFERENCE, San Diago, California, October 26-29, 1987.

N88-16416

Planning and Scheduling for Robotic Assembly†

Barry R. Fox
Artificial Intelligence Group
McDonnell Douglas Research Laboratories
PO Box 516, St. Louis, MO 63166

Abstract. This paper describes a system for reasoning about robotic assembly tasks. The first element of this system is a facility for itemizing the constraints which determine the admissible orderings over the activities to be sequenced. The second element is a facility which partitions the activities into independent subtasks and produces a set of admissible strategies for each. Finally the system has facilities for constructing an admissible sequence of activities which is consistent with the given constraints. This can be done off-line, in advance of task execution, or it can be done incrementally, at execution time, according to conditions in the execution environment. The language of temporal constraints and the methods of inference presented in related papers are summarized. It is shown how functional and spatial relationships between components impose temporal constraints on the order of assembly and how temporal constraints then imply admissible strategies and feasible sequences.

Introduction

Construction of the Space Station requires an unprecedented degree of advanced planning and scheduling. The sequence of construction activities must be consistent with the functional and spatial relationships between the components of the space station, it must be consistent with feasible methods for delivering and fabricating the structure, and it must be consistent with high-level schedules and timetables. It must be possible to produce schedules which satisfy these constraints and it must be possible to quickly produce revised schedules in the event of problems and delays.

In this context the problem is not simply to produce a single schedule of construction activities, but rather to carry the production plan through multiple levels of refinement. In its most general form the plan may be simply a collection of the constraints which must be satisfied. After some refinement the plan may be partitioned into independent subtasks and the plans for those subtasks may be factored into strategies. Ultimately, all of the construction activities will be mapped onto intervals of a timeline. When problems or delays occur it should be possible to revise the schedule according to the constraints imposed by a higher level in the scheduling hierarchy. This view of planning and scheduling is supported by a system for reasoning about robotic assembly tasks described below.

† Research supported in part by the McDonnell Douglas Independent Research and Development Program.

PRECEDING PAGE BLANK NOT FILMED

The Representation of Robotic Assembly Tasks

Robotic assembly is fundamentally a serial process. Although the steps of an assembly can be ordered in a variety of ways, a robot typically performs an assembly one step at a time. This has two important implications. First, the analysis of assembly tasks is simplified. Problems of synchronization and concurrency are found only in the interaction between processes not within the assembly task itself. Second, the execution of assembly tasks is simplified. The flexibility in the ordering of the assembly steps can be exploited to adapt the sequence of operations to execution time conditions [Fox and Kempf, 1986a]. However, the ability to reason about the execution of an assembly task depends intimately upon the form and content of some abstract representation of that task. Proposed representations include state transition networks [Whitney], and-or graphs [deMello and Sanderson], precedence diagrams [Prenting and Battaglin], Petri nets [Drummond],[Malcolm and Fothergill], and temporal constraints [Fox and Kempf, 1986b]. These and other representations are discussed in detail in another paper [Fox, 1987a]. The remainder of this paper will focus on the derivation and analysis of temporal constraints.

The Derivation of Temporal Constraints

Allen defines 13 relationships that can exist between two intervals of time [Allen]. However, the serial execution of an assembly task T reduces the relationship between any two steps to exactly two possibilities: $\forall x, y \in T, (x < y) \vee (y < x) \wedge \neg((x < y) \wedge (y < x))$ (serial axiom) where the expression $(x < y)$ denotes the constraint that step x must strictly precede step y. In addition, the transitive nature of time determines the effect of combinations of relationships: $\forall x, y, z \in T, (x < y) \wedge (y < z) \Longrightarrow (x < z)$ (transitive axiom).

The constraints which govern the execution of assembly task can be stated as a conjunction of the relationships which exist between steps of the task. In some cases the order of two steps is unconstrained and it is unnecessary to explicitly state their relationship. It is simply assumed that their order of execution is governed by the serial axiom. In other cases exactly one of the two possible orderings is required and it is sufficient to state that relationship as a primitive constraint of the form $(x < y)$. In more complicated cases the necessary ordering over two steps will be conditional upon the ordering of other steps. In such situations it will be necessary to state the relationships among these steps using combinations of primitive constraint expressions and the logical operators $\wedge$ (and), $\vee$ (or), $\neg$ (not), and $\Longrightarrow$ (implies).

The underlying physical relationships which determine the necessary temporal constraints are varied. While not exhaustive, a number of cases will be enumerated in order to suggest the range of possibilities.

Some of the relationships between the objects to be manipulated determine enabling conditions for steps of the assembly task. For example, the installation of a major component of an assembly then enables the installation of those parts that are attached to or supported by that part. In other situations, the necessary support for a part may be a collection of other parts. In more complex cases there may exist multiple distinct strategies or methods for completing an assembly and a given step may be enabled by different conditions under each strategy.

In a similar fashion, some of the relationships between the objects to be manipulated determine disabling conditions for steps of the assembly task. In some cases the installation of a part may prevent the subsequent installation of other parts. In other cases, the installation of a part may be prevented by the installation of any one of several other parts. Occasionally the destination for a part can be approached from several different directions and the installation of that part will be disabled only after every one of several other parts has been installed.

It is not unusual for some steps of the assembly to be temporarily disabled. For example, the installation of a special jig may be necessary for some stage of an assembly but it may interfere with some otherwise admissible operations. Those operations must be performed either before the installation of the jig or after its removal but they are impossible while it is in place. In other cases the assembly may go through fragile or unstable states when forceful or agitating operations should be avoided. Although they may be otherwise enabled, the rough operations should either precede or follow all of the delicate operations.

The specification of an assembly task begins with an analysis of the objects to be manipulated and the operations to be performed. That analysis will reveal the significant physical relationships which enable and disable the various operations. The result of that analysis will be a logical formula composed of primitive ordering constraints of the form $(x < y)$ and the logical operators $\wedge$ (and), $\vee$ (or), $\neg$ (not), and $\Longrightarrow$ (implies). Example assembly tasks are presented elsewhere [Fox 1987a],[Fox and Kempf, 1986b].

While this analysis may be a human activity, the translation of physical relationships into temporal constraints can be automated to a certain degree. Common physical relationships can be catalogued along with methods for translating them into temporal constraints. The catalogue implemented by the author includes the generic relationships *enabled-by*, *disabled-by*, and *prohibited-by* as well as more specific relationships such as *supported-by*, and *attached-to*. These are translated into temporal constraints by simple macro substitution.

The specification process is not without difficulty. It is possible that a significant relationship among the parts or operations to be performed will be encountered which has not been previously catalogued. In that case the analyst must directly produce a formula which defines the admissible ordering of the operations involved. There is also the possibility that the analyst will overlook some significant constraint or erroneously include some unnecessary constraint. Some of these difficulties can be avoided by deriving temporal constraints from an analysis of certain significant states of the given task [Fox, 1987b].

The Derivation of Subtasks, Strategies, and Sequences

Given a task composed of a set of activities and a temporal constraint expression which governs their execution, it is quite simple to partition the activities into independent subtasks. First, extract all of the primitive constraints that occur in the given constraint expression (regardless of the logical structure of the formula). Then, for each primitive constraint $(x < y)$, construct the constraint $(x \bowtie y)$ which denotes the fact that x *is related to* y. The relation $\bowtie$ is reflexive, symmetric, and transitive, thereby satisfying the

requirements of an equivalence relation. Next, form the closure of the equivalence relation defined by $\bowtie$. Each of the resulting equivalence classes contains only activities that are related by some combination of ordering constraints. Moreover, there are no ordering constraints which cross the boundaries of the equivalence classes. Hence, the execution of the given task can be treated as the interleaved execution of multiple independent subtasks as defined by the equivalence classes.

Given a task composed of a set of activities and a temporal constraint expression which governs their execution, it is a non-trivial task to construct a feasible execution sequence. It is simple to demonstrate that the problem of determining the satisfiability of an arbitrary temporal constraint expression is NP-Complete and the process of determining an admissible first step is NP-Hard. However, some temporal constraint expressions are easy to analyze. Any task which can be represented by a conjunction of primitive constraints (i.e., a strict partial order) can be analyzed and sequenced by simple polynomial time algorithms. Although not every task can be reduced to such a simple representation, every task can be represented as the union of a set of conjunctive plans (i.e., a disjunction of conjunctions of primitive constraints) which together cover every admissible sequence of operations. Most algorithms which can be applied to conjunctive constraint expressions can be adapted to the more general disjunctive normal form.

Given an arbitrary temporal constraint expression, an equivalent disjunctive normal form constraint expression can be produced by purely algebraic means. Since this is an NP-Complete language, there are two hidden perils associated with this process. If simple, direct methods are used to create the disjunctive normal form, then the size of the resulting expression can be prohibitively large; if more sophisticated methods are used to produce the smallest possible disjunctive normal form, then the time required may be prohibitive. For many problems, careful use of heuristic methods reported previously [Fox 1987a] yield a reasonably small disjunctive normal-form constraint expression in a reasonable amount of time.

The disjunctive normal form of a given temporal constraint expression has several important interpretations. Like the original constraint expression, it still represents the task to be accomplished; it still represents the constraints over the the individual operations to be performed; and it still implicitly represents the set of admissible sequences for performing those operations. The added interpretation that can be applied to the disjunctive normal form is that each conjunctive clause can be interpreted as a *strategy* for performing the given task. This can be particularly useful to engineers responsible for designing and managing the execution of complex tasks. They can first focus on the constraints over the constituent operations. Then they can verify and refine the specification of the task by analyzing the admissible strategies which are produced by this normalization. Methods for analyzing the resulting strategies are presented elsewhere [Fox, 1987a].

The normalization and analysis described above serves only one purpose: the production of a set of constraints which circumscribe the admissible sequences over a set of operations and the transformation of those constraints into a form suitable for processes of sequencing. The sequence of operations can be determined off-line, prior to execution time, according to expected run-time conditions; or on-line, at execution time, according to actual run-time conditions.

A feasible sequence of operations under a single strategy can be derived from the process of pebbling the nodes of a directed acyclic graph derived from that strategy. Each step of the task is translated into a node of the graph and each primitive ordering constraint $(x < y)$ is translated into a directed arc from node x to node y. Pebbling is governed by a single rule: a node can be pebbled only if all of its predecessor nodes have been previously pebbled. An initial node, with no predecessors, can be pebbled at any time. By analogy, the set of steps that can be executed next in some state of the task correspond exactly to the set of nodes that can be pebbled next in the analagous state of the graph. The process is complete when all the nodes are covered. The set of all possible pebbling sequences is identical to the set of all possible execution sequences for that strategy. Simple methods have been developed which extend the pebbling analogy to the execution of a task composed of multiple strategies [Fox, 1987a].

Conclusion

A method for representing robotic assembly tasks based upon temporal constraints has been presented. Methods for deriving those constraints from enabling and disabling physical relationships have been summarized along with methods for factoring a task into independent subtasks, strategies, and feasible sequences of operations.

References

[1] Allen, J., Maintaining knowledge about temporal intervals, *Communications of the ACM* 26, pp. 832-843, 1983.

[2] deMello, L.S.H. and Sanderson, A., And/Or graph representation of assembly plans, in *Proceedings Fifth National Conference on Artificial Intelligence*, Philadelphia, Penn., pp. 1113-1119, 1986.

[3] Drummond, M., Plan Nets: a formal representation of action and belief for automatic planning systems, Ph.D. dissertation, Department of Artificial Intelligence, University of Edinburgh, 1986.

[4] Fox, B.R., A representation for finite serial processes, Ph.D. Dissertation, Department of Computer Science, University of Missouri, Rolla, MO, 1987.

[5] Fox, B.R., The derivation of temporal constraints, submitted to 1988 IEEE International Conference on Robotics and Automation, Philadelphia, PA, April 1988.

[6] Fox, B.R. and Kempf, K.G., Opportunistic scheduling for robotic assembly, in *Industrial Engineering, Selected Readings* E.L. Fisher and O.Z Maimon (eds), Institute of Industrial Engineers, Atlanta, GA, 1986.

[7] Fox, B.R. and Kempf, K.G., A representation for opportunistic scheduling, in *Third International Symposium on Robotics Research*, O. Faugeras and G. Giralt (eds), MIT Press, Cambridge, MA, 1986.

[8] Malcolm, C. and Fothergill, P., Some architectural implications of the use of sensors, in *Intelligent Autonomous Systems*, L.O. Hertzberger and F.C.A. Groen (eds), North-Holland, Amsterdam, 1987.

[9] Prenting, T.O. and Battaglin, R.M., The precedence diagram: a tool for analysis in assembly line balancing, *Journal of Industrial Engineering*, vol 15, #4, pp. 208-211, July-August 1964.

[10] Whitney, D.E., State space models of remote manipulation tasks, *IEEE Transactions on Automatic Control*, vol AC-14, #6, Dec. 1969.

N88-16417

ORIGINAL PAGE IS
OF POOR QUALITY

PLANNING ACTIVITIES IN SPACE

Kai-Hsiung Chang

Department of Computer Science and Engineering
Auburn University, Auburn, Alabama 36849-5347

0. ABSTRACT

This paper presents three aspects of planning activities in space. These include, (1) generating plans efficiently, (2) coordinating actions among multiple agents, and (3) recovering from plan execution errors. Each aspect will be discussed separately.

1. INTRODUCTION

An autonomous space station is required to formulate its own action plan after receiving a mission command. In order to accomplish this goal, a system that is able to generate action plans for various agents, coordinate actions among agents, and decide on recovery plans for execution errors will be required. This paper describes some research works of these areas. The author assumes that the reader already has a basic knowledge on planning.

2. A TWO-PHASE PLANNING STRATEGY

This approach would generate plans efficiently according to the goal requirements. In this strategy, a planning process is divided into two phases, goal analysis and plan generation. The idea of goal analysis is to reduce the fruitless search space at the start of the planning process and to provide a correct outline for the generation of plans. In the block stacking example of Figure 1, most human experts know that in order to build a structure, the lower part has to be built first and the lowest block has to be put on the table. With a simple analysis using these two heuristic rules, an expert can conclude quickly that (ON B C) should be achieved before (ON A B) and that C should stay on the table. If a planning system also adopts this heuristic analysis, the same conclusion can also be reached. We believe that this analysis is close to a human planning model and is more efficient for solving a problem.

In the plan generation phase, a goal-oriented hierarchical operator representation technique is used to avoid the time consuming operator searching process. Usually, a goal can be achieved by several different operators; but some of them may not be applicable at a specific instance, and some of them may have side effects that would cause problems later. Trail-and-error search process is used to select operators in most conventional systems. This time-consuming search process can be avoided. First, if each operator is named by the goal it would achieve, there is no need to search for operator candidates. Secondly, within each operator representation, a sequence of detailed

PRECEDING PAGE BLANK NOT FILMED

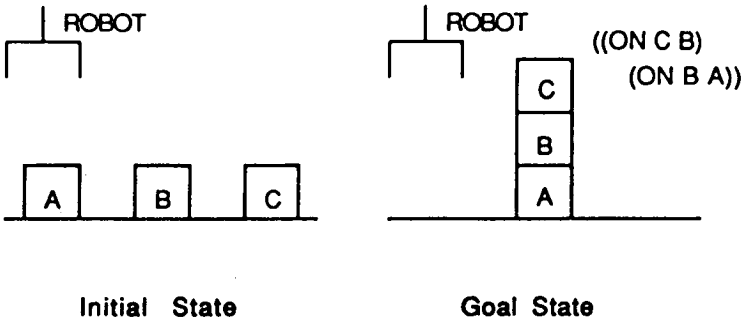


Figure 1 A simple block stacking problem.

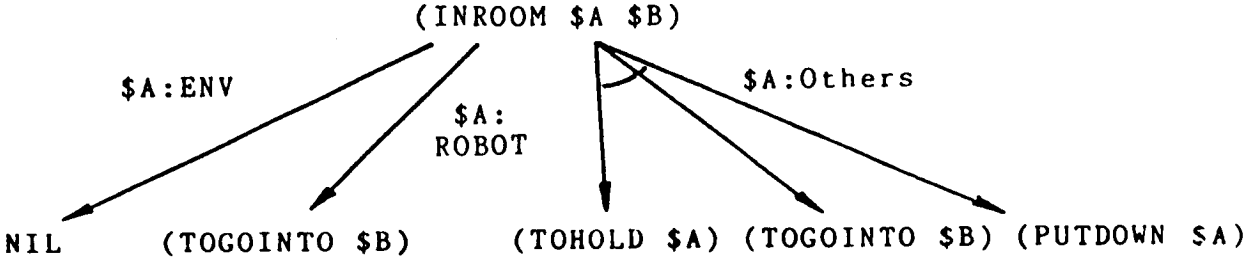


Figure 2 Operator hierarchy of (INROOM $A $B)

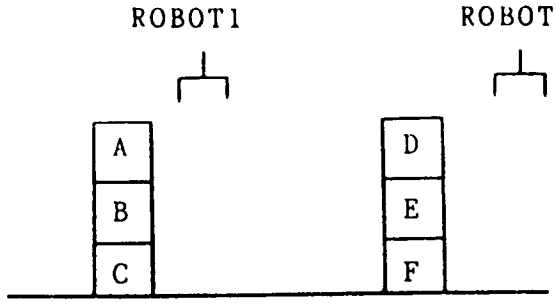
operators can be selected to satisfy special requirements of different situations. A sample operator hierarchy is shown in Figure 2. In this hierarchy, the goal is to move object $A into room $B. Since the name and the goal of the operator are identical, this representation is considered goal-oriented. In this example (INROOM $A $B) must be refined into a sequence of detailed operators before its execution. The sequence selection is determined by the requirements of the goal and the world state. Here, if $A is an ENV (environmental) object, like DOOR, which can not be moved, then nothing has to be done. If $A is a ROBOT, which can move, then the ROBOT just has (TO-GO-INTO $B). Finally, if $A is something else, then the ROBOT has (TO-HOLD $A), then (TO-GO-INTO $B), and then (PUT-DOWN $A). An abstract operator, like (INROOM $A $B), can be refined into more detailed operators by simple condition matching. Detailed examples and applications of this approach can be found in [1,2].

3. A MULTIAGENT PLANNING SYSTEM

In a multiagent environment, the resource sharing and action coordination must be managed carefully. This is critical to the success of an integrated system which involves multiple agents[3]. In our approach, three features have been proposed.

They are meta-level planning, agent-oriented dynamic task assignment, and breakable and unbreakable action sequences.

- Meta-level Planning: The purpose is to transform an original goal (problem) statement into a plan outline that is easier to pursue. The transformation includes grouping and ordering original goal components, adding new goal elements, and posting constraints. A typical example is shown in Figure 3. In the first step, the system groups subgoals according to resource. In the second step, it uses domain knowledge to determine the subgoal pursuing sequence in each group.



GOAL STATEMENT:

```
((ON B C) (ON A B)
 (ON D E) (ON C TABLE)
 (ON E F) (ON F TABLE))
```

Figure 3 A typical multiagent planning problem.

STEP1: Parallel Groups of Subgoals

```
(= ((ON B C) (ON A B) (ON C TABLE))
   ((ON D E) (ON E F) (ON F TABLE)))
```

STEP2: Ordered Subgoals and Groups

```
(= ((ON C TABLE) (ON B C) (ON A B))
   ((ON F TABLE) (ON E F) (ON D E)))
```

The equal sign ("=") of STEP1 shows that there are two parallel groups of subgoals, which can be pursued by different agents. The result of STEP2 shows the sequence of pursuing subgoals within a group. For example, the sequence of stacking the first block pile is (ON C TABLE), (ON B C), (ON A B).

- Agent-oriented dynamic task assignment: This is to find out what an agent can do at different times. The planner always tries to assign one or a team of free agents to be in charge of one group of related subgoals. The assignment is determined by the features and the status of agents, the requirements of the task, and the constraints posted during the meta-level planning. Actions of each agent are then generated accordingly. Normally, an agent works for its own subgoal groups. However, exceptional condition is allowed for an agent to do unanticipated tasks.

- Breakable and unbreakable action sequences: The idea is to distinguish the unbreakable actions that must be executed by the same agent from those breakable actions that can be executed by different agents. This provides (a) cooperative tasks between agents can be identified without difficulty, (b) agent utility can be improved, and most importantly, (c) cumbersome reasoning for concurrent actions is eliminated. Detailed report will be published in the near future.

4. EXECUTION ERROR RECOVERY

Normally, a plan must be carried out in a world whose behavior cannot be predicted exactly, so one must be prepared for failures during execution[5]. A system that is capable of handling such failures is presented. One point this system has made is that it modifies only those parts of a plan that is absolutely necessary. The planning process involves the hierarchical expansion of abstract goals (or actions) into detailed actions. This in essence, generates a tree structure (called expansion tree or plan tree) with the leaves as the primitive actions that constitute the final plan. In order to aid in the error recovery process, a second tree called the decision tree is used. This is similar to the one proposed in [4]. The nodes in the decision tree are in one-to-one correspondence with the decisions made during the construction of that plan. Each node in this tree has a two way pointer from it to the nodes in the plan tree, which was created as a direct consequence of its decisions. The error recovery process consists of error identification, classification, and recovery.

4.1 ERROR IDENTIFICATION

Two methods have been used to identify errors in the plan execution monitoring. They are condition-oriented and object oriented approaches.

CONDITION-ORIENTED APPROACH

Since the problem is to identify errors, one must look for violations of conditions that need to be true at different parts of a plan. Several conditions are considered. 1. Preconditions. They are predicates that must be true before an action can be executed. 2. Expansion conditions. These are the status of world on which the expansion of a node depends. 3. Decision Conditions. These are the decisions made during the planning process and are based on a predefined heuristic function.

With these condition classifications, the identification of errors can be accomplished by comparing the current world state to the conditions recorded in the decision tree.

OBJECT-ORIENTED APPROACH

All the objects involved in the domain can be classified as

ORIGINAL PAGE IS
OE POOR QUALITY

critical or non-critical. Normally, a non-critical object is either an environmental object on which none or limited actions can be performed, or an agent that can perform actions; a critical object is one on which actions can be performed. If an error involves a non-critical object, only local readjustment needs to be made. If the error involves a critical objects, but the predicate involved does not fall into the critical category (door locked is a critical predicate), then local readjustment is needed but changes need to be propagated. This is similar to the violation of precondition case. However, if the predicate involved is critical, as the locked door, then a major replanning is needed.

4.2 ERROR CLASSIFICATION AND RECOVERY

Before an error can be cleared from the "world", the system has to recognize the type of the error so that an appropriate modification can be taken. Four error categories are used in our system.

1. Non-critical error: The modification consists of going one level higher in the plan hierarchy and adding a subplan when an assumed condition has failed, or removing a subplan when the goal was already achieved. This will not affect the rest of the plan in any way. Most expansion condition violations fall under this category.
2. Major error, but not critical: It is handled just like category 1. But the rest of the plan might be affected. So any changes should be propagated along. Precondition and some expansion condition violations belong to this category.
3. Critical error: This requires the abandoning and the replanning of certain subplans. Any changes should be propagated. Decision condition violations fall under this category.
4. Unrecoverable error: No modification takes place and the execution is aborted. A typical example in the blocks world is the malfunction of the robot arm, which will prevent any further actions. Human operators must be informed to resolve the error.

5. REFERENCES

[1] Chang, K.H. and Wee, W.G.,"Planning with analysis", Proc. 2nd IEEE Conf. on Artificial Intelligence Applications, 1985, pp. 275-280

[2] Chang, K.H. and Wee, W.G., "A Knowledge-based Planning System for Mechanical Assembly Using Robots", IEEE Expert, 1988. (To appear)

[3] Georgeff, M.P. "The Representation of Events in Multiagent Domain", Proceedings AAAI-86, 1986, pp. 70-76.

[4] Hayes, P. J. "A representation for robot plans". Proc. IJCAI, 1975, Tbilisi, USSR, pp. 181-188

[5] Wilkins, D. E. "Recovering from execution errors in SIPE". Computational Intelligence I, 1985, pp. 33-45

ORIGINAL PAGE IS
OF POOR QUALITY

N88-16418

Intelligent Man/Machine Interfaces on the Space Station

Rodney S. Daughtrey
Research Associate, Johnson Research Center
Research Institute, Room A-11
University of Alabama in Huntsville
Huntsville, AL 35899
(205)-895-6217

ABSTRACT

The computer systems that will be resident on the space station will necessarily be large, complex configurations of software and/or hardware which must function in concert to maintain the operations of the mission. However, much of the complexity should be transparent to the user of those systems, allowing the user to concentrate as much as possible on the purpose of his/her interaction with the computer, rather than unnecessary detail.

This paper addresses some important topics in the development of good, intelligent, usable man/machine interfaces for the space station. These computer interfaces should adhere strictly to three concepts or doctrines: generality, simplicity, and elegance (just as the programming language code below these interfaces should follow). Generality refers to the commonality of usage and the similarity of form and function that should predominate all the interfaces, so that the user is provided with computer working environments that appear and perform in much the same way. Simplicity is obviously desirable, but not a concept to be taken lightly. It is very important that the interfaces simplify operations wherever possible (and desirable), and make intelligent inferences about the intent of the user to save time and unnecessary attention to detail. Finally, the elegance of the interfaces should be of great concern. The interfaces should be as concise as possible, exhibiting the "principle of least astonishment".

The author will also discuss the motivation for natural language interfaces and their use and value on the space station, both now and in the future.

AI provides an extremely powerful tool with which to think about and develop software applications to run on the space station. But, without well-thought-out, intelligent, truly usable man/machine interfaces to harness this asset, much of this power will be lost.

PRECEDING PAGE BLANK NOT FILMED

INTRODUCTION: LEVELS OF COMMUNICATION AMONG HUMANS AND COMPUTERS

Communication is an interesting subject. Whether we communicate with other humans, machines, or some other entity, we do so to achieve some end. With computers, we attempt to get them to do things for us, (usually) saving time and effort on our part. It only makes sense, then, that getting them to do something should be made to be as effortless and logical as possible. How should we design them in order to maximize ease of use? Should we make them like us?

Figure 1 shows a comparison between human levels of communication and machine levels of communication (as used by humans).

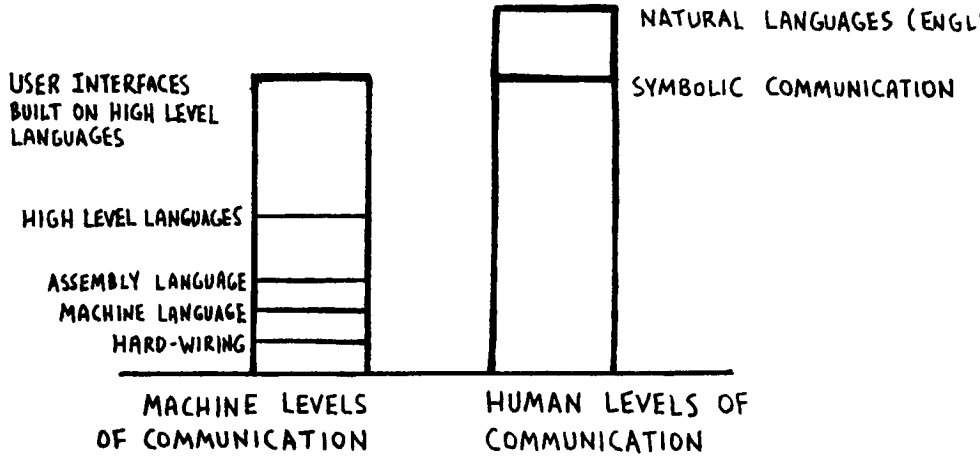


Fig. 1. Comparison of human and machine communication levels

At the bottom of machine communication we have hard-wiring, or direct communication with the physical parts making up the computer. This was done with the ENIAC computer in the late 1940's. The important thing to realize here is that there is no level of abstraction of input, and therefore no user interface, between the human and machine at this level: the human operator is responsible for explicitly determining the actual flow of electricity through the machine. This may seem rather silly today, but it was at one time the price one had to pay for "automated" computing.

Machine language soon followed hard-wiring and abstracted most of the physical aspects of programming a computer, thereby becoming the first user interface. Then came assembly language, which gave mnemonic names to machine language instructions and provided a slightly more forgiving format for writing programs. High-level languages such as FORTRAN and LISP appeared on the scene next, making programming a much simpler and less tedious task and allowing programmers to be more productive and to write programs that were less machine dependent than those in the past.

Much of the software written today makes use of the more friendly and intuitive input methods that have been developed for today's computers (the mouse, user menus, icons, multiple windows on the screen, etc.). This software often reaches a level of communication that humans use in their everyday life, a level the author calls "human symbolic communication". Programs are beginning to interact with the user in the same way that a human interacts with and responds to symbolic stimuli in the real world (traffic lights, road signs, a telephone ringing), making the programs easier to learn and use.

Finally, interfaces which communicate with the user in plain English are currently under a great deal of study and development. Some progress has been made, but many large, difficult problems still remain. The importance and relevance of natural language interfaces to the space station will be discussed later in this paper.

ESSENTIAL ELEMENTS OF A USER INTERFACE

Regardless of the type of interface (be it graphical, natural language, or some other), three principles (in the author's view) should predominate its form and function: generality, simplicity, and elegance [2].

What was the author's motivation behind concentrating on these three ideals? It stems from the fact that these ideals are (usually) the three main goals to strive for when writing computer programs. A programmer's ultimate objective should be to write programs that are easy to understand, easy to modify, and easy to use. Since these principles are also what a good user interface should embody, the same philosophies should apply.

The principle of "generality" might better be termed the principle of "non-specificity". All the user interfaces on the space station (or as many as is feasible) should look, be used, and perform in much the same way, even though they might perform vastly different tasks. These interfaces should each be as non-specific as possible, so that features, commands, and utilities resident in one interface will most likely appear in all the interfaces. This cannot always be the case, of course; some features and commands in one interface may not even make sense in another context. If possible, the similarity of features and commands should extend both functionally and graphically across interfaces; in other words, not only should features common to more than one interface perform the same way, they should even appear in the same form on the screen. The main idea behind generality, of course, is that once a user becomes familiar with one interface, he/she would be able to learn to use other interfaces in much less time and with considerably less effort.

Incorporating simplicity into a user interface may seem

rather an obvious objective, but a fine line lies between keeping things simple and lessening the functional power of the interface. Einstein may have said it best: "Everyhing should be made as simple as possible, but not simpler" [1]. At any rate, a good rule to follow might be that wherever unnecessary detail can be suppressed, it should be, but not at the functional expense of the user. A simple example involves the user input of several parameters for some computing task (a statistical program, say). If the user needs to run the program more than once, he/she definitely should have the program option of supplying the same parameters as on the first execution. Not having this option would violate the concept of simplicity; either the user would be subjected to unnecessary detail (i.e. the program forces the user to input all the parameters again), or the user would lose functional power (the program only allows the same values to be used again on subsequent executions).

Implementing elegance into an interface is a highly subjective task. Elegance is really a combination of good taste and common sense, and although it may mean different things to different programmers, a few guidelines do exist.

Probably the main practice that should be followed is the adherence to the "principle of least astonishment". For those unfamiliar with this principle, it states that the programmer should devote considerable attention to the naming of commands, features, program options, etc. in order to make as obvious as possible what operation or concept is meant by that name. Put simply, "say what you mean". The idea is that the user should be "least astonished" at what the command, feature, or program option implies. Although this is stated tongue-in-cheek, anyone who has worked with software in which the keystroke sequence "CTRL-J CTRL-M ESCAPE R47" was needed to save a file, rather than, say, "CTRL-S", can appreciate its importance.

WHAT SHOULD USER INTERFACES ON THE SPACE STATION BE LIKE?

Are these graphical, symbolic user interfaces discussed earlier the way to approach user interfaces on the space station? Can we develop even higher level natural language interfaces which can communicate with us in English and truly understand our instructions and intentions? Assuming we can, when and under what conditions should we develop them? To answer these questions, we need to identify the motivations for both types of interfaces and the advantages and disadvantages of each.

With graphic-oriented interfaces, speed of use and conciseness of expression are definite advantages over natural language interfaces. Users can accomplish tasks much faster using input devices like the mouse, menus, windows, and other similar features. Tasks that would have to be specified in sentence form in a natural language interface could be effected

with a single mouse click or some keystroke.

Natural language interfaces, however, are not without their advantages. With graphic-oriented interfaces, it is sometimes very difficult or impossible to perform some task that was unforeseen at the time of the building of that interface. Given a powerful enough natural language interface, the user is given the power of completeness of expression: he/she can specify any task needed to be performed through English text (again, assuming that such an interface is implementable). Another argument for natural language interfaces is that there is virtually nothing to learn about the interface for the user (assuming he/she knows English).

Considering the above, it seems logical to make the following conclusions:

1) Both graphic-oriented and natural language interfaces should reside on the space station (assuming powerful enough natural language interfaces can be developed).

2) Graphic-oriented interfaces should be used in conjunction with tasks that are well understood and bounded in terms of previously foreseen needs and capabilities.

3) Tasks characterized by complexity and possible unknown but essential requirements should have both graphic-oriented and natural language interfaces (again assuming their existence). The graphic-oriented interface would be the interface normally used, with the natural language interface used for any appropriate situation.

4) As the space station program develops further, more natural language extensions to the existing graphic-oriented interfaces should be developed to accomodate the greater variety of people that will be participating in the program.

Without a doubt, the space station will contain an amazing amount of computational power, and harnessing that power and making it usable should be a huge consideration. The complexity of the entire operation and the fact that a large variety of people must work together in the same environment should demand a great amount of forethought about the man/machine interface.

REFERENCES

1. Minsky, M. The Society of Mind. Simon and Schuster, New York, New York, 1985, p. 17.

2. Schneider, G.M, and Bruell, S.C. Advanced Programming and Problem Solving in Pascal. John Wiley and Sons, New York, New York, 1981, p. 52.

N88-16419

Interfacing the Expert: Characteristics and Requirements for the User Interface in Expert Systems

Andrew Potter
General Digital Industries
7702 Governors Drive
Huntsville, AL 35806

ABSTRACT

Because expert systems deal with a new set of problems presenting unique interface requirements, special issues requiring special attention are presented to user interface designers. The prime issues addressed in this paper are 1) *External Knowledge Representation:* how knowledge is represented across the user interface, 2) *Modes of User-System Interdependence:* advisory, cooperative, and autonomous, and 3) *Management of Uncertainty:* deciding what actions to take or recommend based on incomplete evidence.

INTRODUCTION

The user interface is critical to the effectiveness of expert systems. Although its importance in securing user acceptance is well known [3], the issue goes beyond concern for acceptance. The interface affects overall system performance. This is because an expert system's ability to solve real problems depends on the accuracy, not only of its knowledge base, but of the factual context established during interaction.

Because many expert system development efforts are begun as feasibility studies, the user interface is often neglected [5]. But if the system is to be integrated into the workplace, the interface is essential to its success. And to construct a finished product, a significant portion of the development effort must go into the interface. Bobrow, Mittal, and Stefik [2] indicate it is not unusual for the interface to account for one-third to one-half of the code comprising an expert system.

Advanced technology in support of the user interface is plentiful. 'High bandwidth' techniques such as windows, icons, and direct manipulation have come to typify the state of the art user interface. Bringing these techniques to bear on particular applications, however, is not easy [1]. Advanced interface techniques are no guarantee of a usable system.

Use of techniques must be guided by higher level concepts, such as intuitiveness, credibility, and locus of interaction control. Techniques focus on the interface mechanisms; concepts provide the criteria for selecting and melding them into a coherent, usable system. This paper attempts to identify a set of

PRECEDING PAGE BLANK NOT FILMED

general characteristics of expert system use interfaces which set them apart from the interfaces of conventional applications.

That intelligent systems in general differ functionally from conventional systems may be seen as a continuation of a trend. It has been observed that the tendency towards increased automation within society has caused a shift in the human's role from operator tasks involving perceptual and motor activities to cognitive tasks emphasizing monitoring and evaluation activities [4]. As systems become more intelligent, this trend is taken a step further. Expert systems undertake to perform cognitive activities previously reserved for humans, and they do so in domains previously beyond the purview of automation. This causes the burden of decision making responsibility to shift from the user to the system. That people would look to machines for the kind of support offered by expert systems is in itself a change in both the user's role and the system's role.

EXPERT SYSTEMS AND THE USER INTERFACE

Several aspects of expert systems are significant in levying unique requirements on the user interface, including external knowledge representation, modes of user-system interdependence, and management of uncertainty. These characteristics and their corresponding user interface concepts are summarized in Figure 1 and are discussed in detail below.

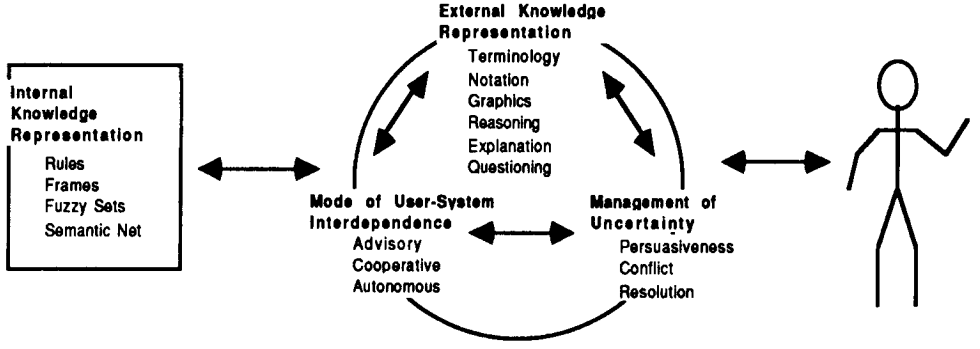


Figure 1: Characteristics of Expert System User Interface

External Knowledge Representation

In designing an expert system, it is helpful to distinguish internal and external representation of knowledge. Internal knowledge representation pertains to how facts, theories, and beliefs are mapped for the purpose of internal manipulation (e.g., frames, objects, rules, and fuzzy sets). External representation refers to how knowledge is represented across the user interface. It is the terminology, rhetoric, notations, depictions, and styles of interaction associated with the problem domain.

External knowledge representation is important in making the expert system intuitive and credible. Intuitive software minimizes the learning required to use the system by building on the user's previous knowledge and expectations [8]. Because interaction with an expert system tends to be a knowledge intensive activity, using the system demands more than familiarity with basic operations such as keyboard commands, menu selections, function keys, etc. For an expert system to be intuitive, it must exploit the user's expectations as to how ideas are organized and expressed within the system's problem domain.

External knowledge representation can be used to support intuitiveness and credibility in several ways: 1) terminology, notation, and graphics should be modeled on the target domain; 2) reasoning should be represented in human terms rather than machine terms; 3) explanations should be explanatory, rather than a traceback of activated rules; 4) questioning should be progressive rather than arbitrary. Also, because interaction errors may be cognitive misunderstandings rather than syntactical typos, they may not be readily detectable, and the user, rather than the system, may be better positioned to notice them. The system may support recovery from such errors by permitting the user to alter the findings of the system by subtly changing the context.

Mode of User-System Interdependence

The mode of user-system interdependence influences the amount and complexity of information exchanged between the user and the system. It also determines the locus of decision-making responsibility, and along with this, the locus of interaction control. There are three modes of interdependency: advisory, cooperative, and autonomous:

1) Advisory expert systems interact with a user who has no expertise in the system domain. While these systems may employ a high level of experise internally, their interactions must be gauged to the user's level. This may require the system to resort to incomplete analogies, over-simplifications, and loosely defined terminology. The system has prime responsibility for gathering information needed for reliable results.

2) Cooperative expert systems support experts in solving problems in their area of expertise [6]. The system may be subordinate to the user, so that the user is in control of interaction as well as decision making [7].

3) Autonomous expert systems are capable of selecting and executing processes without user intervention. The user functions not as a source of facts to be added to the context, but instead as an evaluator, monitor, and manager [4].

Uncertainty Management

Some research indicates that use of numerical probabilities in expressing uncertainty is ineffective because users (as well as experts and knowledge engineers) do not easily understand them [5]. But the problem goes beyond this. Uncertainty must be managed in terms of how persuasive the system is in presenting its conclusions. From the user interface perspective, the issue is not so much one of determining what conclusions can be inferred from the factual context, but of determining what advice to give or what actions to take on the basis of conclusions reached. Consider the following:

1) There is a 75% chance of rain today
2) It will probably rain today
3) Take an umbrella!

These statements could come from a hypothetical weather expert. The first two statements accomplish essentially the same thing: they leave it up to the user to decide how seriously to take the threat of rain. They simply address the question of whether it will rain today; they do not, unlike the third statement, presume to tell the user what to do. This may be acceptable as long as the issue is one of relatively trivial importance. Suppose the example instead involved a life-threatening disease but the probability were only 10% instead of 75%. The odds are much lower, but the stakes much higher. It might be unsatisfactory to simply tell the user the odds in this case. The interface must tread the narrow line between compelling the user to action and causing undue alarm.

Another aspect of uncertainty management is conflict resolution. Depending on the mode of user-system interdependency, presenting multiple conflicting conclusions for user consideration may or may not be acceptable. With cooperative systems, the user accepts final responsibility for resolving conflict. With advisory systems, however, the user may be unequipped to choose among conflicting alternatives. Advisory expert systems that provide users with a list of possibilities in lieu of definitive results may succeed in reducing the developer's liability, but the effectiveness of the system is compromised.

CONCLUSION

The ability of an expert system to solve real problems depends significantly on the accuracy, not only of the knowledge base, but of the factual context as well. The context cannot be established accurately if the user fails to consult the system as intended, or if the system fails to support the user in conveying the appropriate information. For expert systems to provide this support, careful attention to the external knowledge domain, the mode of user-system interdependency, and the management of uncertainty is required.

Because of the importance of the user interface, designing effective expert systems requires developers to do more than simply deal with the knowledge comprising the problem domain. For effective external knowledge representation, it is necessary to consider the way experts and users view the domain, and to accomodate these perspectives in the user interface. Selecting the proper mode of user-system interdependence requires that the developer examine the demands the system makes of the user, the demands the user makes of the system, and how these demands may be met. With respect to uncertainty management, it is necessary to fully grasp the implications of any conclusions reached in terms of their intended effect on the user.

REFERENCES

1. Berry, D. & Broadbent, D. E. Expert Systems and the Man-Machine Interface, Part Two: The User Interface. *Expert Systems*, Feb.,1987, 4(1), 18-28.

2. Bobrow, D. G., Mittal, S., & Stefik, M. J. Expert Systems: Perils and Promise. *Communications of the ACM*, Sept. 1986, 29(9), 880-894.

3. Gaschnig, J., Klahr, P., Pople, H., Shortliffe, E., & Terry, A. Evaluation of Expert Systems: Issues and Case Studies. In F. Hayes-Roth, D. A. Waterman, & D. B. Lenat (Eds.), *Building Expert Systems*. Reading, Mass: Addison-Wesley, 1983.

4. Greitzer, F. L. Intelligent Interface Design and Evaluation. *Proceedings of the Conference on Artificial Intelligence for Space Applications*, Nov. 13-14, 1986. Sponsored by NASA Marshall Space Flight Center, Huntsville, Alabama, and the University of Alabama in Huntsville.

5. Kidd, A. L. & Cooper, M. B. Man-Machine Interface Issues in the Construction of an Expert System. *International Journal of Man-Machine Studies*, 1985, 22, 91-102.

6. Niwa, K. A Knowledge-Based Human-Computer Cooperative System for Ill-Structured Management Domains. *IEEE Transactions on Systems, Man, and Cybernetics*, 1986, 16(3), 335-341.

7. Shortliffe, E. H, Scott, A. C., & Bishoff, M. B. ONCOCIN: an Expert System for Oncology Protocol Management. *Proceedings of the 7th International Conference on Artificial Intelligence*, 1981, 876-881.

8. Stahl, B. *UIMS: A Guide to Designing Friendly User/Computer Interfaces*. Oakland, California: The Interface Design Group, 1986.

N88-16420

Space Languages

Dan Hays
Psychology Department/Cognitive Systems Laboratory
The University of Alabama in Huntsville 35899

Abstract. Applications of linguistic principles to potential problems of human and machine communication in Space settings are discussed. Variations in language among speakers of different backgrounds, and change in language forms resulting from new experiences or reduced contact with other groups need to be considered in the design of intelligent machine systems.

The languages that people and machines will use in near-Space settings will be about the same as the ones that they use on Earth, at least in the short run. These languages, and their users, will operate according to exactly the same principles of linguistic structure and change, whatever the setting.

So far as language use by humans interacting with other humans, for the most part we do not have to worry about it for missions in the near future, whether the humans are working in Space or are members of ground organizations which interact with orbital or planetary locales. People are such capable users of language, and the kind of language we have as a species is so functional, that it need not concern us so much as, say, the reliability of energy transduction and distribution (which is currently seen as critical and somewhat problematic) or the psychological and social effects on Space residents of isolation, restricted quarters, and scheduling pressures (which still do not appear to concern the U.S. space effort as much as they should).

Even so, our complex and in many ways delicately tuned language faculty is a *very important human sub-system,* so to speak. For this reason alone it may bear some examination as we make plans for Space residence. Other more concrete reasons include the following:

- Several human languages will be spoken in near-Space, in some cases in the same locale. Even when the speakers are bilingual, problems can arise when assumptions differ, owing to differences in the native language cultures.
- Within the same basic language, different occupational or regional dialects may be spoken, allowing not just misunderstandings of terminology or nuance, but also invoking potentially troublesome group identity conflicts.
- Errors and misinterpretations occur occasionally because of economy of expression and the partial, general nature of reference in human language.
- Yet other errors of communication, either between people, or in data entry or video-screen perception, seem to occur because of lapses of attention or other "low-level" processing problems.

When it comes to humans interacting in information-rich ways with machines, or even sufficiently advanced machines interacting with other machines, linguistic issues are much more problematic. Since most foreseeable Space habitats will be heavily mechanized, and may be fairly heavily automated, characteristics of human language, and indeed of languages suitable to machine interchange, should concern the designers of complex devices and computer-based information systems. This is true whether these devices are "intelligent" in the usual senses of reasoning logically or heuristically, using analogies, profiting from experience, and so on, or

PRECEDING PAGE BLANK NOT FILMED

just are *sensitive to situations and interactions.*

Communication using some subset of human language has been of concern to computer system designers, perhaps especially during this decade. Diagrammatic and other graphic display of information is more frequently used, also, as machine capabilities increase, and as computer professionals become more concerned for human interfacing (compare, for example, computer displays in the first SpaceLabs with suggestions for the next U. S. space station). As time goes on, we may expect:

- more *actual language analysis,* both of human input, and in generating output from computers, rather than just recognition and delivery of set phrases with slots for variable information,
- developments in *graphic and pictorial* information exchange,
- confrontation and exploration of issues of *semantic analysis* that is rich enough and flexible enough to mimic human understandings, impacting the structure, for example, of content-linked databases or knowledge bases,
- more explicit appreciation of *'pragmatic' and other socially relevant levels* or aspects of language usage.

Advances have been made in robotic devices. But we have hardly explored either their

- social capabilities [4] [8] or symbolic conventions for the informational exchanges that should occur between humans and devices, or among autonomous action devices [7].

Though it was suggested above that "space languages" will not be much different from ordinary Earth languages, really distinctive communication conventions might either be designed or evolve.

- Signaling conventions might evolve or be designed for special situations in extraterrestrial work. For example, gestural or iconic signs useful for work outside of spacecraft seem to exist already and should undergo development. Location markers, whether visual or in some other medium, may be conventionalized for autonomous devices and space craft.
- It is possible that certain conventions might be useful for interchanges among humans that speak different Earth languages.
- If remote sites are inhabited by people, they will almost certainly diverge somewhat in language terminology and possibly in language form as time goes by, just as languages on Earth change.
- Since the beginning of near-Space exploration by people, distinctive jargon has developed (some of it widely known) among Space workers.

The reality of language change will inevitably concern persons who design or work with computing devices where knowledge is stored and communicated via ordinary language. At the very least, new terminology and descriptions of new objects must be incorporated into knowledge-rich systems; and their data structures and interpretation routines must be flexible enough to allow for variation and approximation.

Language Variation

Linguists say that human language varies *synchronically,* at a given time, when we note hundreds of distinct languages and many more dialects; or *diachronically,* over periods of time. References to historical linguistics include [1] and [9].

We are familiar with the phenomenon of new terms coming into common use, and the dropping out of others. A number of processes can be involved. A term might be borrowed from another language ("détente"), or made up from existing parts ("television"). Metonymy, or referring to something by the name of something closely associated, is another ("a Bordeaux" for a wine made in that region, or "a Winchester" for a kind of disk drive or rifle). Other kinds of variation include shifts in meaning, where a term comes to have a related or even an opposite sense; shifts in syntax, for example, in the present instability of adverbial suffixes in English; change in performance style, and so on.

These changes take place against a background of great concurrent stability in a language, which must be predictable for communication. Nevertheless, and sometimes to the dismay of academics and newspaper columnists, language changes at all levels. Some changes are certainly prompted by situational novelty, or contact with others who speak differently; others seem almost intrinsic to language as a complex system. For example, some tendency to simplification of expression, at least for matters of frequent usage, may relate to a basic impulse of cognitive simplification. Exact kinds of change are difficult to predict. However, others follow precedent or cultural custom. For example, both English and Japanese have borrowed technical terms from other languages, though in somewhat different ways. Chinese languages have more frequently constructed new terms by metaphorical extensions of native morphemes.

The dynamic nature of language, together with the fact that Space workers will be confronting situations that Earth planners might not be able to envision in detail, argue for setting up computer-based information systems with some flexibility for growth and modification.

A potential source of tension in future machine knowledge technology in Space is engendered in the contrast between the changeable nature of human language, and the predilection of machine systems, and probably 'rational' systems in general, for fixity of reference. Put another way, any adequate means of representing "meaning" for machine systems must also handle "change in meaning".

Having expanded or modified meanings in machine processing implies that all concerned parties, whether human or electronic, must have a way of learning about the extensions or adjustments that are relevant to their activity. This would apply to occasionally interacting knowledge bases within one setting, such as a space station or small ground facility; and it would be of major importance to autonomous mobile devices, which must confront and symbolize novelty frequently, and which should be able to relay some of this information to companion devices and to people.

If outposts or settlements of humans come to exist in remote places outside the Earth, their language usage will certainly diverge. Interestingly, the isolation may slow the rate of language change, since contact with speakers of variant languages will be minimal. In these settlements, change would presumably stem mostly from situational novelty, intrinsic language characteristics, initial pool of variation among the language users, and chance elements.

References to synchronic language variation, especially sociolinguistic factors,

include [2], [5], [10], and [14].

Languages and Groups

We can often identify a person's important group memberships by the way the person speaks or writes. The basic language used will usually indicate society of origin. Saying "eh..." in a certain way means the person comes from Canada. Using certain technical slang and trailing sentences off will identify an aerospace technical professional. The notion of a "language", which is verbal, can be extended to the more general concept of a "semiotic", or a system of signs that may also be nonverbal. Manner of gesturing, for example, can reveal group allegiances.

The relation of language or semiotic systems to groups and their culture should be of concern in the design of machine intelligence systems in several ways. First, meanings may be clustered in certain ways in a given group [11], or assumptions about what is implied by certain statements may vary with the culture. Additionally, and somewhat more subtly, proprieties and niceties of presentation will probably also vary from group to group. (This point holds not just for language groups but for occupational groups, who may for example prefer different kinds of computer displays or even have feelings about the proper degree of terseness in communication.)

The conditioning of meanings by culture is by no means understood in detail, nor is the related matter of the exact effects of language on thought (and vice versa). Language and group-based differences in assumptions and implications are certainly real, though not always transparent to the participants. 'Dialectal' differences in understandings between persons who speak the same language but have different backgrounds and group-based goals, may be especially hard to detect. As a partial remedy, perhaps heuristic "expert systems" could be prepared to advise Space workers who have to deal with persons from other groups.

It should be mentioned that, although comprehensive machine translation between languages has to be at least as difficult as the analysis of a single human language, interest in this enterprise has revived over the past decade [13].

Special-Purpose Semiotics

Short of full-scale language translation by machines (a task that in its fullness may be better handled by people who spend some time interacting in another culture), we might settle for machine aids to some of the language problems that may arise in Space work.

It may be possible to devise small sets of pictorial or acoustic symbols for important matters, that can be understood with little familiarization by persons from diverse backgrounds. However, one should note that pictorial conventions to some extent differ between cultures, as apparently also do "acoustic icons" [6].

Since language processing is in fact very difficult and demanding of computer and machine storage resources even when fairly well understood, some computational linguists have aimed to characterize *sublanguages* for special content areas. Montgomery and Glover, for example [12], report on a sublanguage for describing events in space missions.

In planning for the entire machine/human communication arrangements for

orbital habitats or planetary bases, it is necessary to consider the physical arrangements as well as functional importance of matters that are communicated about. Use of sound is very natural on earth, for example, but acoustic signals must be relayed electronically when humans are in airless environments, competing with other information that must be delivered and possibly running into problems of reliability. Informally developed gestural languages are easy for humans to use, as Roger Brown has shown [3], but may be constrained by protective clothing in some situations.

1. Anttila, R. An Introduction to Historical and Comparative Linguistics. NY: Macmillan, 1972.
2. Bauman;, R. & Sherzer, J. (Eds.). *Explorations in the Ethnography of Speaking.* NY: Cambridge U. Press, 1974.
3. Brown, R. *Social Psychology: the Second Edition.* NY: Free Press, 1985.
4. Cohen, P. SRI Technical Report, 1987.
5. Fishman, J. *Advances in Sociolinguistics.* The Hauge: Mouton, 1972.
6. Hays, D. "Vocal Iconics". Auburn: SECOL, 1986.
7. Hays, D. "Social Robotics for Exploratory Missions." Seattle: Workshop on Planning for Autonomous Mobile Robots, 1987.
8. Hays, D. "Trans-System Linguistics." In Battistella, E. (Ed.) Papers in Computation and Cognitive Science. Bloomington: Indiana Linguistics Club, 1985.
9. Hoenigswald, H. Language Change and Linguistic Reconstruction, Chicago: U. Chicago Press, 1960.
10. Jesperson, O. *Mankind, Nation and Individual from a Linguistic Point of View.* Bloomington: Indiana U. Press, 1964 (from Unwin, 1946).
11. Lakoff, G. *Women, Fire, and Dangerous Things: What Categories REveal About the Mind.* Chicago: U. Chicago Press, 1987.
12. Montgomery, C. & Glover, B. "A Sublanguage for Reporting and Analysis of Space Events." In Grishman, R. & Kittredge, R. (Eds.). *Analyzing Language in Restricted Domains.* Hillsdale, NJ: Erlbaum, 1986, pp. 129-162.
13. Nirenberg, S. (Ed.). *Machine Translation.* NY: Cambridge U. Press, 1987.
14. Trugdill, P. *Sociolinguistics.* NY: Penguin, 1983.

N88-16421

An Innovative Workstation

James Villarreal
Artificial Intelligence Section / FM7
NASA Lyndon B. Johnson Space Center
Houston, TX

A SBIR (Small Business Innovative Research) contract was awarded to Analytics to develop a system which uses the faint magnetic fields generated by the mental processes of the brain as another route for computer control. The emphasis of this project is not on the direct control of a computer workstation by reading the operator's EEG signals, but to control the workstation, or better, to anticipate the user's needs with the combination of eye data, EEG data and task or workload data.

Analytics, through a previous SBIR, developed a workstation which used the operator's eye movements and position to determine the placement of the cursor on a computer screen. This paper will first provide a brain wave sensing technology overview and an introduction into the known rhythms or signals generated by the brain. This will be followed with a descriptive explanation of OASIS (Ocular Attention Sensing Interface System) and its intended integration into the proposed testbed.

Introduction

With the ever growing computer processing speed comes the growing informational exchange between man and machine. One interface design aspect deals primarily with the presentation or selection of information to the user. Another interface design aspect concentrates on control mechanisms such as the mouse to manipulate the data available to the user. This project's basic foundation is that both aspects should be addressed simultaneously.

Most advanced man machine interfaces are directed toward the cockpit. Complex machines have been developed to simulate the actual sensations and perceptions experienced by a pilot during flight. Should advanced interfaces be confined to just the super cockpit? Could such interfaces benefit the scientist or engineer in recreating known concepts? Clearly, many scientific problems, especially those that can be represented in three dimensions, would benefit by a greater interaction with the computer.

PRECEDING PAGE BLANK NOT FILMED

Electrical/Magnetic Sensing Techniques Overview

Electroencephalography (EEG) is probably the most familiar recording technique. EEG's are routinely used by the the medical community in diagnosing certain sleep or mental disorders. Also, the EEG along with heart rate and perspiration detection instruments are used by interrogative polygraphy or colloquially known as "lie detection". Alpha rhythms were first discovered by Berger (1929). It is interesting to note that alpha rhythms were discovered with galvanometers (a magnetic sensing device). It did not become feasible to record alpha rhythms until the introduction of the vacuum tube amplifier in the 1940's. A major problem with the early vacuum tubes were their inherent noise. Today's solid state low-noise amplifiers are capable of amplifying signals from DC up into the kilohertz regions with little degradation in the signal. In fact, the signals amplified by these devices are so clean that the principle noise source now comes from the neighboring mental activity of the area under investigation. The signals being detected by the EEG scalp probes are not only from the activity directly under the probe but also from other distant sources. EEG readings then are the average effect of several activity sources.

The signals seen by the EEG are frequencies which range from between 1 cycle per second to 13 cycles per second. Different frequency bands represent different states a person is undergoing. An EEG can easily determine whether a person is at rest, daydreaming, sleeping, or mentally active.

Probably the most familiar type of EEG signal is the alpha rhythm. Its range is generally between 8 to 12 cycles per second. An alpha rhythm is characteristic of a person in a relaxed state with the eyes closed but not necessarily sleeping. Its signal is greatest in amplitude over the posterior portions of the head, and it occurs in spindles with varying durations. The spindles can be inhibited by having a person open his eyes. This should indicate that the alpha rhythm is directly correlated with visual activity. However, after a person has had his eyes open and slips into a boring situation, it is possible to have the alpha rhythms reappear even with the subject's eyes open.

Another interesting EEG reading is the beta rhythm. It is characteristic of signals of 13 cycles per second and above. Beta activity is present when an individual has her eyes open and is mentally active. The signal is present in the anterior quadrants of the head.

Theta rhythms are characteristic of signals in the frequency band between 4 and 7 cycles per second. These rhythms are found in both adults and children under emotional stress. The delta rhythms are present during sleep and are in the 1 to 3 cycle per second frequency range. Delta rhythms appear during sleep and characterize the various sleep levels. Due to their dependence on the sleep state, they are not useful for this project.

Other signals which can be detected by EEG's are those that are evoked by certain stimuli. Evoked potentials, then, are time-locked to specific events. Because of their direct relation to cognitive factors the N1, P2, and P3 are particularly interesting to this project. The identifying characteristic for the evoked potential is easily understood by noting that the preceding letter signifies whether the signal is negative (N) or positive (P). The number following the letter indicates how long after the stimulus is given that a response is expected. Therefore, for N1 the signal is negative and occurs 100 milliseconds after the stimulus.

First discovered by Sutton (1963) is the classic P300 (P3) phenomenon. The fact that a component of the auditory related potential occurred about 300 msec after an unexpected stimulus was quite exciting This led to more discoveries of other event related responses. Due to the low frequency characteristic of P3, it is useful only in slowly developing situations. P3 is evoked by surprising or unexpected occurrence of both visual and auditory stimuli. However, for odd visual stimuli, the response is usually more on the order of 400 msec. Its long reaction time can be understood when we imagine a confrontation with a surprising situation ... we absorb the stimulus ... we analyze the stimulus ... we detect an anomaly ... we react.

The N1 evoked potentials are activated by auditory stimuli. Magnetic studies by Pellizone (1984) showed that the source of N1 is in the auditory cortex. It has been demonstrated that the amplitude of these signals are strongly related to the subject's level of attention. Unfortunately, the characteristics of this signal vary across individuals.

Almost anything stated of N1 is also true for P2, except that far less is known of P2 than N1. Magnetic studies of this phenomenon have shown physically separated activity sources within the brain.

With every electrical field there is an associated magnetic field. Unlike EEG which requires that probes be placed on the scalp, magnetoencephalogram (MEG) sensors are placed reasonably close to the scalp but not required to touch. The principle advantage this offers is the elimination of unwanted "skin" noise which poses a large problem for EEG recordings. As was indicated earlier, both theoretical

and experimental EEG studies have shown that different tissues and bone surrounding the brain attenuate or 'smear' the potentials that reach the scalp (Geisler and Gerstein, 1961; De Lucchi et al., 1962; Cooper et al., 1965). However, MEG studies have demonstrated that concentric layers in the head do not affect the magnetic fields produced by sources (dipoles). Therefore, MEG is not subject to 'smearing' and can thereby detect sources which occupy small regions (Grynszpan and Geselowitz, 1973).

Not until the introduction of a Superconducting Quantum Interference Device (SQUID) could magnetic fields emanating from the brain be detected. The brain's magnetic field has a strength on the order of magnitude of 10^{-8} Gauss (G) (Reite et al., 1976). In comparison, the earth's magnetic field has a strength in the order of 0.5 G (Geselowitz, 1979). Strong fields such as the earth's can be filtered with a gradiometer. For a SQUID to detect magnetic fields in the order of magnitude of 10^{-10} G, the sensors must operate in the superconducting region. Not until recent breakthroughs in superconducting materials operating at near room temperatures (Chen, 1987) was this possible without the use of liquid helium to achieve the superconducting effect. Supercooling demanded cryogenics which made any research with SQUID's costly prohibitive. At the writing of this paper only one commercial company, that the author is aware of, has been able to fabricate metals for commercial use. But, as with other electronic technologies, fabricating superconducting materials for commercial use is expected to soon become commonplace.

The Eye/Brain/Task Testbed

Computer interfaces, as presently known, are all basically passive. That is, a computer does not respond to the user without the user's request. Ideally, an intelligent system would know when intervention is appropriate. The passive role of "ready servant" requires that an operator's needs are anticipated. As an analogy to the ready servant, consider the skilled nurse assisting the surgeon anticipating the needs of the surgeon before the request is made. As the computer begins to take on a more active role, the need for the machine to know the operator's activity and intentions becomes essential. This phenomenon we experience everyday, but, may never fully realized. In carrying conversations with other individuals we unconsciously notice the person's facial expressions, posture, eye contact, and variations in speech to anticipate what the person's

intentions are and what he will be saying. Being able to detect these subtleties using optics and computers would be extremely difficult. But, it is believed that using instruments such as MEG or EEG which can estimate a person's mental state, practical steps are being taken toward knowing a person's intention with a computer. The Eye/Brain/Task (EBT) project will be a testbed to test these theories.

Before an explanation for how the various electro/megneto recording techniques will be used on this project, a description of the Ocular Attention Sensing Interface System (OASIS) should be provided. OASIS is comprised of a testbed and prototype eye/voice control system. Basically, a low intensity infrared light is shined onto a person's eyes. The pupils of the eye reflect the light to an oculometer which determines eye position. This information is then fedback to the monitor in the form of a cursor. Therefore, this prototype allows an operator to move a cursor on the screen by simply moving the eyes. A voice recognizer also allows the operator to give verbal commands. By integrating eye data with brain wave activity, it then becomes possible to determine where a person is looking in relation to the screen and the person's associated level of attention within the region.

To fully complete the required components to determine a person's intention, an indicator of the workload or task is still required. The full integration of the three components, eye, brain, and task can best be explained with an example. Imagine a scene as depicted in figure 1. The object of this task is to direct as many vehicles as possible through the obstacles to reach the other side. "Looking" at a vehicle and giving it the verbal command "GO" propels it forward toward the obstacles. Another vehicle can then be selected and also given the command to proceed. By this time it probably becomes necessary to service the first vehicle and direct it to make either a "LEFT" or "RIGHT" turn. Another vehicle is then selected and also given the command to proceed forward. What has been developed here is a rich environment which allows a researcher to correlate task information with brain activity and visual information. Audio stimuli can also be interjected.

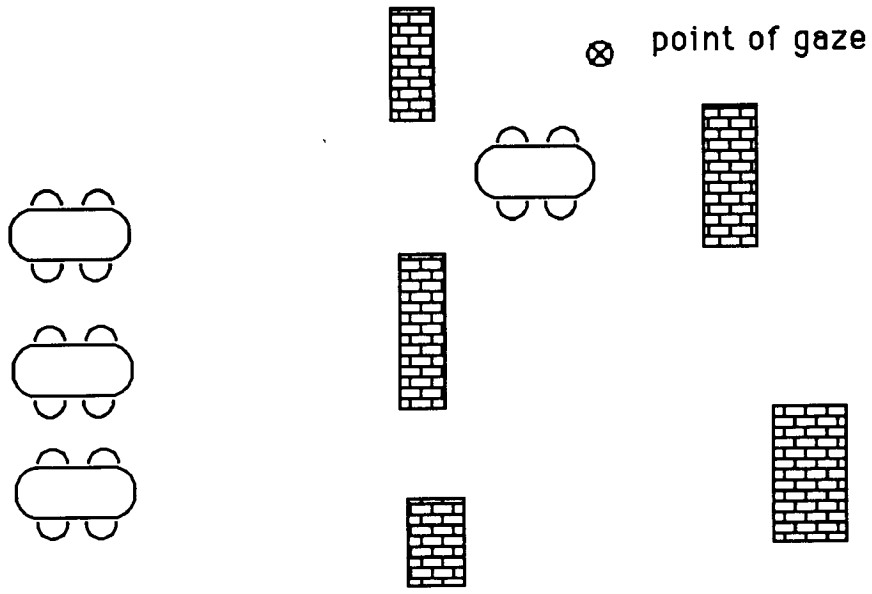


Figure 1. Vehicle manuevering task

Conclusion

The Phase II effort will concern itself with the development of a prototype EBT testbed, and, through applied research and development, the refinement and optimization of the system. The principle 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.

Acronyms

DC	direct current
EBT	Eye/Brain/Task
EEG	electroencephalography
G	Gauss
MEG	magnetoencephalography
OASIS	Ocular Attention Sensing Interface System
SBIR	Small Business Innovative Research
SQUID	Superconducting Quantum Interference Device

N88-16422

Conceptual Information Processing -- A Robust Approach to KBS-DBMS Integration*

Allen V. Lazzara and William Tepfenhart, Ph.D.
Knowledge Systems Concepts, Inc.
262 Liberty Plaza
Rome, NY 13440-4460

Richard C. White
Datalogic Systems, Inc.
2790 Skypark Drive, Suite 110
Torrance, CA 90505

Raymond Liuzzi, Ph.D.
Rome Air Development Center/COTD
Griffiss AFB, NY 13441-5700

ABSTRACT

Integrating the respective functionality and architectural features of knowledge base and database management systems is a topic of considerable interest. This paper addresses several aspects of that topic and the associated issues, beginning with a discussion of the significance of integration and the complexity of the problems associated with accomplishing such integration. Current approaches to integration are described as well as their shortcomings. These shortcomings and the need to fuse the capabilities of both knowledge base and database management systems motivates the investigation of information processing systems based on new information processing paradigms. One such paradigm is concept-based processing, i.e., processing based on concepts and conceptual relations. Reporting on ongoing work, this paper describes our approach to robust knowledge and database system integration by discussing our progress in the development of an experimental model for conceptual information processing.

1 Introduction

Knowledge based systems (KBS) and database management systems (DBMS) technologies have evolved essentially independently of each other. Until very recently, there has been little concern for the mutual integration of these technologies and their respective systems. But with the advent and evolution of numerous experimental knowledge based systems in industry, government, and the military, most of which were developed to establish a proof of concept for automating the performance (or portions thereof) of specific user functions independently of any operational environment, developers have recognized the potential for enhancing current information systems with advancing knowledge based system (KBS) techniques and methods. Since most information systems provide functional support to users through large, shared, database systems, the insertion of knowledge based systems or the integration of knowledge based system techniques into these mainstream environments has resulted in considerable complexities. Despite the current attention to that problem as evidenced by the numerous articles, projects, and discussions devoted to it, there is a conspicuous absence of any definitive successful solutions.

Our examination of recent KBS-DBMS integration efforts suggests that fundamentally different approaches to the fusion of the respective DBMS and KBS capabilities must take place. Although it is clear that the total convergence of KBS and DBMS technologies is many years away, our research has

---

* This research is supported by Rome Air Development Center under contract F30602-85-C-0228.

been directed toward the development of concept-based information processing that results in systems that take on the characteristics of both knowledge and data systems. Our approach to the development of such systems is from an infological, as well as a computational perspective. With regard to the infological, we are attempting to define methods for applying and verifying the concepts and conceptual relations in our system. From a computational perspective, we have been building a library of conceptual and computational primitives and strategies to realize concept based processing. The theoretical basis of which has been formally established in Sowa's conceptual graph formalism.

The paper begins with a synopsis of KBS-DBMS integration R&D trends and an overview of the integration problem. Current integration implementation alternatives are described. Reporting on work in progress, it concludes by discussing our initial steps towards developing a conceptual information processor, and its infological basis.

2 Trends In KBS-DBMS Integration

Researchers in artificial intelligence, database management, software engineering, and machine architecture, as well as numerous other disciplines, have contributed theories, methods, and questions concerning the development of advanced information systems. A principal concern in many of these studies is the linkage of knowledge based and database system technologies. Concentrating on the KBS and DBMS technologies, a diversity of views on KBS-DBMS integration exist. For example, researchers such as Fox and McDermott have addressed the operational issues of inserting knowledge based systems into the heterogeneous and data rich computational environments of manufacturing [Fox 86]. Jarke and Vassilou have reported on a variety of KBS-DBMS interconnection possibilities [Jarke 85]. KBS-DBMS application techniques such as the extraction of knowledge from existing clinical databases have been attempted by Wiederhold and Blum [Wiederhold 86], while Missikoff and Wiederhold have defined criteria and requirements for an expert database system [Missikoff 86]. Bic and Gilbert have taken the view that more expressive data modeling formalisms are necessary to enhance database management systems, see [Bic 86]. Kellogg and many others, including the Japanese, continue their research in logic-based systems, as a vehicle to progress from data management to knowledge management, see [Kellogg 86].

There is some evidence to indicate that DBMS technology is being influenced by KBS technology. Research in DBMS data models is shifting from the development of traditional machine-oriented database modeling formalisms to more expressive models of the user problem domain. Efforts are being made to distinguish data from conceptual models applied to that domain. Thus, within DBMS technology there is the recognition that the work by AI researchers in conceptual modeling, knowledge representation, inference processing, natural language processing, and graphics-oriented user interfaces is relevant for expanding DBMS capabilities. Hence, DBMS researchers are attempting to evolve operational environments which support different user views on the content and organization of their data. In addition, there is considerable interest in producing "intelligent" database systems with capabilities for automatic data acquisition, processing, and dissemination without user intervention.

KBS technology can be effectively applied today if the problem to be solved is narrowly-scoped and its solution by an expert can be captured. Although no common definition or criteria exist for knowledge based systems, [Harmon 86] delineates knowledge based systems that have been developed for business and industrial contexts. Despite the fact that these systems are touted as 'operational', the majority are for very special and limited applications that are not well integrated into the context of larger information systems. It has become apparent that future KBSs cannot exist as such. That is, they must exist within the context of much larger systems with significant interfaces to their subsystems, e.g., database management and communications subsystems. A similar conclusion applies to the KBS development tools. In most instances, these capabilities are extremely "low-level"; that is, skilled programming expertise is necessary to build application solutions from the primitive capabilities provided by the tool set. The languages and tool sets used to develop these systems, e.g., ART, KEE, OPS5, exhibit weaknesses reflecting their state of development and motivation to solve small-scale, narrowly-scoped problems. In addition, they do not support application scaling, meet certain kinds of system performance requirements, and integrate with conventional computational and database processing. Work is ongoing to extend the functionality of these existing tools, e.g., the extension of Intellicorp's KEE to OPUS, and develop next generation tool sets, e.g., Teknowledge's ABE. These extensions are in response to the near-term need to increase the size and performance of knowledge based systems, integrate with conventional database and software systems, modularize and reuse components, and share knowledge bases among several related applications.

Another significant trend overlapping both KBS and DBMS technologies is the movement of KBS development tools, i.e., expert shells, to mainframe environments. The motivation is to place KBS technology in the environment of large scale general purpose computers, thus, increasing the potential for integration with mainstream software and databases. To this end, IBM has introduced the Expert Systems Environment (ESE), an expert systems shell that runs on IBM mainframes under MVS and VM operating systems. Also, Software A&E has ported its Knowledge Engineering System (KES) expert system shell to IBM mainframes, and other vendors are following. While it is too early to assess the repercussions of this movement, it is likely that they could in the near term have a profound influence on the integration of KBS capabilities into those environments.

3 The Complexity of the Integration Problem and Approaches

As discussed in [Lazzara 86], the complexity of the KBS-DBMS integration problem stems from several fundamental differences in the two kinds of systems, including the type of problem each is intended to solve, their period of development, and basic design/implementation features. In addressing some of these differences, the subsequent discussion will focus on the reasons for that problem's complexity, as well as some of the objectives to be achieved in developing a solution to it. Since our experience is in the study and development of prototype Command Control, Communications, and Intelligence (C^3I) knowledge and data systems for the United States Air Force, the remainder of this paper will address this operational domain.

At this point, the discussion will assume a naive interpretation of the concept of integration. General characteristics of recent C^3I prototype KBS developments will be presented, followed by an overview of the C^3I information systems and their constituent DBMSs. Some alternative approaches to KBS-DBMS integration which have thus far been proposed and pursued will also be discussed in terms of their respective merits and viability. The importance of achieving a solution to that problem from a technological as well as an economic viewpoint will also be addressed.

3.1 Current C^3I KBS Prototype Developments

Efforts to apply KBS technology to C^3I functions in recent years have focused on developing prototype KBSs to solve a very specific problem. These prototypes are intended primarily to establish feasibility -- the feasibility of applying KBS technology to perform certain problem solving and decision making aspects of those intelligence functions. Their design and implementation has focused on using AI-based software (e.g., the LISP and PROLOG programming languages) and hardware technology (e.g. LISP machines), rather than on the more conventional hardware and software comprising the C^3I operational computing environments.

For the most part, C^3I KBS prototype development has proceeded in the absence of any considerations for interfacing those systems with or integrating them into the general context of current C^3I-large scale information systems or their constituent DBMSs. Only when a serious effort is made to compare the two system types, does the diversity in their respective problem solving orientations, designs, and implementations become evident. Additional factors such as the respective size of their databases, real-time performance requirements, security, and distributed database and system environments all contribute to that diversity. Thus, efforts to pursue the KBS-DBMS integration problem must necessarily account for this diversity, if they are to be successful.

3.2 C^3I Information Systems and Their Database Management Systems

Today's C^3I computing environments and their DBMSs contrast significantly with the prototype KBSs previously discussed. Many of them consist of conventional large IBM and DEC mainframe systems. Their operating systems and other system software components, e.g., DBMSs, are also rather conventional and may not even reflect the latest state-of-the-art. Application software is typically written in FORTRAN or other common procedural oriented languages. There is little to be found in either hardware or software which has been imported from KBS technology. Consequently, integrating complete KBSs or even portions of their structural and functional elements would be a major design and implementation undertaking in such environments.

The C^3I DBMSs in use today have generally been designed and implemented for large scale system environments. While they may reflect some of the latest DBMS technology, they are not designed to accommodate KBS database elements, e.g., knowledge and rule base elements. Nor are they

endowed with facilities to process such elements beyond the typical data management functions of query processing, updating, and deleting. Thus, DBMSs do not contain a rule processing facility, nor do they support the representation of certain kinds of knowledge and information typical of KBSs. Examples of USAF C^3I systems and their characteristics are presented in [Lazzara 87]. Just a cursory examination of these fundamental system differences is sufficient to indicate the potential magnitude of any integration effort.

An additional significant difference between the two systems which pertains to the very foundations of any integration effort stems from differences in the kinds of problems each was designed to solve. At this point, it will suffice to point out that DBMSs are designed to facilitate very selective acquisition, modification, or deletion of discrete data elements stored on mass memory devices. KBSs, on the other hand, are oriented towards manipulating large quantities of data in main memory and performing operations on it which extend beyond those of a DBMS. This includes applying rules and rule processing functions, search and control, as well as other knowledge-base processing functions, to data structures, many of which are considerably more complex than those maintained in DBMS databases. These characteristics are at least partly due to the fact that KBSs are not just retrieving data in response to a user command, but are also emulating human cognitive functions such as comparing, combining, analyzing, inferring, and decision making.

At one extreme, the KBS-DBMS integration problem might be perceived as a case of trying to mix apples and oranges, and therefore, is a task with little possibility of success. On the other hand, it might be perceived as achievable, at least to some degree, particularly where there is a real need to have KBSs access DBMS databases to support their knowledge processing functions, as well as to infuse some KBS-type functionality into DBMSs. Some approaches to this integration problem are more viable than others, however, as the next topic will indicate.

3.3 Current Integration Approaches: Merits and Deficiencies

Although their are variations in the literature, see [Jarke 85] and [Missikoff 86], four principal approaches are taken to integrate KBS and DBMS technology: (1) tightly integrate KBS and DBMS functionality to develop an entirely new type of system - a knowledge-based management system (KBMS); (2) augment an existing KBS with DBMS structural and functional elements; (3) infuse KBS functionality into DBMSs; and (4) loosely or tightly couple a distinct KBS to a DBMS.

Thus, the C^3I KBS-DBMS integration problem might be approached from any one of these different ways. In the first approach, one could proceed to develop a KBMS. Such a system would constitute an "ultimate" solution to KBS-DBMS integration in that it was developed from the very beginning with KBS-DBMS integration as a goal to be realized in both its design and implementation. But at this point in time, KBMS development is at its most primitive stages. Although a variety of experimental approaches have been tried and others proposed, see [Brodie 86], it is not clear what the architectural features and capabilities of such a system would be. A KBMS has yet to be built, and the likelihood of this kind of system being available for C^3I environments in the near term is remote at best. Nor is it clear how and in what way such a system would either replace or augment DBMSs or other systems in those environments. "Deductive databases" represent a variation of this first approach. Within deductive database systems data manipulation functions and deductive functions are merged into a common system. Examples span from the use of Prolog to unify facts and rules with data manipulation functions to object-oriented approaches to capture the notion of active objects. Both approaches possess benefits, as well as drawbacks, and thus, are actively being researched.

The second major KBS-DBMS integration approach, augmenting an existing KBS with DBMS structural and functional elements, is a less ambitious effort, but none-the-less poses formidable difficulties. For instance, considering C^3I KBS prototype systems implemented in complex and powerful commercially available expert system shells such as ART and KEE, one can easily envision potential complications. In general, these shells are so specialized in their hardware and software implementations, that engaging in a major effort to evolve DBMS capabilities within them could be a significant undertaking. In addition to the inference and data structuring capabilities offered by these shells, the standard functions of a DBMS would have to developed and made available. Most likely, these functions would be implemented in a language suitable to data management functions that may be different than that of the original programming environment. Thus, in preserving the original programming environment and adding the DBMS functionality, these systems would be challenged by large amounts of code and efficiency difficulties.

The more viable approach, i.e., approach three above, to KBS-DBMS integration is towards expanding DBMSs in the direction of more "intelligent" systems. Because of the substantial investment

already made by the USAF in C^3I information system architectures, including recent and ongoing hardware and software updates, and the incorporation of considerable DBMS technology in those systems, the KBS-DBMS integration problem solution seems most likely to be towards of "infusing" KBS functionality into DBMSs. Although conceptually more viable than the other approaches discussed in this section, the infusion of KBS functionality into DBMSs is not without significant technological difficulties. For instance, to provide the functionality of KBSs, e.g., rule processing, and explanation, DBMSs will require the incorporation of conceptual processes, e.g., comparing, abstracting, inferring, and the conceptual definitions and computational primitives to support these processes.

The fourth approach is currently the most common. Two variations exist: loosely coupling a KBS to a DBMS and tightly coupling a KBS to a DBMS. In both arrangements, the KBS and DBMS maintain their identity and their respective functionality. That is, the KBS performs its deductive processes, while the DBMS performs data management functions. A major advantage is that existing databases can be used. In the tightly coupled arrangement, the KBS can request data from the DBMS whenever necessary. When the KBS requires data, it queries the database. In practice, the KBS arrives at a point where external data is necessary to satisfy processing goals. In programming terms, the shell attempts to resolve an external reference, usually by executing a user specified program outside the shell. This program structures the request to a form acceptable by the DBMS's query processor. The query processor is then called with the request. The data manager performs the data retrieval and returns the results to the calling program. The calling program then reformats the results of data retrieved in the form usable by the shell, and the shell continues its deductive processing. As it is evident, many drawbacks exist in a loosely coupled arrangement. The deductive component must state the query in a precise way. This query must be formatted when it is issued and reformatted when data returns. The interface becomes a critical point, often slowing the deductive component. In addition to efficiency and formatting issues, other issues such as dealing with null data exist. In a loosely coupled arrangement, the KBS takes a snapshot of the database prior to performing its deductive processing. That is, selected data is retrieved, structured into the internal format of the KBS, and then processed. Some of the drawbacks that exist in this approach are: the data requirements have to be determined prior to KBS deductive activity; data consistency becomes a problem if the DBMS is updated during the KBS's deductive cycle; and the volume of the data retrieved could be greater than the storage capacity of the deductive component, thus causing severe performance problems.

4 Conceptual Information Processing

It is evident from the preceding section that significant complexities exist in the interconnection or integration of KBS and DBMS technology. Our approach to the integration of KBS and DBMS technology is a variant of case 1 above; that is, the system pursued in our research takes the notion of KBS-DBMS integration as a fundamental goal to be realized in both the design and implementation of the system. Our work has been directed towards defining and implementing a formal systems model for concept-based processing, or in our terminology a **conceptual information processor**. Within this research, we have been proceeding along two dimensions. First, we have been investigating the application of a rigorous semantic theory for describing the conceptual processor system. Second, we have delineated an architecture and conceptual and computational primitives for realizing the architecture. These dimensions will be described after a discussion of concept-based processing.

4.1 An Introduction to Concept-Based Processing

Concept-based processing and derivatives of such processing have been emerging from Sowa's comprehensive and cohesive theory of knowledge representation and computation. Examples of the application of this theory can be found in [Sowa 86], and [Fargues 86]. Sowa has formally described a theory of concept-based processing derived from research in linguistics, psychology, logic, and philosophy. The basis of which is the notion that people understand the world by building mental models. Key assumptions underlying Sowa's theory are: (1) concepts are discrete units; (2) combinations of concepts are not diffuse mixtures, but ordered structures; and (3) only discreet relationships are recorded in concepts; continuous forms must be recorded by patterns of discreet units. Concept types, conceptual relations, and percepts are the building blocks used to represent mental models. These are represented by three types of objects: referents, concept types, and concept relations. A referent denotes individuals, sets, or values explicitly mentioned in the domain being models. Examples of referents are "Bill", [Knowledge Systems Concepts, Datalogics], and 2.17. A concept type asserts the existence of something of a corresponding type, e.g., [PERSON], [BITE], and [APPLE] are concepts. Thus, concept type nodes can represent entities, attributes, states, and events. A conceptual relation denotes how concepts are related or connected, e.g., (AGT) and (OBJ) denote the conceptual relations "agent", "object". Concepts

and their conceptual relations are represented by a structure called a conceptual graph. A simple example in Sowa's linear notation of the statement *"Bill bit an apple"* follows:

[PERSON: Bill] <- (AGNT) <- [BITE] -> (OBJ) -> [APPLE]

In this example, concept types are represented as upper-case type labels surrounded by square brackets, and relations are represented as upper-case type labels surrounded by parentheses. Bill is the name of an individual of concept type PERSON. BITE and APPLE are concept types, while (AGNT) and (OBJ) are relations. (AGNT), denoting agent, links an act, in this example "to bite", to the actor Bill. Likewise, (OBJ), denoting object, links an actor to an entity which is acted upon, i.e, (OBJ) links BITE to the APPLE. Thus, the notation represents the statement *"Bill bit the apple"*.

Conceptual graphs and relations are further organized in schemas, where the terminology 'schema' refers to the organization of these building blocks into larger structures. The conceptual graph notation makes it fairly easy to describe schemas and to formally make useful distinctions. For instance, a schema is composed of a coherent set of statements, instead of a series of isolated statements. Likewise, a schema description for a particular concept contains relations to other concepts rather than to slots and values or attributes and values. The notion of schema and the utility of schema as active objects, instead of inert, static descriptions is fundamental to our goal of building a robust functionally integrated knowledge and data system. The conceptual graph formalism provides the representational expressiveness and the computational model to define active objects and conceptual definitions that allow users and applications to interact at a higher level than that of data stored in the database.

4.2 Our R&D Approach and Status

Our current development of an experimental model of a concept-based information processor, i.e., robust KBS-DBMS, has been driven from three key motivations. First, we are aware of the current limitations of both the knowledge base and database system technologies and intend to develop a robust functionally integrated system. Second, we have defined a partial cognitive model which accounts for the types of cognitive-like processes, e.g., abstracting, inferring, comparing, desired in the system. Third, we have been investigating potential infological approaches for performing a rigorous analysis of the concept types, concept relations, and conceptual processes necessary for a particular application. The first motivation has been previously described, the second and third are described as follows.

The cognitive model driving our implementation is based on Sowa's process model for perception. Basically, four functional categories have been described: perception, learning, reasoning, and memory management. Of these four, our concentration has been placed on the later two. To realize an experimental functional capability in these two areas, we have defined an architecture composed of a user interface, reasoning module, information manager executive, and a global knowledge and data base.

Addressing the reasoning module, we have been developing within a Lisp-based environment the conceptual and computational primitives to realize the definition, representation, and manipulation of computer-based concepts. Applying a machine-independent object-oriented paradigm, we have developed data structures for the primary objects of Sowa's theory, and have extended that declarative framework with the inclusion of a procedural component that can be used to represent dynamic real world processes. With regard to the computational analog for memory management, we have designed and are in the process of implementing the information management executive. An architectural scheme has been developed for uniformly storing and retrieving knowledge and data structures, i.e., objects in both memory and disk have identical representation. A significant contribution of this storage architecture is that knowledge structures, as well as data instances, reside on secondary storage and they are accessed and processed as needed.

The infological techniques being explored are derivative from Sundgren's theory of databases, Newell's knowledge level description, and Barwise and Perry's situation semantics. The goal is to furnish the application developer with the logical analysis techniques to define the concepts, conceptual relations, and referents of the application, as well as provide clearly defined semantics for the application. A larger issue in concept-based processing is the role and representation of contextual information and its influence on 'meaning' and 'interpretation'. Although we are not primarily concerned with the interpretation of natural language, we are confronted with the interpretation of concepts in the larger context of individuals, events, location, and time. The theory of situations developed by Barwise and Perry furnishes a basis for these distinctions and the influences of the context on interpretation. The principal idea of their theory is that the meaning of a sentence is a relation between the sentence and the described situations, while the interpretation of the sentence is the described situation. Conceptual graphs seem to be sufficient representations for semantic information, but in addition to computing over conceptual graphs,

we are striving for a formal basis for capturing the semantics of domain and the respective contextual influences on interpretation.

5 Summary

In this paper we have characterized some things positively and taken a negative stand with regard to others; in both cases goal has been to promote a correct understanding of the KBS-DBMS integration problem which underlies the difficulty in fusing these technologies. Our attempt has been to "cut" through the plethora of information and research studies that seduces us into failing to recognize the real problem or to believe that simple solutions exist. As an approach to this larger problem, we have described a concept-processing paradigm which we believe will resolve many of the difficulties associated with KBS-DBMS integration. Likewise, we have overviewed the status of our ongoing work in the development of such a system, and also, have delineated some initial goals and approaches for infological analysis.

BIBLIOGRAPHY

[Barwise 83] Barwise, J. and Perry, J., Situations and Attitudes, MIT Press, 1983.

[Bic 86] Bic, L., and Gibert, J., "Learning From AI: New Trends in Database Technology", Computer, IEEE, March 1986.

[Brodie 86] Brodie, M.L. and Mylopoulos, J., On Knowledge Base Management Systems - Integrating Artificial Intelligence and Database Technologies, Springer-Verlag, Inc., New York, 1986.

[Fargues 86] Fargues, J., et al.,"Conceptual Graphs for Semantics and Knowledge Processing", IBM Journal of Research and Development, Vol. 30, No. 1, January 1986.

[Fox 86] Fox, M. and McDermott, J., "The Role of Databases in Knowledge-Based Systems", On Knowledge Base Management Systems, Brodie, M. and Mylopoulos, J. (Eds.), Springer-Verlag, 1986.

[Harmon 86] Harmon, P. (ed.), Expert Systems Strategies, Monthly Newsletter, Cutter Information Corp, August 1986.

[Jarke 85] Jarke, M., et.al., Coupling Expert Systems with Database Management Systems, Artificial Intelligence Applications for Business, Proceedings of the NYU Symposium, Ablex Publishing Corporation, Norwood N.J., 1985, p. 65.

[Kellog 86] Kellog, C., "From Data Management to Knowledge Management", Computer, IEEE, January 1986, p. 75.

[Kerschberg 86] Kerschberg,L. (ed.), Expert Database Systems, Proceedings From the First International Workshop, The Benjamin/Cummings Publishing Company, 1986.

[Lazzara 86] Lazzara, A., et al.,Knowledge Based and Database System Integration: A Baseline for a Design Methodology, Interim Report, RADC-TR-86-132, September 1986.

[Lazzara 87] Lazzara, A., et al.,KBS-DBMS Integration Analysis and Design Alternatives, Interim Report, RADC-TR *in publication*, February 1987.

[Missikoff 86] Missikoff, M. and Wiederhold, G., "Towards A Unified Approach for Expert And Database Systems", in: Kerschberg,L. (ed.), Expert Database Systems, Proceedings From the First International Workshop, The Benjamin/Cummings Publishing Company, 1986.

[Sowa 84] Sowa, J.F., Conceptual Structures: Information Processing in Mind and Machine, Addison-Wesley, 1984.

[Sowa 86] Sowa, J.F. and Way, E.C.,"Implementing a Semantic Interpreter Using Conceptual Graphs", IBM Journal of Research and Development, Vol. 30, No. 1, January 1986.

[Weiderhold 86] Weiderhold, G., et al., "An Integration of Knowledge and Data Representation", On Knowledge Base Management Systems - Integrating Artificial Intelligence and Database Technologies, Springer-Verlag, Inc., New York, 1986.

N88-16423

FOUNDATION: TRANSFORMING DATA BASES INTO KNOWLEDGE BASES

R. B. Purves

Boeing Aerospace Company
PO Box 1470, MS JA-68
Huntsville, AL 35807

James R. Carnes
Dannie E. Cutts

Boeing Huntsville AI Center
PO Box 1470, MS JA-65
Huntsville, AL 35807

ABSTRACT

One of the recent problems faced by developers of knowledge based systems is how to best use data already stored in traditional data bases. This paper describes one approach to transforming information stored in relational data bases into knowledge based representations and back again. This system, called Foundation, allows knowledge bases to take advantage of vast amounts of pre-existing data. A benefit of this approach is inspection, and even population, of data bases through an intelligent knowledge-based front-end.

1. INTRODUCTION

A workstation tool (Foundation) has been developed to assist in the complex task of analyzing information contained in engineering data bases. Foundation users are able to graphically display networks of components, query the network for information about the components, and edit data on components in the network. Analysis of the system from various perspectives is supported.

Foundation synthesizes initial structural relationships from the data base data dictionary and is then able to read records from data base tables. Some tables contain structural information, others contain functional information, while still others contain attributive information for particular applications. Structural information is used to construct the initial hierarchical model, then functional and attributive information is used to construct a set of network links between the former data base key values, which are now objects in the knowledge base.

The initial application for Foundation is a knowledge-based interface to Boeing's Space Station Engineering Master Data Base, a system designed for information management in design and development of the NASA Space Station. There are two major key-object mappings: one, part-classes from diverse discipline models describing the many individual components used in the design, and two, part-instances used in describing how all the different parts fit together into an engineering working model. This paper

presents current status and goals of the project, focusing on lessons learned and further application potential.

2. THE WORKSTATION

This development joins two distinct environments, relational data base and knowledge base. The initial work has been in data base environments with well-defined structure and behavioral elements such as those found in engineering design. These elements encompass both static engineering discipline models, (i.e., invariant component structure), and dynamic system models, (i.e., component instance behavior). Translation from the data base to knowledge base system is accomplished through conventional data base access methods driven by the complex qualitative model that is the "foundation" of the knowledge base[3]. The qualitative model used within the knowledge base is the major topic of this section.

2.1. The Data Base

To fully understand the knowledge base model it is necessary to inspect the structure of the typical engineering data base. At high levels the data base is partitioned into tables representing structural or descriptive information, and functional or behavior information. Structural information represents the generic structures of the data base objects, i.e. parts in the data base, as well as the connections between generic parts, such as subcomponent and interface relations. Descriptive information includes mass, thermal, and electrical properties of data base objects. Functional information about data base objects describes the behavior of system components, such as how thermal properties are used to represent conditional behavior or how separate instances of a component function differ within a system.

2.2. The Knowledge Base

In order for Foundation to be used with various data base systems, all access to the data base information is through a query language (SQL) rather than by accessing tables images directly. This approach also helps to ensure data base integrity through the use of the native data base system access and lock protocols.

Structural information is extracted from the data base to define templates of attributes common to all components in a system, as well as component types from which specific component instances are created. Other tables in the data base contain behavioral data about component types. These data are read from the relational data base into the knowledge base in rule form.

The process of translating from the data base to the knowledge base consists of several steps:

1. Read data dictionaries for various tables describing the data base architecture as well as the format and type of data (structural, descriptive, or behavioral) contained within.

2. Establish bounds, (i.e., levels of detail, proper views or perspectives, etc.) on information to ensure that sufficient data are retrieved from the data base. Performance and storage issues drive the need for bounding the data.

3. Read static structures from appropriate data base tables. These structures are used as keys to read static attributive data required for current view or perspective.

4. Read static behavioral data. These data are used to describe the generic behavior of components, but do not vary as the component is used in specific system settings (such as the operating temperature range for a component).

5. Read attributes to describe the instance behavior of a component. The working behavior of a component includes functional descriptions specific to the component and its particular application in the system. Some of these attributes within the description serve only to contain values related to some "working" status, such as current temperature of a fluid in a pump. Some attributes have default initial values set upon creation while others have values assigned during various behavioral inference schemes[1,4].

The architecture of the knowledge model is similar to the data base, with a strict division of generic component and instance information. Structural and descriptive data about components are combined into frame-like structures. However, real differences exist in the complex connections drawn between objects in the instance environment, and in the ability to reason about instances using generic and specific component rules. The component system may be viewed as a hierarchical tree of objects, where each object can be decomposed into its component parts, in a well-connected network of component instances. Figure 1 gives examples of description data contained in the knowledge base.

System functionality is implemented in rule bases, figure 2, where inferences within the knowledge base can be made through forward and backward chaining of component properties or constraints[4]. The resulting inference mechanism, based on propagation of constraints, is applicable to a wide class of physical systems exhibiting discrete or continuous behavior, and can be used with a variety of representations (e.g., quantitative or qualitative)[5].

```
GENERIC DEFINITIONS:
  (deftype PUMP-A (generic-type 'PUMP))
  (defproperty PUMP-A (pressure (LOW (less-than 0))
                                (NOMINAL (range 0 50))
                                (HIGH (greater-than 50)))
                      (temperature (LOW (less-than 10))
                                   (NOMINAL (range 10 80))
                                   (HIGH (greater-than 80))))
INSTANCE DEFINITIONS:
  (defproperty PUMP-A ((PUMP-SWITCH-STATUS 'ON)
                       (PUMP-STATUS 'WORKING)
                       (PUMP-PRESSURE-STATUS 'NOMINAL)
                       (PUMP-TEMPERATURE-STATUS 'NOMINAL)))
```

FIGURE 1: Examples of attribute definitions.

```
GENERIC RULES:
  (defrule PUMP-WORKING
    IF [AND [IS-A-PUMP =some-pump]
            [INPUT-TO-PUMP =something]
    THEN (tell [OUTPUT-FROM-PUMP =something])

INSTANCE RULES:
  (defcomponent PUMP 362 'PUMP-A)
  (defrule PUMP-SHUTDOWN
    (IF [AND [COMPONENT '<PUMP-362>]
             [PUMP-SWITCH-STATUS '<PUMP-362> 'ON]
             [PUMP-PRESSURE-STATUS '<PUMP-362> 'LOW]
             [PUMP-TEMPERATURE-STATUS '<PUMP-362> 'HIGH]]
     THEN (tell [PUMP-STATUS '<PUMP-362> 'ABNORMAL])))
```

FIGURE 2: Examples of rule definitions.

3. CONCLUSIONS AND FUTURE DIRECTIONS

Early work on Foundation shows the feasibility of transforming data from a relational data base into a knowledge based format. Work of this nature is instrumental in bridging the gap between traditional data base systems and knowledge based systems, allowing knowledge based systems to take advantage of existing information stored in data bases. It also paves the way for storage of the large amounts of knowledge structures necessary for more advanced reasoning systems.

Although this project focuses on the "well-defined" environment found in engineering disciplines, future work will represent engineering information at greater and deeper levels of detail. Finer grained properties will be attributed to smaller entities, while larger collections or systems of these entities will be attempted. Reasoning over time will be accomplished with the coupling of time intervals and properties[5].

A new problem challenging the artificial intelligence community is "design knowledge capture," that is, the construction of systems with inherent built-in evolvability toward more advanced technologies and machine intelligence[2]. Knowledge within these detailed designs can be physical, conceptual, functional, or structural. Design knowledge includes, for example, traceability to requirements, standards, and specifications. Attributes of a part are described, as well as analyses, simulations, and trade studies. Other uses include validation/verification and operations/maintenance activities. Success in this evolution process depends on being able to capture "as-built" design knowledge from the outset. In a design knowledge environment "knowing why a quantity has a specific value is at least as important as knowing what the value is.[5]"

REFERENCES

1. Brachman, R.J., R.E. Fikes, and H.J. Levesque, KRYPTON: A Functional Approach to Knowledge Representation, IEEE Computer, Vol. 16(10), 1983, pp. 67-73.

2. NASA TM 89190, Advancing Automation and Robotics Technology for the Space Station, Advanced Technology Advisory Committee (ATAC), Progress Report Three, October 1, 1986, pp. 27-28.

3. Reiter, R., A Theory of Diagnosis from First Principles, Artificial Intelligence, Vol. 32, 1987, pp. 57-95.

4. Rowley, S., H. Shrobe, R. Cassels, and W. Hamscher, Joshua(TM): Uniform Access to Heterogeneous Knowledge Structures, AAAI-87, Proceedings of the Sixth National Conference on Artificial Intelligence, July 13-17, 1987, pp. 48-52.

5. Williams, B.C., Doing Time: Putting Qualitative Reasoning on Firmer Ground, AAAI-86, Proceedings of the Fifth National Conference on Artificial Intelligence, August 11-15, 1986, pp. 105-112.

N88-16424

THE INTELLIGENT USER INTERFACE FOR NASA'S ADVANCED INFORMATION MANAGEMENT SYSTEMS

William J. Campbell
Nicholas Short, Jr.
NASA/Goddard Space Flight Center
National Space Science Data Center
Code 634
Greenbelt, MD 20771

Larry H. Rolofs
Computer Technology Associates Inc.
McLean, VA 22102

Scott L. Wattawa
Science Application Research Inc.
Landover, MD 20706

ABSTRACT

In the past decade, operations and research projects that support a major portion of NASA's overall mission have experienced a dramatic increase in the volume of generated data and resultant information that is unparalleled in the agency's history. The effect of such an increase is that most of the science and engineering discipliens are undergoing an information glut, which has occurred, not only because of the amount, but also because of the type of data being collected (i.e., spatial data).

This information glut is growing nonlinearly, and is expected to continue to grow in this fashion for the foreseeable future. Consequently, it is becoming physically and intellectually impossible to identify, select, modify , and access the most suitable information specifically applicalbe to the various engineering and research projects of interest. Thus, the dilemma arises that the amount and complexity of information has exceeded and will continue to exceed using present information systems, the ability of all the scientists and engineers to understand and take advantage of this information.

Based on the scope, expected growth and dominance of this problem, it is anticipated that the future ability of NASA to perform meaningful space and earth related research will be significantly affected by its inability to manage and use its collected information to derive useful knowledge. Consequently, it is envisioned that dramatically different approaches to data management will need to be taken if earth and space related operations and scientific investigations

PRECEDING PAGE BLANK NOT FILMED

are ever to take full advantage of the information and data being collected and stored.

As a result of this situation, NASA has initiated the Intelligent Data Management Project to design and develop Advanced Information Management Systems (AIMS). The project's primary goal is, using advanced technologies, to formulate, design, and develop advanced information systems that are capable of supporting the agency's future space research and operational information management needs. The first effort of the Project was the development of a prototype Intelligent User Interface (IUI) to an operational scientific database, using expert systems and natural language processing technologies. This paper presents an overview of the IUI formulation and development effort.

N88-16425

An AI Approach for Scheduling Space-Station Payloads at Kennedy Space Center

by

D. Castillo, D. Ihrie
HARRIS Corporation
Melbourne, FL 32935

M. McDaniel
MDAC-KSC
KSC, FL 32899

R. Tilley
NASA/SS-OCO
KSC, FL 32899

ABSTRACT

Payload Processing for Space-Station Operations, including mission manifesting and its effect on KSC Ground Resource Allocation, represents a class of ill-structured, complex scheduling problems which are often unsuitable for applying optimization algorithms. The situation has inspired the development of AI-based planning and scheduling systems specifically designed for Payload Processing activities. This paper examines the application of an AI-based system, called PHITS, to integrated payload scheduling and its effect on Ground Resource Allocation at KSC.

Unique to the PHITS approach is the process by which schedule generation occurs. Experiments are represented in terms of objects which are semantically related based on mission goals. Unlike conventional scheduling systems, task flows are only defined for individual objects. Integrated schedules are generated by evaluating object, attribute, value (OAV) triplets for experiments considered candidates for flight. OAV triplets contain user-defined constraints on object interaction. A goal directed simulation subsystem examines the schedule and performs conflict resolution as needed to achieve the on-orbit requirement goal.

INTRODUCTION

Part of Kennedy Space Center's (KSC) responsibility is the prelaunch preparation and integration of experiments for transport to Space Station and the deintegration and processing of returned hardware. As an example, consider an experiment which operates in a pressurized laboratory module. The experiment is first received at the launch site where it is inspected and integrated into a laboratory equipment rack. A high fidelity simulator then verifies the module-to-rack interfaces prior to installing the rack in a logistics module. Finally, the logistics module is installed in the orbiter for transport to the Space Station. Each payload generates unique and complex demands which require a large array of resources including facilities, equipment, materials, personnel skills and training. Furthermore, the process becomes even more complex when processing multiple shuttle flights in parallel. As a result of the relatively fixed level of available resources and

highly structured set of schedule constraints imposed by the shuttle launch and landing, planning and scheduling of processing activities associated with new flights represents a nontrivial task. Efficient planning and management of this process is a key element in maximizing the effective use of the Space Station while minimizing the cost.

The above situation has directed researchers toward AI-based scheduling systems designed to operate in the domain of resource-constrained scheduling. This effort has produced systems such as MAESTRO [2], PLANNET [8], MARS [9], PEGASUS [4] and others. A commonality among many of these systems is the methodology used in generating conflict-free schedules. In many cases, a schedule is generated using traditional CPM routines, followed by heuristic methods that attempt to produce a conflict free schedule. More recently, researchers have investigated the effects of temporal reasoning applied to resource-constrained scheduling [1] [11] in hopes of automating deductions about time.

The Payload Handling Inventory Tracking System (PHITS), developed by Harris Corporation [5], deviates from the above philosophical methodologies in that it provides a modeling environment that couples scheduling, simulation and AI technologies in one unique modeling environment. Bruno et al. [3] share this philosophy and have applied it to the domain of Flexible Manufacturing Systems (FMS). However, the distinction between PHITS and Bruno's FMS system is the way schedule generation and simulation are performed. This paper examines the application of PHITS to integrated payload scheduling and the effects this has on Ground Resource Management giving particular attention to the usability, scheduling and simulation aspects of PHITS. The reader is referred to Ihrie et al. [6] for an overview of the technologies used in PHITS.

STORAGE STUDY OBJECTIVES

MDAC-KSC was tasked to identify Space Station payload storage policy alternatives at KSC. This required a forecast of storage requirements in relationship to experiment types, sizes and numbers that flowed through the processing facilities at KSC. It was determined that a software tool capable of tracking resources and forecasting their requirements in a very dynamic environment would greatly assist MDAC-KSC in accomplishing the objectives of the study. In addition, a "what if" feature for performing sensitivity analysis on the primary variables defined in the study would provide the flexibility for examining competing scenarios. MDAC in cooperation with NASA-KSC agreed to use the PHITS system for supporting their study efforts.

Having defined the manifest as the primary variable for determining storage study policies, it was recognized that the ability to generate manifests was necessary since manifests were generally unavailable for most flights considered in the study. PHITS possessed the capability to generate manifests based on

experiments from the Civil Needs Database and their associated constraints. Flights from 1994 to 2002 including 127 experiments were considered. Based on the results of the manifest, PHITS produced a payload schedule defining the timeline for all tasks and resources. The schedule was then simulated to determine storage requirements and resolve any resource conflicts that occurred. The following sections describe this process.

BUILDING THE STORAGE STUDY MODEL

Developing the Storage Study Model required the user to define the objects relevant to the study, such as experiments, log-modules etc. PHITS provides a powerful user interface for simplifying the process of identifying and defining objects. The Genealogy Editor was utilized to identify each object as a class or instance and graphically portray all parent-child relationships. Figure 1 illustrates the Genealogy Editor.

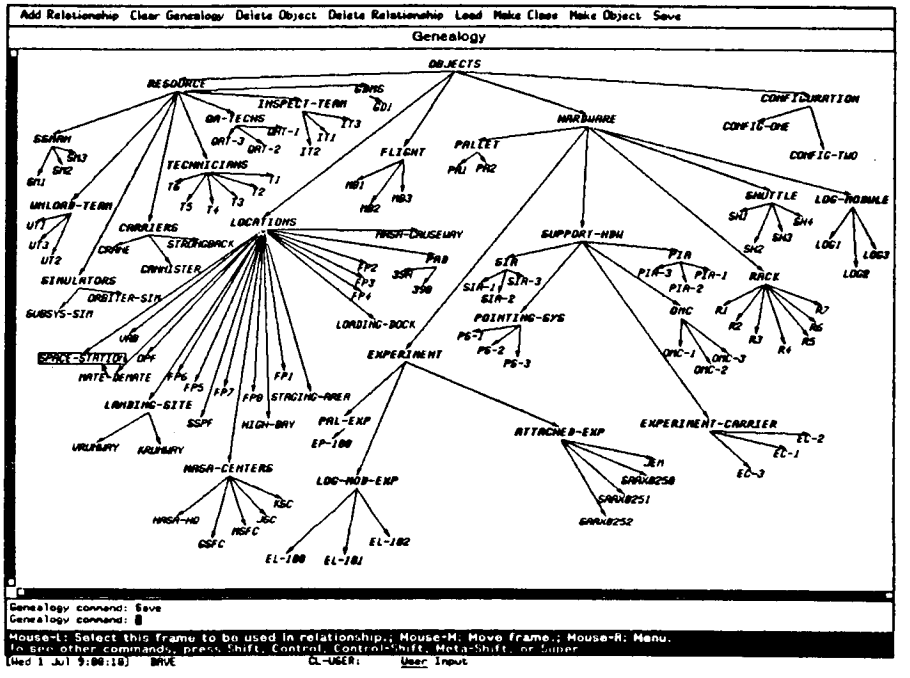


Figure-1: PHITS Genealogy Editor

Object task flow scripts were then defined for each object. Task flows were either inherited from a class of Experiment, or uniquely created for the specific experiment being defined. Task-based resources and storage types were identified during this process. OAV requirements were also instantiated at this time. OAV triplets represent constraints one object imposes on other objects. In the Storage Study, for example, an experiment object that affects available shuttle mass is represented by the following OAV triplet:

```
((shuttle mass 2345))
```

ORIGINAL PAGE IS
OF POOR QUALITY

PHITS uses this property list to determine if the experiment is capable of being attached to the flight. The scheduling component utilizes OAV triplets for developing the entire task network for a given mission.

Once all objects were identified and defined in terms of their task network flows, the Attribute Editor was accessed for defining object attributes. For example, object MB-3 contains an attribute called "launch-date" with a facet of "is" and a value of "July 15, 1994." For the Storage Study, each experiment contained an attribute called "sq-ft" which represented the square-foot dimensions of the experiment. Furthermore, each experiment class contained an algebraic expression attribute utilized to compute storage requirements based on the sq-ft attribute of each experiment.

PHITS provides a Structure Editor for attaching experiments to a given flight. Attaching experiments in this manner guarantees the experiment will be manifested during payload scheduling. This is a useful feature when a manifest has been previously set by NASA or when analyzing different manifests.

PAYLOAD SCHEDULING

Payload scheduling in PHITS consists of connecting individual object definitions into a single integrated payload flow that satisfies all connector constraints. Figure 2 illustrates the task flow script for individual objects. From Figure 2 the object shuttle is defined by more than one set of task flows. Each flow may contain one or more start and end nodes. End nodes contain information corresponding to a connecting set of tasks on another object.

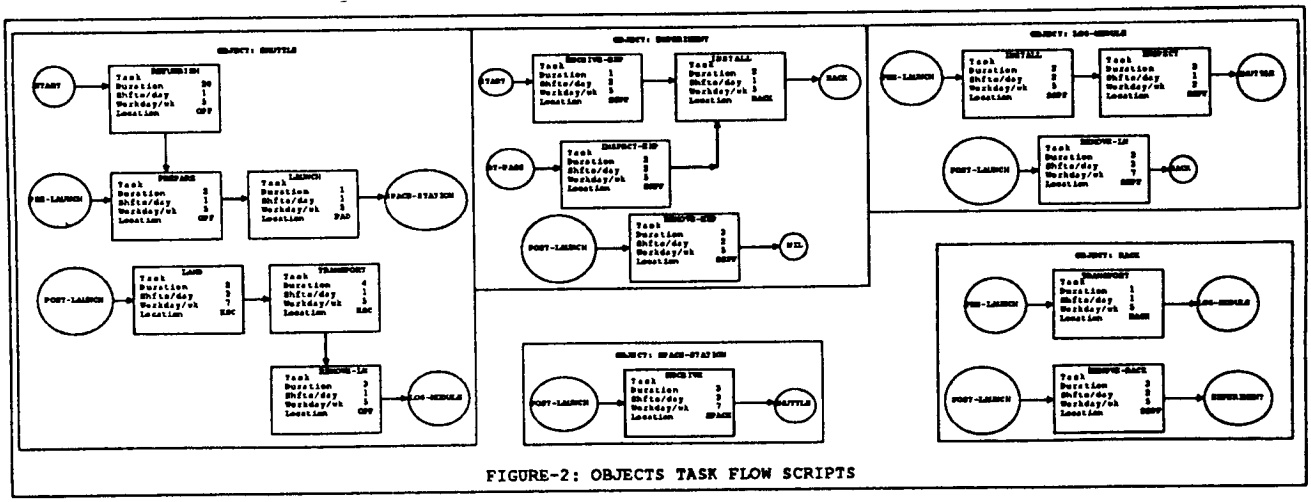


FIGURE-2: OBJECTS TASK FLOW SCRIPTS

The scheduler operates on a user-selected set of missions and experiments. A mission is considered an object with a launch date and possibly containing a set of experiments. Additional experiments are left unattached and may be considered candidates for flight. For each manifested and unmanifested experiment, the

ORIGINAL PAGE IS
OF POOR QUALITY

ORIGINAL PAGE IS
OF POOR QUALITY

scheduler first constructs a complete stand-alone task flow by following connections between object task flows. OAV triplets are not considered during this forward pass. Figure 3 represents this process:

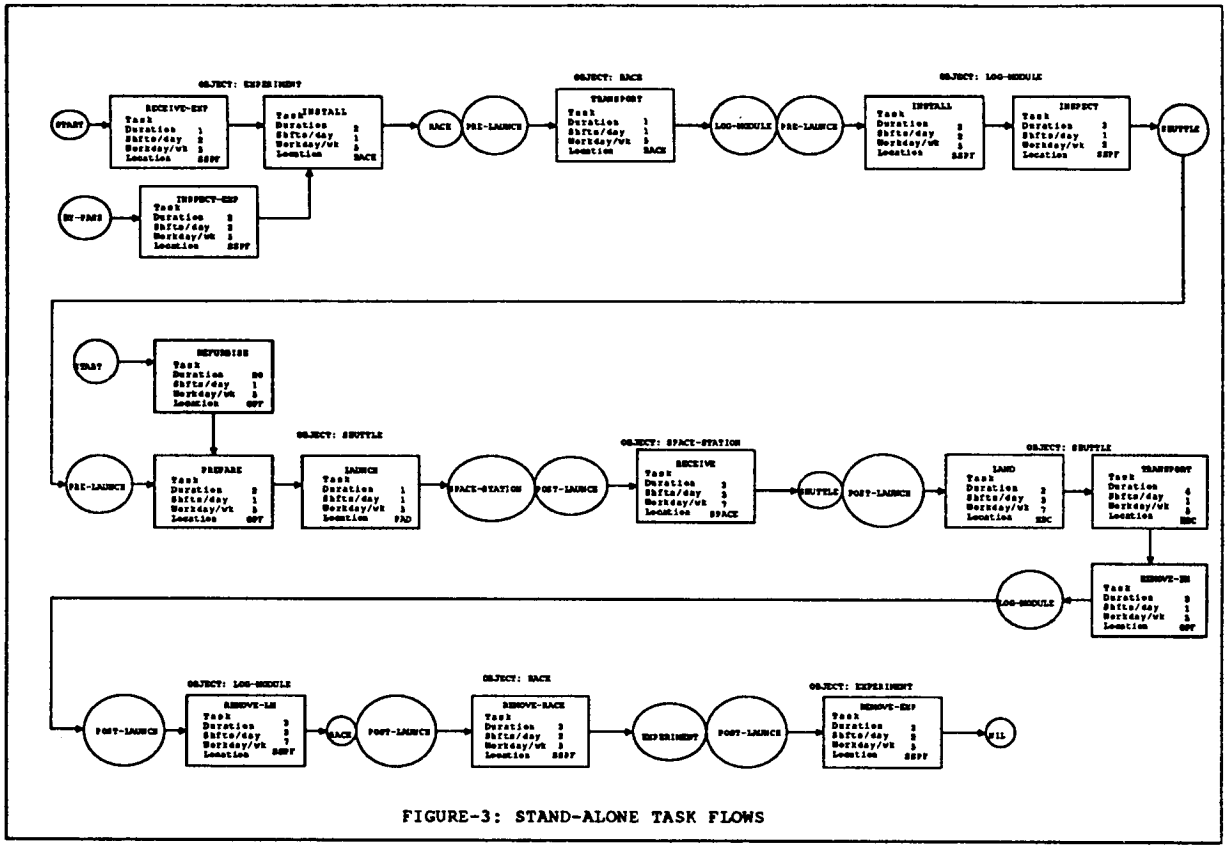


FIGURE-3: STAND-ALONE TASK FLOWS

At this point, the scheduler must eliminate duplicate tasks from experiment flows attached to a single mission, manifest the unmanifested experiments, replace object classes with specific objects and assign task dates to meet launch dates. The key ingredient here is ensuring that all mission constraints are met. Constraints are represented by the OAV triplets mentioned previously. At each point in the flow where multiple objects are integrated, the scheduler attaches as many objects as possible without violating the OAV constraints. Manifested experiments are attached to a mission first, followed by unmanifested experiments wherever possible, based on a user-selected priority. The final output is a fully integrated task flow for each mission, such as that shown in Figure 4. Using this approach, the scheduler guarantees each mission is internally conflict free and satisfies all object constraints.

A CPM routine is then applied to the overall network to instantiate the Early-Start, Late-Start, Early-Finish and Late-Finish dates on the network. The critical path is defined as the task path where Early-Start is equal to Late-Start and Early-Finish is equal to Late-Finish.

ORIGINAL PAGE IS
OF POOR QUALITY

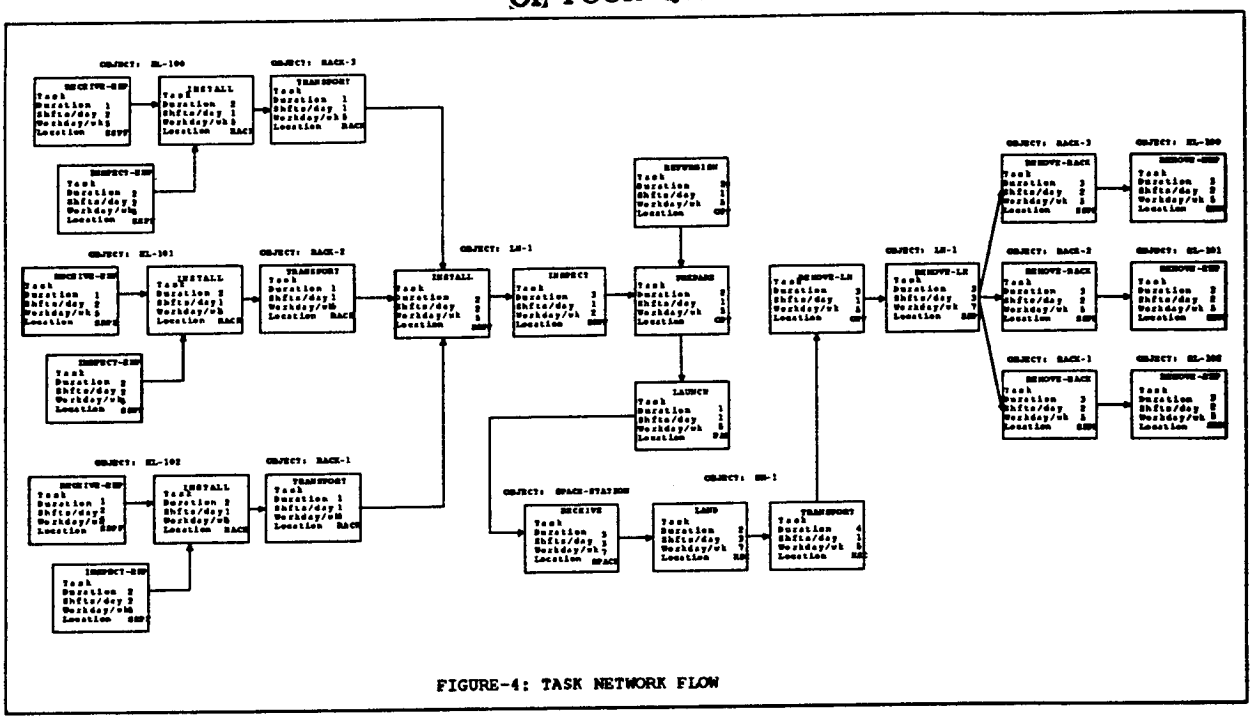


FIGURE-4: TASK NETWORK FLOW

As mentioned above, the scheduler instantiates the early-start late-finish range for each task. This range dictates the permissible time interval of each task. If the range is violated, then the on-orbit requirement date is not guaranteed, resulting in the possibility of a slipped schedule. One of the designated tasks of the simulation subsystem is to ensure schedules are maintained. Figure 5 illustrates the early-start late-finish range for a task.

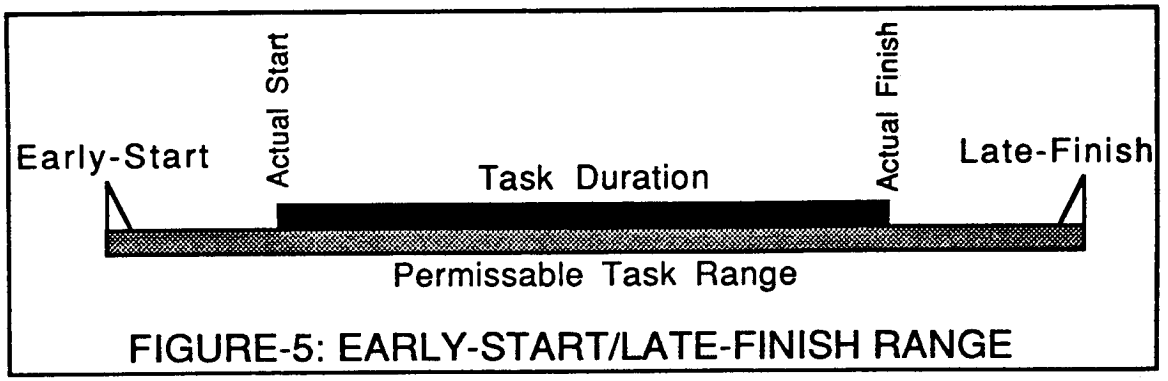


FIGURE-5: EARLY-START/LATE-FINISH RANGE

SIMULATION

Simulation in PHITS represents a complex process of tracking objects, resolving resource conflicts and dynamically re-adjusting in order to achieve the on-orbit requirement goal. An event calendar is utilized as the central control structure that communicates with other objects by message passing in much

the same way as KBS [10] and ROSS [7]. Tasks are examined by the event calendar to determine if a violation in the early-start late-finish range is imminent. If a violation is detected, the task object is given to the Meta-level Resource Manager to determine if a local resolution is feasible. Local resolution implies conflict resolution is applied to the task in question without disrupting the processing of other tasks.

Local resolution is an attempt to quickly resolve resource conflicts by applying a small set of heuristic information on a local scale. PHITS utilizes a Pre-processing facility as a vehicle for performing local resolution. If a task range violation is detected by the event-calendar or Resource Manager, then Pre-processing is initiated. Figure 6 represents a conceptual illustration of a task range violation. Notice the Late-Start milestone has been exceeded, therefore, local resolution will attempt to shorten the task duration so that the task does not violate its Late-Finish range. Pre-processing applies very simple heuristics such as overtime, increased resources where applicable, etc. to satisfy the task's range constraints. If this effort is unsuccessful, then global resolution is initiated. Global resolution is defined as the process where resolving a conflict for one task affects another task's processing. The current version of PHITS contains the architecture for supporting global resolution, however, a significant amount of knowledge engineering is needed before full scale implementation can occur. Although lack of global resolution did not adversely effect the outcome of the Storage Study, it was recognized that other candidate studies would probably require this capability.

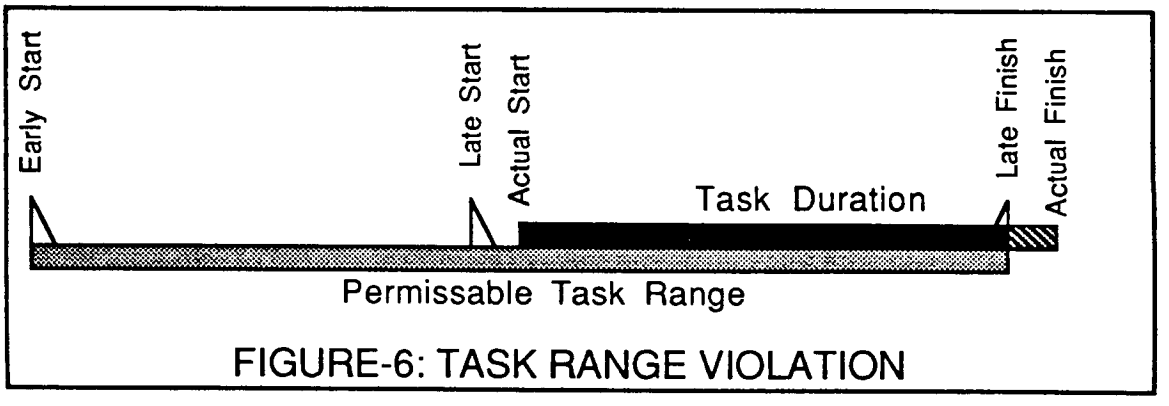


FIGURE-6: TASK RANGE VIOLATION

Due to the object-oriented nature of PHITS, storage requirements were easily generated. Attribute expressions containing algebraic equations and sq-ft values were evaluated at simulation time. Storage requirements based on experiment and storage types were collected. Cumulative storage requirements were reflected graphically as temporal representations which proved useful for comparing multiple manifest scenarios. Gantt charts were utilized for displaying the overall processing schedule. Figures 7 and 8 illustrate these features.

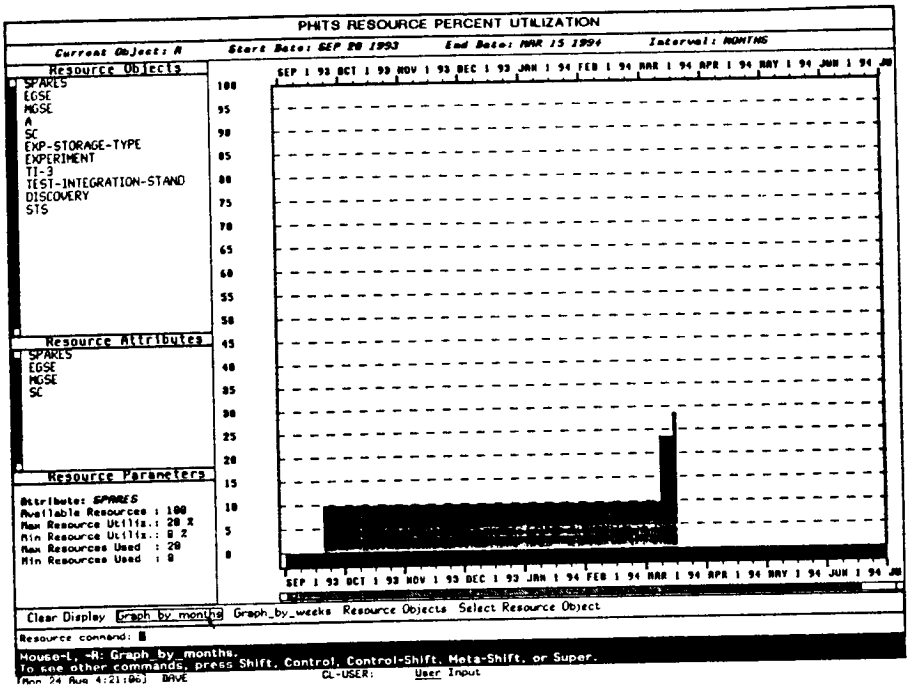


FIGURE-7: RESOURCE UTILIZATION GRAPH

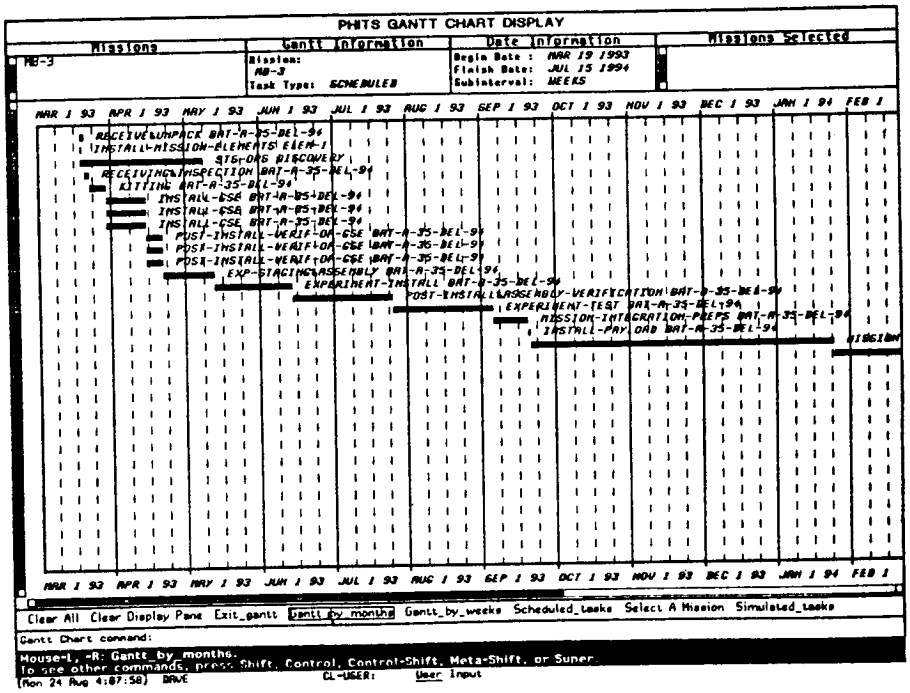


FIGURE-8: GANTT CHART OF SCHEDULE

ORIGINAL PAGE IS
OF POOR QUALITY

Although not utilized during the Storage Study analysis, PHITS features an animation component which demonstrates a pictorial view of the simulation process. Object icon attributes are updated dynamically to represent the discrete changes in the simulation status. This proves valuable when an analyst is concerned with tracking specific objects throughout the simulation, or discovering potential bottlenecks of a process.

FUTURE RESEARCH

Future research efforts for PHITS will focus on the Resource Manager and global resolution. Temporal relations will be examined in an effort to better understand and manipulate conflicts on the global scale. Other directions include porting the technology from a Symbolics environment to an 80386 environment. Incorporating intelligence features into the manifesting capability of PHITS also has potential for future research. Finally, the technology contained in PHITS will be investigated for modeling other problem domains within the Space Station Program.

CONCLUSION

PHITS is a prototype modeling tool capable of addressing many Space Station related concerns. The system's object-oriented design approach coupled with a powerful user interface provide the user with capabilities to easily define and model many applications. PHITS differs from many AI-based systems in that it couples scheduling and goal-directed simulation to ensure on-orbit requirement dates are satisfied.

ACKNOWLEDGMENT

This work would have not been possible without the patience and support of the people at MDAC-KSC, particularly John Weber and David Dixon. The authors express their appreciation to these individuals.

REFERENCES

1. Allen, J.F., Koomen J.A., "Planning using a Temporal World Model", Proceedings of the Eighth International Joint Conference on Artificial Intelligence, Karlsruhe, West Germany, 1983, pp. 741-747.

2. Britt, D., Geoffroy, A., Gohring, J., "A Scheduling And Resource Management System For Space Applications", Proceedings on Artificial Intelligence for Space Applications, Huntsville, AL, (Nov.) 1986, pp. 303-310.

3. Bruno, G., Antonio, E, Laface, P., "A Rule-Based System to Schedule Production", COMPUTER, (Jul.) 1986, pp. 32-39.

4. Harris Corporation, "PEGASUS Software User's Manual", 1986.

5. Harris Corporation, "PHITS Phase 3 Quarterly Report", 1987.

6. Ihrie, D., Castillo, D., Kovarik, V., Tilley, R., "PHITS: An Intelligent Modeling Environment for Space Station Operations at Kennedy Space Center", Proceedings Southwest Conference on Simulation and M Huntsville, AL, (Oct.) 1987.

7. McArthur, D., Klahr, P., The ROSS Language Manual, N-1854-AF, The Rand Corporation, 1982.

8. McDonnell Douglas Astronautics Company, The Plannet Expert System: Version 1.1 User's Manual, Project Report to NASA, 1985.

9. McDonnell Douglas Astronautics Company, Mixed Architecture For Reactive Scheduling, contract NAS8-32350, KSC-1, 1986.

10. Reddy, Y.V.Ramana, Fox, M., Hussain, N., McRoberts, M., "The Knowledge-Based Simulation System", IEEE Software, (Mar.) 1986, pp. 26-37.

11. Vere, S., "Planning in Time: Windows and Durations for Activities and Goals", Technical Report, Jet Propulsion Laboratory 1981.

N88-16426

SCHEDULING SPACECRAFT OPERATIONS

Daniel L. Britt Amy L. Geoffroy Philip R. Schaefer
Martin Marietta Space Systems
P.O. Box 179, MS 0427
Denver, Colorado 80201

John R. Gohring
Martin Marietta Data Systems
116 Inverness Drive East
Englewood, Colorado 80112

ABSTRACT

A prototype scheduling system named MAESTRO currently under development is being used to explore possible approaches to the spacecraft operations scheduling problem. Results so far indicate that the appropriate combination of heuristic and other techniques can provide an acceptable solution to the scheduling problem over a wide range of operational scenarios and management approaches. These can include centralized or distributed instrument or systems control, batch or incremental scheduling, scheduling loose resource envelopes or exact profiles, and scheduling with varying degrees of user intervention. Techniques used within MAESTRO to provide this flexibility and power include constraint propagation mechanisms, multiple asynchronous processes, prioritized transaction-based command management, resource opportunity calculation, user-alterable selection and placement mechanisms, and maintenance of multiple schedules and resource profiles. These techniques and the scheduling complexities requiring them will be discussed in this paper.

INTRODUCTION

As the complexity and sophistication of spacecraft and the experiments they carry increase, the cost of operating them increases as well. It is imperative that these spacecraft be operated as efficiently as possible. This will require significant changes in the way spacecraft are managed, including more sophisticated scheduling techniques. Assumptions made in the past to simplify the scheduling problem will no longer be supportable. A system which controls spacecraft must be capable of evolving to meet demands for more payload intelligence and autonomy, more real-time user control, more complexity in the interactions possible between activities aboard one or more spacecraft, etc. In designing a scheduler for spacecraft operations a number of as yet unsolved problems arise as a product of various interactions among experiment and systems requirements, constraints on ground and spacecraft systems capabilities, and so on. The degree to which these problems can

be solved will significantly affect how well spacecraft management is carried out in the 1990's and beyond.

Scheduling, as defined in this paper, consists of fixing the execution times of activities on a timeline, such that all constraints (e.g. resource requirements, environmental conditions, etc.) on these activities are met. This can be contrasted with the definition of planning, in which a set of operations is ordered such that a desired goal state is reached. A scheduler assumes the orderings for operations are fixed, and does not have the knowledge or mechanisms necessary to order them.

It often happens that a partial or completed schedule will prove to be in need of revision due to changes in mission requirements or resource or conditions availabilities. Making the required changes to a schedule, including unscheduling, is also part of the scheduling process.

The scheduling problem is extremely difficult for several reasons. The most critical factor is the computational complexity involved in developing a schedule. The size of the search space, the space of possible schedules, is large along some dimensions and infinite along others. There can be an infinite number of ways to place a single activity, and a large number of choices of crew assignments to activities, for example. Additionally the goal of the scheduling process is ill-specified - the requirement is to produce a "good" schedule, one which meets a number of often conflicting requirements. These requirements can include efficient use of resources, no time or resource constraint violations, and maximum production during a specified time period, for example. There exist many additional factors that make scheduling a difficult problem, e.g. there are interactions between particular activity placements and resource usages that make constraint violations difficult to predict and avoid.

The specific requirements of a scheduler for spacecraft in the Space Station era have not been defined, and are expected to evolve as spacecraft and instruments become more complex. Thus part of the scheduling problem is to create a system which can schedule within a number of possible operational scenarios and management approaches. The next sections discuss some of the solution methods implemented in MAESTRO.

SCHEDULING TECHNIQUES

The approach taken within MAESTRO to scheduling involves a representation of scheduling objects and operations which generate schedules based on the relevant information. (An expanded description is given in [Britt, et al 1986] and [Geoffroy, et al 1987]). Objects of the scheduler include models of the activities to be performed, and models of all relevant constraining resources and conditions. Activities within MAESTRO are modelled as ordered series of subtasks, each

of which requires a set of resources and conditions which does not vary over the duration of the subtask. The duration of each subtask, and any delays between them, can vary. There are several types of constraints which can be considered within MAESTRO. These include resources such as crew time, electrical power and pieces of equipment, consumables such as water and liquid nitrogen, and conditions such as ambient temperature, vibrational stability and spacecraft attitude. The scheduling operations within MAESTRO involve repeatedly executing a selection-placement-update cycle, in which an activity is chosen, the activity is placed on the schedule, and resource availability profiles are updated to reflect that placement.

Selection of an activity to schedule on each cycle is based on heuristics such as relative constrainedness of activities, the priority assigned to each, and the success level, defined as the ratio of performances scheduled to those requested, for each. These criteria are combined using weightings which reflect the relative importance of each of these factors. Selection is thus based on several characteristics of the activities in relation to the current partial schedule. Two of these characteristics, priority and level of success, are calculated in a straightforward manner. Relative constrainedness is a more involved measure.

Relative constrainedness of activities can loosely be defined to be the number of performances of each activity that could be placed given the current partial schedule and resource availabilities. The system first obtains a rough measure of placement opportunities. In order to obtain this rough measure the system maintains knowledge of all possible placement alternatives for every subtask in each activity considered for selection, taking into account resource requirements, subtask temporal specifications, and a number of other factors. This process, called viable intervals calculation, results in a set of time windows for each subtask during which all of the conditions for the operation of that subtask are met. These windows are pruned to take into account temporal constraints between subtasks within an activity as well, but the process achieves only a good approximation to the specification of all and only those subtask time windows which are possible.

A second process, temporal constraint propagation, based on a technique developed for scene understanding by Waltz [1975], further refines the specification of placement opportunities providing an exact measure of all possible start and end times for each of the subtasks. This function handles a variety of constraints on the start and end time points, including minimum and maximum durations of all subtasks making up an activity, delays between subtasks, duration of each performance of an activity, delays between performances of an activity, starting and/or ending time windows for activities or subtasks imposed by mission requirements, and the set of ordering relations between activities enumerated by Allen [1983] such as precedes or follows. The result of these two processes specifies all and

only those points on the timeline which are candidate start and end times for each of an activity's subtasks. These results can be used to measure constrainedness - how hard it will be to find a place on the timeline where the activity can be scheduled meeting all of its constraints.

Once an activity has been selected for scheduling there are typically a large number of times each subtask could start or end. This necessitates making use of placement heuristics appropriate to each activity, determining the placement of the activity in relation to the overall scheduling time period, maximizing or minimizing subtask durations, minimizing or maximizing delays between subtasks or between performances of the activity, and placing the activity in relation to other activities already scheduled. In determining exact placements, these placement heuristics are used in conjunction with the Waltz function described above to prune possible placements down to a unique specification of each subtask's start and end times.

Unscheduling may be required for a number of reasons - e.g. a new high priority item may need to bump some previously scheduled activities, or there may be a downward revision in projected resource availabilities. In these cases, a number of factors must be considered when deciding which performances of which activities must be taken off the schedule. Heuristics for unscheduling when constraints are violated include goodness of fit between activity resource use and magnitude of resource overbooking, base priorities of activities, dependencies between activities, other opportunities to place each activity, the ratio of performances scheduled to requested for each activity, interruptibility and restartability of each, and so on. As with selection for scheduling, these factors are combined with weightings and compared to determine which performances to unschedule.

These and other automated decision-making functions are complemented in MAESTRO with a highly interactive user interface, allowing the user to choose the level of interaction or intervention in the scheduling process that he desires.

DESIGN ISSUES

Consider the contrast between two operational scenarios - one for control of unmanned orbiting platforms with numbers of simple instruments, the other for control of experiments in a Space Station core module. In the first scenario, it is likely that control will be geographically distributed, schedule development will occur relatively close to actual schedule implementation (espescially for those experiments determined by targets-of-opportunity, or recent atmospheric or political occurances), significant on-going schedule revision will be required, and resource availabilities and requirements may be somewhat unpredictable. In the Space Station scenario, experiment planning and scheduling will be much more fixed, the

environment will typically be more predictable, and scheduling may tend to be more centralized. For these two scenarios scheduling philosophies may differ radically - e.g. resource envelopes may be used which exactly specify the resources which will be used for an activity, or may reflect a loose operational envelope in which the activity must fit; the system may host a single user or a variety of user types, each with different requirements and different levels of authority for scheduling decisions; and scheduling may occur either as a batch or an incremental process. For these different scenarios, the core scheduling problem remains the same - what differs is the implementation of the interfaces surrounding the core scheduling system.

Because these and future interfaces may differ, the MAESTRO system has been developed such that the core scheduling functions are independent of the transactions that interact with the scheduler. The scheduling core does not differentiate between interactions with a user on the host processor, a transaction log on a file, or a user utilizing a workstation in a different location. MAESTRO and its interactive displays may be implemented on a single processor or may function as the scheduling node in a larger network of computers and/or systems. Further, processes external to MAESTRO may be used to directly or indirectly enforce the appropriate philosophy. The scheduler has different selectable options for scheduling in batch or incremental mode. External processes can determine which users are allowed to perform which operations on the scheduling system. Decisions regarding how loosely or tightly resources will be assigned can be determined by the way in which activity resource requirements are modeled, and how closely the resource profiles provided to the system reflect the actual resource availabilities. This design permits the implementation of interfaces appropriate to various scheduling philosophies and viewpoints while maintaining the core capabilities of the scheduler.

CONCLUSION

Scheduling is a difficult problem. The complexity of the scheduling problem can be overcome by heuristic decision-making, temporal constraint propagation, maintenance of multiple schedules and resource availability profiles, and other techniques. The problems introduced by the considerable variability in possible operational scenarios can be vitiated by separation of the scheduler from its interfaces, use of multiple asynchronous processes, prioritized command management, and intelligent preprocessing of scheduling requests. These and other techniques are implemented in the prototype scheduling system MAESTRO, Further work is necessary to refine these techniques and make them execute more efficiently, but a solid base has been laid for scheduling in the Space Station era. MAESTRO appears to be a suitable vehicle both for future research and as a starting point for production software.

REFERENCES

1. Allen, J.F.,"Maintaining knowledge about temporal intervals", Communications of the ACM, v.26 no. 11 pp.832-843, November 1983.

2. Britt, D.L., Geoffroy, A.L., and Gohring, J.R., "A Scheduling and Resource Management System for Space Applications", Proceedings of the Conference on Artificial Intelligence for Space Applications, Huntsville AL, November 1986.

3. Geoffroy, A.L., Britt, D.L., Bailey, E.A., and Gohring, J.R., "Power and Resource Management Scheduling for Scientific Space Platform Applications", Proceeding of the 22nd Intersociety Energy Conversion Engineering Conference, Philadelphia PA, August 1987.

4. Waltz, D., "Understanding line drawings of scenes with shadows", in P. Winston (Ed.), The Psychology of Computer Vision, 1975.

N88-16427

The Resource Envelope as a Basis for Space Station Management System Scheduling

by
Joy Bush and Anna Critchfield
Computer Sciences Corporation, System Sciences Division
4600 Powder Mill Road
Beltville, Maryland 20705

Abstract

This paper describes the Platform Management System (PMS) Resource Envelope Scheduling System (PRESS) expert system prototype developed for space station scheduling. The purpose of developing the prototype was to investigate the resource envelope concept in a pratical scheduling application, using a commercially available expert system shell. PRESS is being developed on an IBM PC/AT using Teknowledge, Inc.'s M.1 expert system shell.

The research includes a proposed definition of the content of the resource envelope, and examination of the resource envelope's flexibility and limitations for scheduling. Our definition of the resource envelope includes time parameters, resource usage, and constraint considerations. The suggested format is exercised by the PRESS scheduler, which performs both initial planning and replanning, fulfilling two of the functions currently defined for the PMS: short-term planning and constraint conflict resolution.

I. Introduction

PRESS is a prototype expert system developed as part of an effort to study the feasibility of using expert system technology in the Space Station environment. PRESS implements some of the functions that have been defined for the PMS, and uses the not yet fully defined "resource envelope" concept that has been developed for Space Station applications.

In a previous article [1], presented following completion of the rapid prototype, the authors described in detail the PRESS system functionality for the rapid prototype, and planned full prototype capabilities. Since then, the full system prototype, with finalized functionality and key concept refinement, has been developed and demonstrated for NASA representatives. In the present article the existing PRESS capabilities will be briefly described for familiarization purposes, but the main emphasis will be placed on those assumptions, concepts, and technical approaches which were developed and/or finalized after the PRESS rapid prototype demonstration and the previous publication.

II. Discussion

The goals of PRESS are: to research the feasibility of expert system applications in providing PMS functionality; to use (and

therefore help define) the resource envelope concept as a basis for automatic scheduling; to use realistic examples like the Cosmic Background Explorer (COBE) and the Upper Atmosphere Research Satellite (UARS) spacecraft scheduling needs as a proof-of-concept; and to evaluate the suitability of the system development environment for expansion of this prototype or developement of an operational system.

The PMS [2] is a software system that provides operational management services among payloads and platform systems for the space station. PMS consists of an automated on-board segment, the Platform Management Application (PMA), and a ground segment, the Platform Management Ground Application (PMGA).

PRESS addresses two of the PMS functions. The first of these is the Short-Term Plan Management function. This function involves the PMGA receiving a plan from the Platform Support Center, and uplinking appropriate portions to the PMA. The PMGA and the PMA may receive plan changes requested by operators, customers, subsystems, and payloads. The second function, Conflict Recognition and Resolution, involves the monitoring of resource usage, allocation, and margins. Conflicts for resource usage are to be resolved on a priority basis. This function will be used in deciding whether a given request may be scheduled. The PMA and the PMGA are required to modify the short-term schedule while maintaining a conflict-free plan that does not exceed the platform's resource capabilities or compromise its safety.

A final PRESS prototype system was completed and demonstrated for NASA on September 3, 1987. Some functions which were implemented in the rapid prototype were removed from the full prototype due to memory limitations. Both PRESS prototypes taken together serve as a proof-of-concept for all the defined functions. The following functions are implemented by PRESS.

- Perform initial scheduling, processing requests by priority
- Perform rescheduling
- Accept schedule modification requests as resource envelopes
- Accept schedule modification requests as changes to resource availability and/or constraints
- Resolve schedule conflicts based on assigned envelope priorities
- Perform conflict checking
- Accept multiple envelope requests
- Place scope of user interaction at operator's discretion
  - provide advice for modifications to the resource envelope requests that would permit successful scheduling (rapid prototype version only)
  - provide capability for operator to cease processing before completion of input file
- Perform input error checking
  - on user input data file via preprocessor
  - on legality of interactive query responses via M.1 capabilities
- Provide graphic and textual representation of system output

Brief definitions of some terms are presented here to avoid confusion with possible active usage elsewhere. An "activity" is the item being scheduled. It may be anything from a complex scientific experiment to a single use of a communications link. An activity may consist of more than one event, with each event represented as one "resource envelope". A request for scheduling an activity is represented to PRESS in the form of one or more resource envelopes, together with an activity header. Each "resource envelope" is equivalent to an event and represents a time period, one or more resources whose use is required, and a usage level for each resource. Resource usage is assumed to be stable for scheduling purposes. Examples of "resources" are power, an instrument, a communications link, etc. The "schedule" or "schedule timeline" is produced by the system to show what activities have been scheduled within a given time period. PRESS output shows the activities plotted against the resource used over time, so that the schedule timeline is actually a set of parallel timelines, one for each resource, with usage periods identified with an envelope identifier. PRESS accepts user input interactively or from a stored file, and is capable of both initial and rescheduling functions, including rescheduling as required by changes in resource availability or operational constraints. Requests are scheduled based on externally determined priorities. Higher-priority requests receive preferential treatment during scheduling. This includes the automatic deletion of already scheduled lower-priority requests if needed to successfully schedule a new higher-priority request. (See detailed description of PRESS functionality [3]).

The presumption at the outset was that the resource envelope concept would permit a global view of resource usage, which is important for shared resources like power or communication links. A global view provides protection against oversubscription, especially in concurrently used resources.

PRESS implementation of the the resource envelope concept included the following major points: (a) the activity is viewed in terms of resource consumption; (b) within a resource envelope, resource usage is considered constant, for scheduling purposes, over the time period of the envelope; (c) activities which vary in resources used, or dramatically in level of resource usage, must be broken into separate events, with each event represented by a resource envelope; (d) envelopes in one activity may not overlap in time, and events within an activity are defined by the user in chronological order; and (e) activities may overlap in time, and use the same resources, availability permitting.

PRESS is implemented as a production rule system in M.1 by Teknowledge. The knowledge base contains approximately 300 entries; about 145 of them are rules. PRESS represents resources and constraints as lists, but treats them as virtual objects. The representation includes: resource identifier; start and stop times; amount of resource; and an event identifier (or nil) field. Constraints are represented in a similar manner to the resources. The resource and constraint objects are dynamically created,

deleted, divided or combined by PRESS during the process of scheduling. Resource and constraint identifiers are not "hard-coded" and are transparent to PRESS. Resource levels can be represented as absolute units or as percentages of use, as long as the convention is maintained consistently within one resource. These two characteristics afford a maximum flexibility; the intent was to make PRESS as generic a scheduler as possible, easily applied to a number of specific applications. Requests for scheduling include an activity header and one or more request envelopes. The activity header specifies the priority and type of the request. The request envelope is implemented as a list, containing the following fields: activity identifier; envelope identifier; start time and duration windows; resources and amount required; and constraints generated and avoided.

The initial test application of PRESS was to schedule COBE communication link usage. The communication links themselves were considered as the resources (uplinks, downlinks, and links via TDRSS or ground links were treated as separate resources). Line-of-sight times determined resource availability. The COBE example had no constraints. PRESS could easily perform scheduling in this case, but the example was too limited to exercise all of PRESS's capabilities.

The UARS observation instrument scheduling was selected as a more complex problem. UARS contains a platform with three separate instruments capable of performing solar or stellar observations. Although the instruments make independent observations, the attitude of the platform, which all share, determines what object may be observed at any one time. Resources included power, the instruments and the platform. Initially, in a manner analogous to the COBE approach, the sun and stars were also treated as resources to be "used" by being observed. However, it was found that the presence of the sun, for example, constrained stellar observation. This meant that even if the sun was viewed as a resource, it must be considered as a constraint as well. Alternatively, if solar availability was treated as a constraint solely, it became necessary to specifiy both the sun's presence and absence as constraint objects, since solar observations could only occur in the presence of the sun, and certain stellar observatons only in the absence of the sun. It was felt to be more consistent to represent solar availability as a constraint than to have it appear as both a constraint and a resource. Even so, it was difficult to create a test scenario to accurately reflect a realistic activity in UARS terms. Such an activity might include slewing the platform to obtain a sighting, opening instrument shutters, placing certain filters, and taking the observation. It was difficult to define this with resources limited to power, the platform, and the individual instruments, at least not in any obvious one-to-one correspondence with the actions taking place. And although PRESS can express concurrent use of resources, it cannot express truly shared use. An example is the need to say that if a scheduled activity already has the platform slewing toward the sun, a second activity requiring the same action need not use resources to do it, but rather can "share" the existing action.

III. Conclusions

PRESS is a prototype system, with certain conditions and simplifications assumed. However, conclusions, and problems the authors have encountered during system development may be applied to operational systems which use the resource envelope concept in a similar technical environment.

The authors feel that the resource envelope as a theoretical concept is very useful; it provides a global view and permits tight control over concurrent usage of shared resources. However, resource envelopes do not seem to be straightforward in practical application because of the need for additional definition and assumptions, such as determining what should be considered a resource and what should be considered a constraint. Another difficult decision involves breaking down an activity into envelopes, which must take into consideration "wastage" of resources due to the assumption of a constant level of use against the complexity of defining and processing the request.

From this perspective, the resource envelope concept may be needed in an environment similar to space station environment. It is not as easily used as the exclusive scheduling approach for individual payload activities. The optimal balance between the resource envelope approach and a more traditional commanding scheduling approach in a multi-payload application awaits further investigation.

Acknowledgements

The authors wish to acknowledge Ed Beach, Karen Leavitt, Gail Maury, and Audrey Loomis (CSC), Steve Tompkins, Steve Wadding, and Larry Zeigenfuss (GSFC code 511), and Brian Keeler (Bendix Field Engineering Corporation), for their encouragement and helpful discussions. This work was funded by the National Aeronautics and Space Administration, Goddard Space Flight Center for the Mission Operations Division (GSFC code 510) and the Data Systems Technology Division (GSFC code 520) under Contract NAS5-28620, task assignment 319.

References

1. Bush, J.L., A. Critchfield, and A. Loomis, "Space Station Platform Management System (PMS) Replanning Using Resource Envelopes", May, 1987.
2. Goddard Space Flight Center (GSFC),"The Platform Management System Definition Document", October, 1986.
3. Computer Sciences Corporation (CSC), "Space Station Platform Management System (PMS) Resource Envelope Scheduling System (PRESS): Prototype Expert Scheduling System for the Multisatellite Operation Control Center (MSOCC)", September, 1987.

N88-16428

Prototype Resupply Scheduler

Steve Tanner and **Angi Hughes**
General Research Corporation
635 Discovery Drive
Huntsville, AL 35806

Jim Byrd
United Space Boosters Incorporated
188 Sparkman Drive
Huntsville, AL 35805

Abstract

Resupply scheduling for the Space Station presents some formidable logistics problems. One of the most basic problems is assigning supplies to a series of shuttle resupply missions. Some supplies relate to life-support, others are required by critical experiments, and still others are necessary for routine maintenance. If an emergency occurs or the space station inventories are depleted unexpectedly, resupply plans must be quickly adapted. The speed at which a logistics expert can replan schedules is a critical factor in the successful operation of the space station. The logistics expert requires a great deal of knowledge to construct a resupply schedule which satisfies the life-support, experiment-support, and maintenance constraints.

The Artificial Intelligence Department of General Research Corporation (Huntsville) with the logistics expertise of United Space Boosters Incorporated constructed a prototype logistics expert system which constructs resupply schedules. This prototype is able to reconstruct feasible resupply plans and, in addition, analysts can use the system to evaluate the impact of adding, deleting or modifying launches, cargo space, experiments, etc.

1 Introduction

In this paper the Heuristic Assistance in Tactical Scheduling (HATS) prototype system is described. This system was built by Steve Tanner and Angi Hughes of GRC with domain expertise provided by Jim Byrd of USBI. It was implemented on a Symbolics 3670 lisp machine with the use of the KEE software tool. A more advanced implementation will be undertaken using tools developed in house and which will work in VAX and PC environments. The system, while still very much in its infancy has proved viable for small scheduling tasks, and with time will work with larger more complex problems.

PRECEDING PAGE BLANK NOT FILMED

The system is general enough in nature to help with many types of scheduling and logistic problems, however, the main problem domain addressed by HATS is the supply and resupply payload schedules for shuttle launches necessary to maintain the space station.

Some of the techniques that HATS incorporates are: object oriented structures that take full advantage of inheritance facilities, hypothetical reasoning features that allow the system to try different solution paths without data corruption, heuristic methods to allow for some level of control and constraint of the immense search space that scheduling problems often generate, and rule based reasoning to help with the capture of domain expert knowledge.

2 Structure of the Knowledge Base

By using a rich object oriented environment in which to implement this system, a great deal of reusable and interrelated information can be stored in a concise, consistent and understandable manner. The two main categories of objects dealt with are the supplies that must be scheduled on launches and the launches themselves. The supplies that are to be scheduled include a broad range of supplies necessary to operate and maintain the space station, support experiments and deploy, retrieve and maintain communications and observation satellites. This means that criticalities I, II, and III items are scheduled as well as the experiment containers, platforms and any equipment that the experiments themselves need. Extra crew and scientists that a particular experiment needs are simply represented as another necessary supply.

Thanks to the use of inheritance, much generic information about each of these categories is defined in their high level object abstraction. The specific information about each individual launch and supply is stored in each individual object instantiation. Stored information for launches includes such data as projected launch date, maximum payload size and weight, crew size (excluding experiment scientists), and other pertinent launch information. Supply information is considerably more complex. The supply objects have basic size and weight information, but in addition, they have date needed by, and date needed after information. Also, because of the interrelated nature of some supplies, relationships between supply objects are also possible. For example, as mentioned above, extra crew needed for a particular experiment are scheduled as another supply. It would do little good to schedule the extra crew for one launch and the experiment equipment for another. There must be some way for each supply to know what other supplies it requires. This type of information is easily stored in attribute slots on the objects themselves.

As another more complicated example of supply interrelationships, there can be several scheduling paths based on interrelationships of supplies. In the above case, the experiment equipment could indeed go up on a launch prior to the experiment crew. The equipment could sit dormant on the space station until the required personnel finally showed up. It is also possible, but

not likely, that the crew could go up before the experiment equipment and sit around until it arrived. This scenario happens quite frequently with field service personnel back on earth. The three different schedules mentioned here (crew and equipment on one launch, equipment first, crew first) are all three possible and even workable supply schedules. Ways to make the system favor one schedule over the others is discused a little later on, but the ability for objects to have defined relationships with one another is a great help in sorting out these problems.

Because objects in this environment allow for methods, these supply objects can be made to automatically make changes of themselves and other objects as the data changes. (Methods are like programs that can be run when certain events occur.) For example, adding an extra crew member for an experiment will mean that more food and oxygen will be consumed during a specific time period. This means that there must be a corresponding change in the food and oxygen supply objects. Either current objects must be increased, dates shifted, or new objects created. This type of automatic readjusting can be handled by methods that are keyed to run whenever supply objects are added. Also methods can be used to create supply objects to help with cyclic types of resource use. For example, the user may tell the system that 100 pounds of gas x is used per month. The methods may automatically generate the necessary gas x objects to make sure this requirement is met. Direct down links will eventually provide on-orbit inventory quantities against which the system will automatically plan replenishment payloads.

Another type of interrelationship of supplies is one supply with a fixed date of need and another supply with no known date, but with a known link to the first. Because the first supply has a fixed date, the second supply must be scheduled either before or concurrent to the first supply. This means that even though the second supply has no obvious date itself, there is a necessary scheduling date inherent in its relationship with the first supply.

The initial state of the knowledge has the basic set up of the supply and launch data. The user of HATS has a graphical user interface to help with this initial set up. The dates placed on the supplies is optional, and as the system works, any unknown dates are taken to mean that the supply can go on any launch, unless of course there are relationships with other supplies.

3 Hypothetical Worlds and Rules

Once the initial data has been set up, the system tries to find a way to place all the supplies on the limited number of launches available. This is accomplished by the use of a rule base and two techniques known as hypothetical reasoning and worlds. As rules in a rule base are fired, they generally assert new facts. In hypothetical reasoning these new facts are considered hypothesis rather than true facts. These hypothesis are asserted only in separate autonomous worlds. In this way, if a trail of asserted facts leads to a dead end, there is no need to retract all the facts. Often it is

impossible to retract a fact, and at the very least it is time consuming and difficult. With hypothetical worlds, if a path leads to a dead end, the useless worlds are ignored or thrown away, and no facts need to be retracted.

As the system runs, a trail of worlds is created as a tree. Each world leads to at least one more world until a dead end is reached. A dead end means that either a workable schedule has been found and the system can quit, or this current path is unworkable and another must be found. Each world has one supply placed on one launch. If there are 10,000 supplies to be scheduled, a branch of worlds with a depth of 10,000 nodes represents a workable solution. If the branch does not go 10,000 deep, then some supplies would not fit the current launch configuration.

These worlds can be used as a simple brute force method for finding all possible scheduling solutions. It is a simple matter to try all supplies on all launches until a complete path is found. However, this is combinatorial and even with the use of several super computers, would require an unreasonable amount of time. This is especially problematic for last minute payload changes. That is where the use of the rule base can really pay off.

Because rules are used to create the world tree, they can help to trim the tree and eliminate many of the paths that will probably lead to unworkable dead ends. As a very simple example, scheduling larger items on launches before looking at the smaller items will point out dead ends far faster than the other way around. Also Criticality I items can be scheduled before other items so that if only partial schedules are found, then the critical items are scheduled and optional equipment can be left behind.

Another advantage with the combination of worlds and rules is that new ideas can be tested quickly and effectively. It is fairly easy to change or add rules to the rule base. If a change speeds things up and finds solutions faster, then the new technique can be left in place. If not, then the changes need only be taken out of the rule base.

4 Future Work

As funding allows, work will concentrate on several areas. The rule based heuristics that help trim the search tree will be improved considerably. The current rule base takes about thirty criteria into account. As domain expertise is codified, this will improve and expand. This expansion will be aimed specifically at speeding up the system by trimming unfruitful branches off the world search tree.

The interrelationships between objects will be expanded. This area is fairly rudimentary now and should be revamped and improved. Many types of relationships have been defined and need only be incorporated into the system. The infrastructure is already in place to do this, and is flexible enough to take new relationships into account as well.

The system will be rewritten to use an Entity Attribute Relationship database and rule system that we are currently working on in house. This system will be considerably faster than the current knowledge base and rule system now being used. During this rewriting, a more generic form of HATS should emerge. This generic tool will be useful as a baseline for other types of scheduling problems.

An improved user interface will be implemented. The current interface is very useful for creating launch and supply objects and placing them in the object hierarchy, however there are several things that need to be added. The interface needs to make defining relationships between objects easier and filling in slot attributes easier.

The entire system will be ported to more mainstream types of machines. The Symbolics is a very good environment in which to develop systems like this, however at this time it is unreasonable to assume that eventual users of HATS will have such equipment on their desks.

More forward looking analysis techniques should be added to the system. For example, if no complete schedule can be found, the system could suggest other solutions. It might determine what items would cause the least trouble if left behind, or suggest the fewest number of additional launches that would alleviate the problem.

5 Conclusions

The HATS system has shown that several AI techniques are useful when applied to scheduling problems. Basic object oriented structures are good for setting up the types of data that are required and in defining the relationships between the data. Objects that represent launch and supply items are easy to manipulate and alter. A world tree is a useful way to represent the planning paths taken while determining a schedule. Rule based reasoning is very effecting in both creating the tree and constraining the search paths that the tree represents. The system is flexible and can expand as domain expertise is incorporated. Application of this system to logistics support will not only improve support response and effectiveness, but also provide a valuable tool for future program planning.

N88-16429

Neural Network Based Speech Synthesizer: A Preliminary Report

James A. Villarreal
Artificial Intelligence Section/FM7
Lyndon B. Johnson Space Center (JSC)
NASA
Houston, Texas

Gary McIntire
Advanced Systems Engineering Dept.
Ford Aerospace
Houston, Texas

Abstract

The growth and development of our goals in space utilization will reach out to utilize every possible technology. Artificial Neural Systems (ANS) is a newly emerging technology which has already indicated a potential solution to many space engineering problems. A particularly interesting feature of ANS's is their ability to construct vital generalizations or inferences from sample data without the need for conventional programming. In order to evaluate ANS's, the Artificial Intelligence Section is conducting several initial projects implementing ANS's, developing a dedicated ANS workstation, and developing applications to assist with the immigration of this technology.

This paper will describe the neural net based speech synthesis project. The novelty is that the reproduced speech was extracted from actual voice recordings. In essence, the neural network (NN) learns the timing, pitch fluctuations, connectivity between individual sounds, and speaking habits unique to that person. The parallel distributed processing network used for this project is the generalized backward propagation network which has been modified to also learn sequences of actions or states given a particular plan.

Introduction

An inherent nature to human behavior is its dependency on serial order. Our learning methods, speech, and chain of thoughts all follow a succession of events. The parallel distributed processing network being used by this project is a generalized backward

PRECEDING PAGE BLANK NOT FILMED

propagation network with the usual input, hidden, and output layers but with an added layer which learns a sequence of actions which are produced in a learned order given a particular plan. In describing the NN based speech synthesizer, the technique used to formulate the various characteristics in a persons speech is first discussed. An introductory explanation of the generalized backward propagation network along with its modification to learn sequences will also be provided. This will be followed with a description of the various components of the NN and their relation to the speech parameters.

Linear Predictive Coding

Humans produce speech by sending pulses of air (the vocal chords) through a resonating cavity (the vocal tract) which is constantly changing shape (velum, tongue, lips, and nasal cavity) to produce a wide range of sounds. The human vocal tract can then be modeled as a time-varying linear filter. The filter can be excited by a series of pulses to represent the pitch or the voiced segments of speech and white noise to represent the unvoiced segments of speech.

Linear prediction coding (LPC) is a technique which efficiently represents the speech signals in terms of a small number of slowly varying parameters[1]. Linear predictive analysis converts the combined spectral contributions of the glottal flow within a pitch period, the vocal tract, and the radiation at the lips into a single recursive (all-pole) time-varying filter. The transfer function in the complex Z domain of the LPC filter can be written as:

$$H(z) = \frac{GAIN}{1 - \sum_{k=1}^{p} a_k z^{-k}}$$

Thus the linear filter is completely specified by a scale factor GAIN and p coefficients $a_1, a_2, \ldots, a_p$. The linear filter has p poles which are determined on factors such as the length of the vocal tract, the coupling of the nasal cavities, the place of the excitation, and the nature of the glottal flow function. Nineteen poles and a 16 KHz sampling rate were selected to compute the LPC coefficients for this

project. Speech recordings were taken with the MIT developed software package: Speech Phonetic Research Environment (SPIRE) [2].

Ordinarily, LPC is used to compress speech on bandwidth limited communications channels. LPC analysis produces a set of all pole filter coefficients with an added "residual". The resultant residual is the "true" excitation source. To compress speech, this residual is usually discarded and replaced with an approximation of the gain, and an indicator of whether the segment is voiced or unvoiced. For voiced segments of speech the pitch rate is also expected. The approximations for the voiced/unvoiced decisions and pitch rate explain why the reconstructed signal sounds mechanized. At this point, it is important to understand that if the filter coefficients produced by LPC analysis are re-excited by the "true residual", then the waveform can be reconstructed without any degradation.

Parallel Distributed Processing

The parallel distributed processing network used in this project is a form of the "generalized delta-rule" known as "back propagation of error"[3,4]. In summary, the NN consists of a network of unidirectional processing units connected by weights. The state of each processing unit is determined according to the activation function:

$$x_j = \emptyset \left(\sum_{i=1}^{n} w_{ji} x_i + ¥_j \right)$$

where x_i is the activation function of the ith unit, w_{ji} is the weight from the ith unit to the jth unit, $¥_j$ is a bias associated with the jth unit, and n is the number of units in the network. A processing node whose output is fedback onto itself is termed *recurrent*.. This recurrence is the technique which provides a NN the capability to learn a sequence of events given a particular plan[5]. A NN with an input, hidden, state, and output layer are depicted in figure 1.

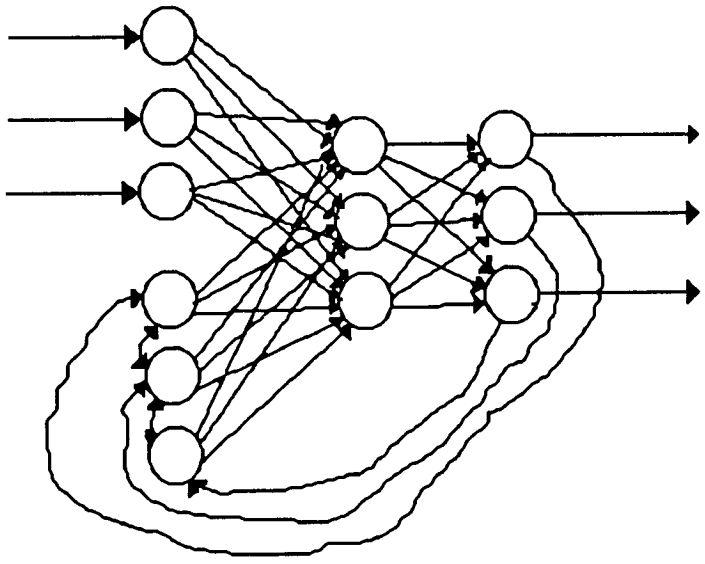


Figure1. Network showing input, hidden, output, and state layers.

Speech Synthesis Using a NN

The network produced here is similar to that produced by Sejnowski[6]. Unlike NETtalk which generates the phonetic translation for an English word, this synthesizer will produce the appropriate sequence of LPC coefficients and the spectrum of the residual for an English input.

To generate the data for the net, phrases which are rich in the English speech sounds were selected. For instance, for a phrase rich in the sounds [i] as in *see*, the phrase "Eva likes green peas" was selected. Other examples include the phrases "Ethel held that you lose your friends if you permit them to be in debt." for the vowel sound [e] as in *help* and "Alice planted three rows of asters." for the vowel sound [æ] as in *bat*. The attempt here is to generate a database with many examples of the various phonetic sounds in context with other sounds to capture the varying filter characteristics as phonemes are merged. After recording a phrase, the laborious task of segmenting and labeling the time based waveform with the appropriate letter follows. Having completed the segmentation and labeling of a phrase, the LPC analysis and spectrum for the entire phrase is computed. Several more phrases and 500 of the most commonly used words in the English language are still require[7, 8].

The NN is presently configured such that the input layers consist of 5 slots for the alphabetic characters and 2 special characters ("." and "?"). Training takes place by shifting a character

into the input layer at each cycle. The output layers consist of the 19 LPC coefficients and the spectrum of the true residual at a particular instant in time. Because the network is also learning sequences, each letter is represented by a series dependent set of 19 LPC coefficients and 256 spectral lines between 0 and 8kHz to represent the residual of the waveform. An added output layer is trained as a decreasing counter of the number of LPC sets for each letter. This output layer is monitored for a zero count to shift in a new letter.

Conclusion

The synthesizer is still under development, however, preliminary experiments have provided positive indications that the procedure will work. The experiments were conducted on the Symbolics computer. Due to the large size of this NN and the large data base, significant computer time is required. Currently under development by the AIS is a transputer based NN workstation. This workstation is being developed to satisfy the processing speed requirements inherent to most NN problems[9].

Acronyms

ANS	artificial neural systems
kHz	kilo Hertz (cycles per second)
LPC	linear predictive coding
NN	neural network

References

1. Rabiner, L. R., and R. W. Schafer. "Digital Processing of Speech Signals." New Jersey: Prentice Hall, 1978, pp. 396-453.

2. Kassel, R. H. "A User's Guide to SPIRE." Speech Communications Group Research Laboratory of Electronics, Massachusetts Institute of Technology, 1986.

3. Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning Internal Representations by Error Propagation." (Tech. Rep. 8506). La Jolla: University of California, San Diego, Institute for Cognitive Science, 1985.

4. McClelland, J. L. and Rumelhart, D. E. "Distributed Memory and the Representation of General and Specific Information", Journal of Experimental Psychology: General, Vol. 114, No. 2, pp. 159-188, 1985.

5. Jordan, M. I. "Serial Order: A Parallel Distributed Processing Approach." (ICS Report 8604), La Jolla: University of California, San Diego, Institute for Cognitive Science, 1986.

6. Sejnowski, T. J. and Rosenberg C. R. "NETtalk: A Parallel Network that Learns to Read Aloud." Johns Hopkins University, 1986.

7. Eisenson, J. "The Improvement of Voice and Diction." New York: The Macmillan Company, 1958.

8. Thorndike, E. L. and Lorge, I. "The Teachers Word Book of 30,000 Words." New York: Teachers College Press, 1972.

9. Mcintire, G., Villarreal, J., Baffes, P., and Rua, M. "Design of a Neural Network Simulator on a Transputer Array." Houston: Presented at Space Operations Automation and Robotics Workshop, 1987.

N88-16430

THE POWER OF NEURAL NETS

J. P. Ryan and B. H. Shah
Computer Science Department
University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

Neural net models work by simulating a collection of biological neurons and the interconnections between them. The learning abilities of their algorithms derive from ingenious ways to self-modify the connection weights. The specification of neural net models is done in terms of the characteristics of an individual node, the interconnection between the nodes, and the initial weights of interconnections and how they change. These models are based on the present understanding of how the biological neurons function. In this paper, we present implementation of the Hopfield net which is used in image processing type of applications where only partial information about the image may be available. Image-classification type algorithm of Hopfield and other learning algorithms, such as Boltzmann machine and back-propagation training algorithm have many vital applications in space.

1. INTRODUCTION

Neural net models are based on the present understanding of biological nervous systems since they offer many invaluable insights. Designing artificial neural nets to solve problems and studying real biological nets is changing the way we think about problems and lead to new insights and algorithmic improvements.

A neural network node as a model of a biological neuron is usually implemented as a non-linear processing element whose output is a non-linear function of input. Typically there are continuous input to and output from a node. An individual node is slow compared to modern digital circuitry, however, massive parallelism increases overall speed. An individual node weights ith input with a factor w_i and determines a weighted sum, S.

$$S = \sum_{i=1}^{N} w_i x_i$$

To each node is associated a quantity, theta, called internal offset, which determines a threshold above which the neuron represented by the node will fire. The quantity, alpha = S - theta, is passed through a non-linear function, f(alpha) to get the output. The three main types of non-linear functions are designated: (1) hard limiter (step function), (2) threshold logic element, and (3) sigmoidal non-linearity. Other more complex

non-linearities (based on time dependencies, time integration, operations other than summation) are possible but they cause increased computation time.

2. SPECIFICATION OF A NEURAL NET MODEL

Following three quantities are generally required to specify a neural net model [3]:

A. Net topology: This includes interconnections among nodes and number of layers of nodes (one or two or more).

B. Node characteristics: This includes offsets of individual nodes and the type of non-linearity.

C. Learning rules: These are concerned with connection weights and include initial set of weights and how weights should be adapted during use to improve performance.

As previously mentioned, neural networks achieve high computation rates due to massive parallelism. Each computational unit is simple. Because of a large number of processing units we have a high degree of fault tolerance (or robustness). Damage to a few nodes will not significantly impair overall performance of the system. Another important feature is the adaptation of weights ("learning") based on new inputs. This enhances the robustness because even if there are minor variations in the characteristics of processing elements, the overall performance is maintained.

3. CLASSIFICATION PROBLEM

The classification problem can be stated as follows: determine which of M classes is most representative of an unknown static input pattern containing N input elements. Such problems are of common occurrence in many situations. Examples include: (a) speech recognizer, where input patterns are spectra of sounds and output classes are corresponding vowels or syllables; (b) image classifier, where input patterns are gray scale level of each pixel for a picture, and output classes are symbols identifying corresponding objects; and (c) spatial locator: where input patterns are omni-directed range measurements and output classes are identifications of sub-regions.

A neural net classifier is characterized by parallel computations and parallel input/output. N input elements are fed in parallel over N analog lines. Inputs may be bits or continuous over a range. The network first computes matching scores and then selects the maximum score and enhances it so that only one most likely class will be selected. Neural net classifier can be made adaptive to new classes or exemplars by using a learning algorithm that will modify the weights of

connections as new classes are presented to the net. An implementation of a particular type of classifier is discussed in the next section.

4. THE HOPFIELD NET

The Hopfield network is designed for binary inputs. Examples of such situations include (a) pictures or images in terms of on-off pixels, and (b) ASCII representation of text with each character represented by 8-bits. Hopfield network applications are in associative memory and in solving optimization problems. In associative memory applications this network can locate the correct information from a supplied piece of partial information. When applying the Hopfield network the following limitations should be carefully considered.

(1) Spurious convergence: M should be small compared to N. If M is larger than about 15% of N, convergence may incorrectly occur to a pattern not matching any of the exemplars.

(2) Instability: A Hopfield net is said to be stable if upon using an exemplar as an input, the same exemplar is output. If too many bits are common between two patterns, the net becomes unstable.

We implemented the Hopfield network in Lisp on Texas Instruments Explorer which is a Lisp architecture computer. A run was made using two exemplar patterns with 171 pixels represented as *'s and -'s which are converted into +1 and -1 by the program, respectively. When the network is presented with a test input pattern with some of the pixels changed, it responds with the number of the exemplar which comes closest.

Another set of interesting runs was made with M = 2 exemplars and N = 10 pixels. We see that when only one pixel is different from the exemplars in the input pattern, the network does come up with the correct answers for the matching exemplar pattern. The algorithm converges to the correct exemplar pattern. The network was then presented with two input patterns which differed from both the exemplars in exactly the same number of pixels, namely, 5 pixels, half of all the pixels in each exemplar. The network does not converge to any of the exemplars, as would be expected.

We have also demonstrated what may be termed the "soldier's helmet" phenomenon. The network is presented with two different input patterns with only the first two pixels matching with either one of the exemplars. The network identifies the correct match and, in addition, the convergence is to the appropriate exemplar. This run dramatically illustrates the ability of the Hopfield network to reconstruct the whole picture from a partial one.

5. PLACE-FIELD AND GOAL LOCATION MODELS FOR A ROBOT

5.1. PLACE-FIELD MODEL

This model is based on the behavior of place-field cells of the hippocampus (one of two ridges along lateral ventricle of the brain) of a rat. Place-field cells fire at their maximum rate only when the animal is at a particular location relative to a set of landmarks. Such locations are called place fields. Zipser [5] developed a computational model to relate the configuration of landmarks to the location, size, and shape of place fields. The inputs to the model consist of configuration of landmarks in the environment together with the location of the observer. The output from the model represents the activity of a place-field neural unit in the observer's brain. A set of simple objects is used as place cues and the size of retinal images is indication of location. The model uses this information to provide quantitative predictions of how the shape and location of place fields change when the size, tilt, or location of these objects is changed.

The place-field neural model is designed for pattern recognition. It fires at maximum rate when the observer is at a desired location. A stored representation of a scene is compared to a representation of the current scene. Closer the viewer is to the stored scene, the better is the match. In determining the closeness, it is sufficient to use only a few discrete objects rather than the entire scene. A point P in two dimensional space can be uniquely located by its distance from three landmarks a, b, and c. It can be shown that in three dimensional space a point can be uniquely located by its distance from four landmarks.

When the robot is at a location P, the representations of the landmarks a, b, and c including the distance to P are recorded in some way in the memory. Upon return to P, the robot's sensory system can now generate a set of current distances. If these representations, other than their distance components, are not affected too much by the viewing postion, then the current representation of each object will differ from its stored representation only to the degree that the object's current distance from the observer differs from its distance to P. A neuron whose output is a summated measure of the similarity between these current and stored representations for each landmark will have the properties of a place-field unit.

5.2. CURRENT APPROACH

In contrast to the place location approach described above, we will not make the assumption of a "landmark recognizer" or any other sophisticated pattern recognition devices. Rather, our method will be dependent only upon the range sensor information.

The robot is assumed to operate in a two-dimensional region and to have range sensor detectors fixed in many directions from its center. The robot will travel translationally only -- so that it is always oriented towards a specific direction and there is no rotation. Many of these assumptions can be relaxed, as more sophistication is added to the model. The object of the robot is to explore the two-dimensional region to which it is confined. After sufficient exploration, it is able to navigate between any two points. As mentioned before the robot will rely only on range sensor data and will be unable to distinguish the angle at which it is approaching an obstacle. Movement of obstacles will be allowed and the robot will expected to navigate in a reasonable manner. So as not to become myopically confused by the obstacles and boundaries of the region, the robot will have a buffer distance always separating it from any other objects or boundary. The robot sensors are assumed to be of unlimited distance.

Navigation mode of the robot will start by moving from a "corner" in the direction of one of its sensors. The robot will assigned a predefined rectangular subregion, R, which is subdivided by the robot for exploration. The key factor in determining the "acceptability" of a subregion is whether the sensor vector varies continuously. An unsatisfactory subregion will be further subdivided into smaller regions so that the sensor vector is continuous. The robot gathers sensor data and data on change in position from each successive rectangular subregion.

In the navigational mode, the robot will have to periodically back up in order to insure that its connection weights are sufficiently tuned to discriminate one subregion from another. This is a highly unsupervised type of learning scenario, so that only certain types of neural net models would be acceptable. The work on learning neural nets [2], namely, Boltzmann machine [1,2], the competitive neural net, and sigma-pi neural nets [4] is being continued.

REFERENCES

[1] S. E. Fahlman and G. E. Hinton, "Connectionist architectures for artificial intelligence", IEEE Computer, pp. 100-109, January 1987.

[2] G. E. Hinton and T. J. Sejnowski, "Learning and relearning in Boltzmann machines", in "Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 1, Foundations", edited by D. E. Rumelhart and J. L. McClelland, M. I. T. Press, pp. 282-317, 1986.

[3] R. P. Lippmann, "An introduction to computing with neural nets", IEEE ASSP Magazine, pp. 4-22, April 1987.

[4] T. Maxwell, C. L. Giles, and Y. C. Lee, "Generalization in neural networks: the contiguity problem", to be publiched in the "Proceedings of the IEEE International Conference on Neural Networks", San Diego, Calif., June 1987.

[5] D. Zipser, "Biologically plausible models of place recognition and goal location", in "Parallel distributed processing: Explorations in the microstructure of cognition, Vol. 2, Psychological and biological models", edited by J. L. McClelland and D. E. Rumelhart, M. I. T. Press, pp. 432-470, 1986.

N88-16431

NEURAL NETWORKS AS A POSSIBLE ARCHITECTURE FOR THE DISTRIBUTED CONTROL OF SPACE SYSTEMS*

E. Fiesler
A. Choudry
Center for Applied Optics
University of Alabama in Huntsville
Huntsville, AL 35899

ABSTRACT

Future space systems are envisaged to be large, complex, multi-modular multi-functional systems. A large degree of intermodular autonomy and at the same time a high level of interconnectivity is expected. We have attempted to identify the features essential for such a system. These features have further been studied in the context of Neural Networks with the aim of arriving at a possible architecture of the distributed control system-specific features of the Neural Networks, and their applicability in space systems will be discussed.

*Work supported by Advanced Space Flight Systems of United Technologies.

N88-16432

A Data Structure and Algorithm for Fault Diagnosis

Edward L. Bosworth, Jr.
Computer Science Department
The University of Alabama in Huntsville
Huntsville, AL 35899

Abstract

This paper will present results of preliminary research on the design of a Knowledge Based Fault Diagnosis System for use with on-orbit spacecraft such as the Hubble Space Telescope.

This paper discusses a candidate data structure and associated search algorithm from which the Knowledge Based System can evolve. This algorithmic approach will then be examined in view of its inability to diagnose certain common faults. From that critique, a design for the corresponding Knowledge Based System will be developed.

Introduction

The research reported in this paper is focused on the development of an efficient fault diagnosis software system to be used in the operation of on-orbit spacecraft such as the HST (Hubble Space Telescope). There are several factors which indicate the need for an efficient fault diagnosis system. Among these reasons are 1) the desire to detect any fault before it causes damage to the spacecraft (an unlikely but possible event), 2) the desire to have confidence in the accuracy of the scientific data, and 3) the desire to schedule maintenance missions on some sort of cost effective time-line.

The scenario envisioned in this report is an automated fault diagnosis system monitoring the downlinked telemetry reporting on the health status of the spacecraft. This system would act as an intelligent assistant to the operations staff. It would detect abnormalities in the telemetry and deduce the hardware fault causing this indication. It should also be able to project trends in the data and give reasonable predictions of the remaining useful on-orbit time of any replaceable component.

The Data Structure and Associated Algorithm

The efficiency of a computational solution to any interesting problem depends, in general, on the choice of two items: a data structure appropriate to represent the structure of the problem and an algorithm which can operate efficiently on the selected data structure. This observation holds true even in those cases in which the algorithm is complemented by heuristic procedures.

PRECEDING PAGE BLANK NOT FILMED

The importance of the data structure arises from the following observation which is generally true: it is usually the case that the more appropriate the data structure the less complex the algorithm and associated heuristics. One criterion used for selecting a data structure is its resemblance to the human representation of the problem being investigated.

One of the primary human tools in fault diagnosis is a design schematic which shows the relations between the functional components of the system. There are two data structures which are analogous to a design schematic: a Directed Acyclic Graph (DAG) and a Tree. Both of these are hierarchical structures which can represent naturally the functional hierarchy found in design schematics. In the data structures this hierarchy is represented by "parent-child links" where a parent node can have several child nodes and represent the fact that a functional component in the design schematic can have a number of functional subcomponents.

Both a DAG and a Tree have at least one designated node which has no parent node. This node is called a "root node". A tree, by definition, has exactly one root node. A DAG may have many such nodes. The concept of a unique root node in the data structure has a strong analogue in the functional design schematic: the entire system. For this reason, the choice of data structures is limited to a Tree or a "Tree-like DAG", the latter being a Directed Acyclic Graph restricted to having one root node.

The main difference between a Tree and a Tree-like DAG is that the nodes of the former are constrained to have exactly one parent node whereas the nodes of the latter may have more than one parent node. This difference will impact the efficiency of representation of the design schematic. If a component is the design schematic can be a part of only one superior component, then the Tree data structure is the preferable representation. If a component may be a part of more than one superior component, then a Tree representation will have to duplicate nodes for that component, attaching a duplicate as a child node to each node which represents one of the superior components. Since the DAG does not require this artificial duplication, it is the data structure selected for this research.

In addition to the Tree-like Directed Acyclic Graph, an auxiliary data structure is used in order to improve the efficiency of the node creation algorithm. This auxiliary data structure is to be used to detect duplicate names and avoid the creation of duplicate nodes in the DAG. In order to use this auxiliary data structure, one must remember that a node in any graph has both an ID and a label. The ID is the variable by which the program accesses the structure representing the node. In LISP this would be called the structures "print name". The label is a name associated with the node. In this data structure, the name of the node is the name in the design schematic of the the component represented by the node. The requirement is to avoid generation of duplicate nodes for the same component in the schematic; i.e., nodes with the same label.

The auxiliary data structure chosen for this is a Hash Table which will store pairs of the form (key, value). The key for an entry will be the node label and the value will be the node ID; thus we have (Node-Label, Node-ID). Any attempt to create a node to represent a named component in the design schematic must first check the Hash Table. If the component be already represented, its Node-ID will be returned. Otherwise a new node may be generated and named as usual.

There is one restriction which must be observed when using an auxiliary data structure. Both the Directed Acyclic Graph and the Hash Table must be defined and accessed as a single abstract data structure. More specifically, there must be a single set of constructor and accessor functions which treat the two data structures and an interdependent pair and enforce the logical relations between the two. Otherwise, the data in the two will become inconsistent and thus useless.

The Directed Acyclic Graph was implemented in VAXLISP by use of the COMMON LISP structure. The following describes a node:

```
(Defstruct (Component
     (:conc-name Node-)
     :predicate)
"A node for representing a component in the design schematic"
  (Name Nil)              ;The name in the design schematic
  (Subcomponents Nil)     ;Note the default values.
  (Contained-In Nil)
  (Search-Seq  0))
```

The Hash Table was implemented as a COMMON LISP global variable.

```
(Defstruct *Component-List* (Make-Hash-Table :Size 197))
```

As mentioned above, the Directed Acyclic Graph and Hash Table must be accessed as a single abstract data type. A typical function is the one which creates a new node. It first checks the Hash Table to avoid making a duplicate. If it continues, it first updates the Hash Table and then creates the node.

```
(Defun Create-Component (Schematic-Name)
 "Creates a node to represent the component with
the specified name in the schematic"

    (Unless (Gethash Schematic-Name *Component-List*)
       (Let ((Node-ID (Gensym "NODE-")))
            (Setf (Gethash Schematic-Name *Component-List*)
             Node-ID)
            (Set Node-ID
               (Make-Component :Name Schematic-Name)) )))
```

Having established the data structure for representing the components, it is now time to discuss the design of an algorithm to do the searching required by fault diagnosis. This design actually has two such algorithms, SEARCH and SWEEP, built around the concept of a search sequence number.

Search sequence numbers are a generalization of the concept of node markings found in many graph and tree search algorithms. Node marks are generally thought of as Boolean variables having the values TRUE or FALSE. An alternate representation of the node mark would be a search sequence having only the permissible values on 0 or 1.

In the search sequence approach, there is a global variable which counts sequentially the searches undertaken during the current session. This variable is passed as an argument to the search procedure. As the procedure visits each node, it processes the node only if the nodes search sequence number is less than the current search sequence. If processed, the node is marked with the current search sequence, is expanded, and its subnodes evaluated for possible search.

The SWEEP procedure is called periodically to reset the search sequence of each node to 0 and to reset the global search sequence variable to 1. The nodes are visited by a simple Depth First Search. One should note that this exhaustive search is called much less frequently than the directed search mentioned above. This results in a reduced overhead due to calling SWEEP.

The algorithm SEARCH is a Best First Search with iterative deepening. It is called with two parameters - a node ID and a search sequence number. At each level, the node is examined to see if it is marked with the current search sequence number. If it be so marked, the next node in the search priority list is examined. Should the node not be so marked, it is given the current search sequence number and examined. Part of the examination is obtaining the subcomponent list and merging that with the rest of the search priority list to form a new search priority list. Nodes which appear more than once will be given a higher search priority on the assumption that if two failed components share a subcomponent then that subcomponent is suspect.

Conclusions and Directions for Future Work

While it seems obvious that an automated fault diagnosis system would be of considerable benefit in the operation of on-orbit spacecraft of the complexity of the Hubble Space Telescope, it is also apparent that an algorithmically based system will not be sufficiently sophisticated.

One flaw in an algorithmically based system is its inability to reason about faults that do not correspond to failed components in the design schematic. A simple example of such a fault is a bridging fault or short circuit, both of which represent components which are not present in the design schematic.

One of the major modifications which will be necessary is the design of a heuristic which can make reasonable modifications to the data structure representing the design schematic in those cases in which the observed fault is not consistent with the normal schematic. This heuristic will include representations of the physical components in order to postulate plausible bridging faults and short circuits. Such systems are discussed extensively in the research literature [1,2,3,4,5].

References

1. Davis, R.; Diagnostic Reasoning Based on Structure and Behavior; Artificial Intelligence 24 (1984) 347 - 410

2. de Kleer, J. and Williams, B.C.; Diagnosing Multiple Faults; Artificial Intelligence 32 (1987) 97-130

3. Genesereth, M.R.; The Use of Design Descriptions in Automated Diagnosis; Artificial Intelligence 24 (1984) 411-436

4. Keravnou, E.T. and Johnson L.; Competent Expert Systems, McGraw Hill, 1986.

5. Reiter, R.; A Theory of Diagnosis from First Principles; Artificial Intelligence 32 (1987) 57-95.

N88-16433

"Case-Based Reasoning for Space Applications: Utilization of Prior Experience in Knowledge-Based Systems"

James A. King
Member of the
R&D Staff, NCR Corporation *
1700 S. Patterson, WHQ-5E
Dayton, OH 45479

Abstract

It is imperative in resource-sensitive and critical command and control applications to provide automated systems with the proper amount, organization, presentation, and utilization of expert knowledge. The goal of this paper will be to describe Case-Based Reasoning as a vehicle to establish knowledge-based systems, based on experiential reasoning, for possible space applications. This goal will be accomplished through an examination of reasoning based on prior experience in a sample domain, and also through a presentation of proposed space applications which could utilize Case-Based Reasoning techniques.

Introduction

Much research has been conducted with the purpose of understanding and formalizing the representation and use of experiences in problem solving [1, 2, 6, 7, 9, 10]. Case-Based Reasoning, CBR, is an area of study which examines the role of experiential reasoning in human problem solving. CBR techniques provide the ability to define, store, retrieve, and process previous experience. These techniques have been used in various domains, i.e. legal reasoning, structure survivability, diplomatic interaction, and others [1, 4, 5, 8, 10].

The majority of the efforts to date have been in the academic environment. Case-based reasoning is just beginning to emerge as a viable method for providing rich, knowledge-intensive foundations for the production and operation of expert systems. For example a new initiative, led by DARPA, is under way to develop CBR systems for the Defense Department.

There exists a need for the encapsulation of prior experiences for many forms of problem analysis and decision aids. Some general domains where prior situational experience provides human operators with valuable insight are:

- Multi-sensor fusion and interpretation,
- Feature extraction and modeling,
- Diagnostic action and monitor facilities,
- Command, control, and communications,
- Training by Example or Tutorial Assistance.

Case-Based Reasoning, CBR, can be utilized as a method for structuring appropriate domain knowledge and processing capabilities to provide experiential reasoning in knowledge-based systems and conventional systems. For purposes of this paper the basic concepts of CBR will be discussed along with a description of an example domain. The last portion of the paper will present a list of possible applications related to the space program.

* *Research for this paper was completed and is ongoing through independent efforts. Contact for further information: (317)-478-5910 j.a.king@dayton.ncr.com*

PRECEDING PAGE BLANK NOT FILMED

Case-Based Reasoning and a Representation Framework

Case-Based Reasoning, CBR, has been used by experts in various fields to accomplish the generation, examination, and learning of situation-based actions, or problem solutions, which are based on a collection of previous cases of experience or hypothetical experiences. Dr. Edwina Rissland of the University of Massachusetts states: "Design is an excellent example of a domain where CBR techniques are used for complex problem solving, where new solutions are found through analogical transformations of past ones ..."

There are two basic types of CBR:

- Precedent-based CBR in which past cases are precedents which are interpreted to provide solutions, analyses, and explanations of present cases.
- Problem-solving CBR in which past cases are accessed to provide new approaches to present situations. Analyses and explanations are not provided as a product of the case-based reasoning process.

A Case-Based Reasoning System Structure

The following steps are appropriate as a guideline of the process for a case-based reasoning system [3, 6a].

A. Presentation to the system of a new situation in the domain of the case-based system.

B. Analysis of the new case to create the reference points for retrieval.
   - This may include a specific interface for the new case. Such as a forms interface.
     - A forms interface may provide the system with the capability to begin retrieval based on incomplete information.
     - The retrieval space may be reduced based on various attributes. As the form is filled out to describe the new situation, specific rules may apply to guide the user in defining the new situation. This guidance will also cause pruning of the amount of knowledge that the system will search through during runtime.

C. Locate and retrieve a potentially applicable case from long term memory.
   - Locate a suite of potentially applicable cases.
   - Provide a structure for a long term memory structure which will represent "worlds" of cases, or groupings, meta-sets of cases.

D. Determine which portion of the old case might be applicable to the present situation.
   - Determine which portion or portions of the individual cases may simulate or relate to the present situation.
   - Relate the combinations of portions to provide a relationship which can help solve the present problem.

E. Derive the targeted value for the current case through various interpretations and actions:
   - The role of the targeted portion of the previous case in the success or failure of the previously derived plan.
   - Choice between a previously-used inference method or the value.
   - Potential applicability of the above to the current case.
   - A "weave" technique will be needed to relate target locations from various cases.
   - Evaluate "real-world" constraints that are associated with previous case attributes for determining the "real-world" effects on the present situation.

F. Check the proposed value for consistency with the current case.
   - Further constraint checking based on "real-world" knowledge that is associated with the domain. (Common-sense checking)

G. Provide default inference techniques if no similar case is found or if the constraints associated with the reasoning about the new situation are not common-sense in nature
   - Results for the value(s) of the case can be computed by default inference techniques and updated in the case to provide a new experience for the system.

H. Update the representation of the current case.
   - Update any "ripple effects" which are caused through the determination of a value in the present case.

I. Establish the new case in the memory of the case-based system
   - Provide the proper links and definitions to encapsulate this knowledge and provide a learning experience for the computer-based system.

Ongoing Research in an Applicable Domain

The SURVER system, **SUR**vivability/**V**ulnerability Analysis Through **E**xperiential **R**easoning, is being built for the Air Force Weapons Lab, AFWL. The system will provide pre-test predictions for the survivability/vulnerability, S/V, of buried structures in nuclear blast simulations [4]. Currently pre-test predictions are based on finite-element analysis of blast effects on structural and material models and interpretation of the results by experts. Through knowledge elicitation it was apparent that the experts base judgements on previous experience with similar and dissimilar situations, or cases. The prototype is being built following a case-based reasoning design which provides for the definition, storage, retrieval, and processing of previous experience and will provide for automated S/V analysis.

A case-base is being assembled which consists of actual test data and the experiential factors which went into the analysis and solutions to the previous situations. A goal of the effort is a more in-depth case-base for proper utilization of the capability of case-based reasoning. Hypothetical cases will be produced through simulations of test situations which include variances to the following attributes: structure dimensions, material models, and the blast description. These hypothetical cases require that actual expert analyses be included within the case.

Another important aspect of this case-based reasoning system is that the CBR approach will provide the system with the ability to learn about new situations. Learning is defined in our prototype as the storage of a new situation along with the solution or analysis that has been provided through reasoning on past cases. A new case can be stored, after expert confirmation, through the definition strategy developed in the prototype. Cases will be indexed through structure type, blast type, and materials model type.

The very important aspect of reminding, or retrieval, in a CBR system will be accomplished in the SURVER system through two distinct methods. The cases will be indexed according to their representative attributes and an intersection method for retrieval will be used to retrieve the cases which best fit the present situation. As a refinement to the set of initial cases retrieved the system will perform a generalization function to provide retrieval through matching on not only specifics but also generalizations of values. This method is referred to as hierarchical domain-space retrieval. In this method a match can be provided on a minimum of information about the new situation through pruning of the search space via hierarchical representation.

The system is being developed following an object-oriented, frame-based approach on a Zenith-248 using the development system GoldWorks^{tm} from Gold Hill Computers.

Case-Based Reasoning and Space Applications

The following is an abbreviated list of possible applications of Case-Based Reasoning for the space domain. This list was developed through an examination of possible areas which depend on human experience for control and interaction [1a].

A. Satellite system and component survivability/vulnerability modeling and analysis
   - Fault diagnosis satellite, station, and ground-based subsystems

B. Communications support through knowledge-based operator support
   - Satellite control
   - Ground-based systems, positioning
   - Signal interpretation

C. Automation of prior astronaut experience in critical situations:
   - EVA's
   - Situation assessment
   - Attitude control
   - Diagnostics and Repair

D. Command and control functions
   - Both onboard and ground-based

E. Weather forecasting
   - Critical decisions for mission control
   - Feature extraction and analysis

F. Training
   - Tutorial assistance for C^3 applications
   - Ground support, etc.

G. Remote operations:
   - Decision processes for landers and robotic appendage control
   - Capture and retrieval operations
   - Space Station remote support facilities

H. Launch monitoring

I. Modeling and Simulation
   - Component design and testing
   - System and operations

Summary

It is apparent that knowledge and reasoning are products of expert experience. Case-Based Reasoning provides an appropriate representation strategy, methods for reminding or retrieval of prior experiences, and the techniques for manipulating and reasoning about prior cases. The SURVER project for AFWL is one example of the application of these techniques to real world problems and analyses. Similar systems can be built for the

space program to provide for the ever expanding needs for automation and integrity assurance.

References

[1] Bain, W. M. Case-Based Reasoning: A Computer Model of Subjective Assessment. Ph.D. Thesis, 1986, Yale University.

[1a] Faught, W. S. "Aerospace Applications of AI". In the Proceedings of the First AAAIC Conference, 1985, Dayton, Ohio, 331-344.

[2] Hammond, K. J. "Planning and Goal Interaction: The Use of Past Solutions in Present Situations". In the Proceedings of the National Conference on Artificial Intelligence, August, 1983, in Washington, D.C. 148-151.

[3] King, J. A. "Reasoning With Prior Experience: Providing Knowledge Based Systems a Case-Based Approach." Submitted to the 12th IMACS World Congress, 1988 Paris Conference.

[4] King, J. A. "Utilizing Hypothetical Cases in Knowledge-Based Prediction of Survivability/Vulnerability for Concrete Structures." To Appear in the Proceedings of the Simulation and Artificial Intelligence Conference, San Diego, 1988.

[5] King, J. A. "A Case-Based Reasoning Approach for Software Design and Reusability." In Press.

[6] Kolodner, J. L. Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model. Hillsdale, NJ: Lawrence Earlbaum Associates, 1984.

[6a] Kolodner, J. L., "Some Little Known Complexities of Case-Based Inference". In Proceedings of TICIP-86, August, 1986, 1-7.

[7] Kolodner, J. L., Simpson, R. L., & Sycara, K. "A Process model of Case-Based Reasoning in Problem Solving" in Proceedings of IJCAI-85, 284-289.

[8] Rissland, E. L. and Ashley, K. D. "Hypotheticals as Heuristic Device". In the Proceedings of AAAI-86, 1986, 289-297.

[9] Rissland, E. L., Valcarce, E. M., & Ashley, K. D. "Explaining and Arguing With Examples". In the Proceedings of AAAI-84, Austin, TX, 1984, 288-294.

[10] Simpson, R. L. A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation. Ph.D. Thesis, 1985. Technical Report No. GIT-ICS-85/18. School of Information and Computer Science, Georgia Institute of Technology.

N88-16434

KNOWLEDGE REPRESENTATION BY CONNECTION MATRICES : A method for the on-board implementation of large EXPERT SYSTEMS

A. Kellner
Space Systems Group
MBB/ERNO
Bremen, West Germany

ABSTRACT

Extremely large knowledge sources and efficient knowledge access characterizing future real-life AI applications represent crucial requirements for on-board AI systems due to obvious computer time and storage constraints on spacecraft.
In this paper a type of knowledge representation and corresponding reasoning mechanism is proposed which is particularly suited for the efficient processing of such large knowledge bases in expert systems.

1. INTRODUCTION

Many of today's AI systems are still experimental prototypes aiming at feasibility demonstrations. These systems normally have relatively limited knowledge bases, since research so far has had its focus more on the conceptual side of reasoning than on the treatment of large knowledge sources. However, it is becoming increasingly clear that for AI to display its full potential power not only the reasoning capacity of the human mind needs to be imitated, but also the fact that it has enormous knowledge sources at its disposal and is able to draw on this knowledge with extreme efficiency. Given the computer time and space constraints on spacecraft, this aspect becomes particularly crucial for on-board applications of AI and might necessitate, to a certain degree, a re-assessment of today's state of the art which has been mainly dictated by the endeavour to capture the reasoning aspect of intelligence only, regardless of the software overhead, high memory space demands, low performance, garbage collection problems etc. this often entailed.
In this paper a type of knowledge representation and corresponding reasoning mechanism is thus outlined, which is particularly suited for the processing of large knowledge bases in expert systems, maintaining the usual functionality of expert systems at the same time.

2. CONNECTIVITY IN THE BRAIN

Looking at the brain as the great example for processing speed and compactness of knowledge storage one notices, among other features :

f_1) extremely high performance in classification processes, i.e. fast mapping of large data sets on discrete descriptors (such as optical or acoustical data on corresponding objects or words)

f_2) relatively low performance in inference processes, i.e. comparatively slow generation of (long) chains of logically connected elementary operations (such as in doing maths problems step by step, where each step is usually solved or instantiated by a classification process "learned by heart", such as performing the simple mapping 2x2 = 4)

f_3) extremely efficient prompting of associated information (often quite unsolicited)

f_4) tolerance to incomplete or locally erroneous information in the classification process, such as in the identification of partially obliterated images, correction of misspelt words etc.

PRECEDING PAGE BLANK NOT FILMED

These features are supplemented by some knowledge about the brain's structure, such as :

s_1) incoming continuous data (e.g. optical or acoustical) is spatially discretisized prior to further processing by resonant excitation of dedicated sensor cells (e.g. in eyes or ears)

s_2) these sensor cells are connected by nerve fibres to neurons, which obviously act as logical gates to the discretisized data, being themselves interconnected by an intricate fibre structure

s_3) which also seems to allow for lateral inhibition, i.e. the weakening of the sensory input of neurons by neighbouring neurons with stronger sensory input, apparently supporting the generation of excitation maxima in the neuronal network.

An extremely simplified model inferred from these features could consist for example of a set $O = [o_1, \ldots, o_n]$ of "observation cells" o_j and a set $D = [d_1, \ldots, d_n]$ of "descriptor neurons" d_K connected by a "connective structure" composed of nerve fibres as shown in Figure 1.
The descriptor neurons represent particular objects, situations, diagnoses etc. characterized by sets of discrete features.
Each observation cell represents one of these features and is only excited if this feature is found in the data stream (such as a particular frequency in the case of incoming acoustic data).
The connective structure is such that fibres link each descriptor neuron to all observation cells characterizing it. Conversely, this means that each observation cell o_j is linked to all descriptor neurons which contain o_j as a feature.

After the "local" classification of incoming data by the excitation of discrete observation cells (structural feature s_1) this excitation is thus passed on via the nerve fibres (feature s_2) in such a way that each descriptor neuron, whose characteristic features are contained in the incoming data, receives some input. Maximum input obviously is received by the neuron representing the situation, object etc. generating the incoming data, and feature s_3 given above could be viewed as a clue to the fact that the "global" classification of the incoming data is indeed achieved by some comparison of the input intensity of the descriptor neurons.
This could, for example, be achieved by setting the threshhold controlling the "firing" of a neuron d_K so that it only fires if input has come from the full set of all the observation cells o_{kj} connected to it.

Given the functionality of the human nervous tissue, the local data classification, message transmission from observation cells to descriptor neurons and particularly the checking of the firing conditions of each neuron could be performed concurrently, thus leading to an extremely efficient classification process (feature f_1).

Allowing for secondary, lower thresholds permits the firing of neurons having received input from less than all the observation cells linked to them, thus generating associations, i.e. situations or objects sharing features with the primary data source, in an equally efficient manner (feature f_3). Moreover, lower threshholds obviously could also be used for approximate classifications based on some maximum input evaluation in the case of incomplete or locally erroneous data (feature f_4).

Inference processes could be realized by supplementing the set O of observation cells by additional cells d'_K which are not excited by incoming data, but by the firing of descriptor cells, thus providing additional input to the next cycle of mapping observations on descriptors.
The fact that such inference cycles have to proceed in series could be one reason for the relatively low performance in inference processing, as mentioned in f_2.

3. CONNECTION MATRICES IN EXPERT SYSTEMS

In the terminology of expert systems, where one distinguishes between a knowledge base and an inference engine, the main feature of the brain model described in the previous chapter and illustrated in Figure 1, is the fact that its knowledge base is mainly embodied in the connective structure between observations and descriptors, the appropriate knowledge representation being given by "connection matrices"

$$M_{kj} = \begin{cases} 1 & \text{if observation } j \text{ is connected to the descriptor } k \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

As implied by the model described in Chapter 2, data processing using this knowledge representation requires a prior local classification in which slots representing observations o_j of individual features or symptoms such as "temperature higher than nominal value" or "valve V_1 is open" etc. are instantiated by values o_j between 0 and 1 describing the degree to which the local classification holds (in many cases a two-valued classification : $o_j = 0$ or 1, however, is sufficient) providing an input i_{kj} (0 or o_j) to the kth descriptor via the connective structure M_{kj} :

$$i_{kj} = M_{kj} \cdot o_j \quad \text{for each } j. \quad (2)$$

These inputs are collected by accumulation functions F_k to provide an integrated input intensity :

$$I_k = F_k\ (i_{k_1}, \ldots, i_{k_n}), \quad (3)$$

a possible example of F_k being

$$F_k = \sum_j a_{kj}\ i_{kj}, \qquad a_{kj} = 1 \quad (4)$$

a_{kj} describing the "implication strength" of o_j with respect to d_k

The global classification is then performed by identifying the descriptors d_K for which $I_K = 1$ holds. If none can be found, as in the case of incomplete or erroneous information, this threshhold value $I_O = 1$ is reduced to yield approximate classifications as described in chapter 2.
(Details of the treatment of uncertainty on which this reasoning mechanism is based can be found e.g., in Ref. 1).

The ability to ask the user for missing data in case of incomplete information which is displayed by many expert systems, can be achieved by simply looking up the observations o_j connected to the approximate classifications d_K via M_{kj} and asking for them. This method can also be used to accelerate the classification process by first generating approximate classifications based on just a few randomly picked observations $o_j \neq 0$ and then automatically collecting the remaining evidence for just those classifications, a process which might be called "attention focussing on clues".
Knowledge processing can be further accelerated by parallel processing, dedicating a processor to each function F_k or at least to subsets of functions F_K.

Consecutive inferences are realized according to the method outlined in chapter 2.

4. ON-BOARD IMPLEMENTATIONS

The main functions of on-board AI systems will be failure diagnosis and recovery, MMI support (mainly by natural language understanding systems) and planning. AI systems for the first application fall into the category of typical expert systems, these being roughly characterized by their functionality as knowledge-based classification systems, as opposed to the somewhat

different functionality of natural language understanding and planning systems, although they also encompass classification processes.

Thus connection matrices can be used particularly for the first type of application but also for the other two as far as classification is involved.

Obviously knowledge bases using this type of knowledge representation require much less computer storage and knowledge processing time (even more so if parallel processing is employed) than in the case of systems based on representations which imply symbolic knowledge representation, symbol matching techniques, symbol handling overhead, special implementation languages and garbage collection problems.
For example, whereas the realization of a connection matrix embodying the knowledge to classify, say, 400 sensor readings into 20.000 different malfunctions roughly requires 1 Mbyte of computer storage which is still quite feasible for on-board implementation, the realization of a corresponding rule-based system with tens of thousands of rules would pose a formidable problem concerning the required memory and processing time.

Moreover, a rule based system of this size could also pose a formidable problem as far as the development, verification, validation and maintenance of the knowledge-base is concerned, whereas the simple structure of connection matrices and the underlying reasoning processes greatly enhance the transparency of the system, the development of the knowledge base simply being effected by an enumeration of observations and states and an identification of their interconnections. Methods to automize this process on the basis of FMECA (Failure Mode Effect and Criticality Analysis) and system simulations are presently being investigated.

5. CONCLUSIONS

On the basis of an assessment of some of the features of the human brain which seem to pertain to its extremely efficient utilization of very large knowlege sources a type of knowledge representation and corresponding inference mechanism for expert systems has been presented which is particularly suited for the processing of large knowledge bases at comparatively low storage and computer time requirements.

Whereas expert systems of this type are conceptually similar to systems involving the treatment of uncertainty, they are representationally different in that the effort in describing the connectivity between observations and descriptors has been reduced to a minimum by replacing symbolic descriptions by simple elements of connection matrices thus eliminating high storage and computer time demands typical of systems characterized by symbolic knowledge representation, symbol matching techniques, symbol management overhead, special implementation languages and garbage collection problems.

6. REFERENCES

1. Zimmermann, H.F.
Fuzzy Set Theory and Its Applications
Kluwer-Nighoff Publication, 1986.

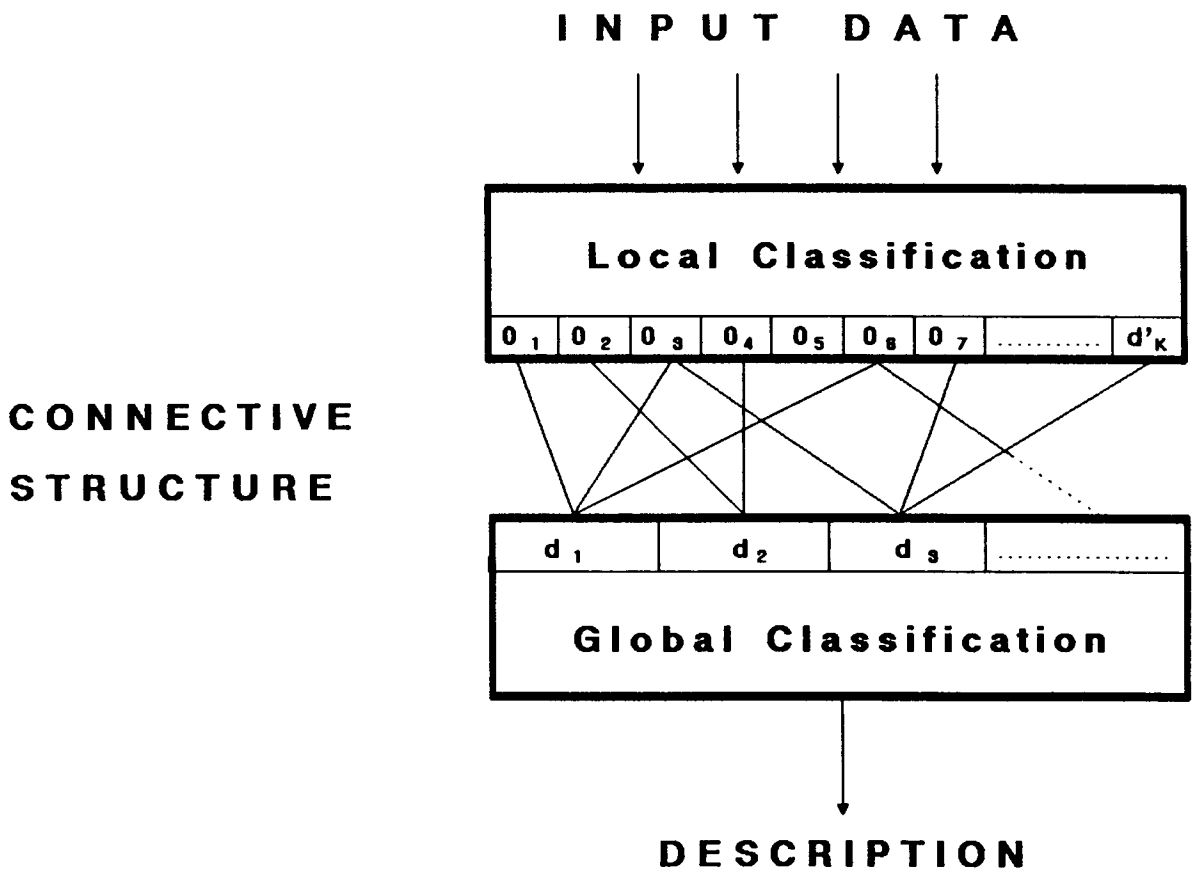


Figure 1 : Model for expert systems based on connection matrices

N88-16435

RB-ARD: A PROOF OF CONCEPT RULE-BASED ABORT

Richard Smith
John Marinuzzi

ABSTRACT

The Abort Region Determinator (ARD) is a console program in the space shuttle mission control center. During shuttle ascent the Flight Dynamics Officer (FDO) uses the ARD to determine the possible abort modes and make abort calls for the crew. The goal of the RB-ARD project was to test the concept of providing an onboard ARD for the shuttle or an automated ARD for the mission control center (MCC). This goal required a knowledge system to capture the expertise of the "man in the loop". A proof of concept rule-based system was developed on an LMI Lambda computer using PICON, a knowledge-based system shell. Knowledge, derived from documented flight rules and ARD operation procedures, was coded in PICON rules. These rules, in conjunction with modules of conventional code, enable the RB-ARD to carry out key parts of the ARD task. Current capabilities of the RB-ARD include: continuous updating of the available abort mode, recognition of a limited number of main engine faults and recommendation of safing actions. Two types of main engine faults are recognized, Pc shift and oxidizer turbine discharge temperature anomalies. Safing actions recommended by the RB-ARD concern the SSME limit shutdown system and powerdown of the SSME Ac buses.

PRECEDING PAGE BLANK NOT FILMED

N88-16436

A LEARNING APPRENTICE FOR SOFTWARE PARTS COMPOSITION

Bradley P. Allen
Peter L. Holtzman
Inference Corporation
5300 W. Century Blvd.
Los Angeles, CA 90045

ABSTRACT

We provide an overview of the knowledge acquisition component of the Bauhaus [1], a prototype CASE workstation for the development of domain-specific automatic programming systems (D-SAPS). D-SAPS use domain knowledge in the refinement of a description of a application program into a compilable implementation [2]. Our approach to the construction of D-SAPS is to automate the process of refining a description of a program, expressed in an object-oriented domain language, into a configuration of software parts that implement the behavior of the domain objects.

We view this process of software parts composition as a problem-solving task. By structuring a problem-solving task so that the types of knowledge required are made explicit, the acquisition of knowledge useful in performing the task can be made simpler, and the resulting knowledge base becomes easier to maintain [5]. The Bauhaus incorporates a problem-solving architecture based on the RIME [7] and SOAR [3] systems that provides such a structure. In this architecture, the task of refining an initial program description is represented as a goal. A goal determines a problem space and an initial state in that space. A problem space is a set of operators that are useful in the satisfaction of a given goal. An operator transforms a state in the problem space into a new state, or creates a subgoal, or recognizes when a given goal is satisfied. The goal of refining the initial program description is satisfied when the system has composed a set of software parts to form an implementation of the program.

Operators are applied by the system by iterating through three stages:

1. Propose the set of operators that can be applied to the current state;
2. Choose an operator from the set to apply to the current state; and
3. Apply the operator, generating the next current state.

User intervention in the choice of an operator is requested when the system reaches an impasse: when no operator applies, then the system is unable to express a preference for an operator, or when the system's preferences are inconsistent. The types of user intervention that can occur correspond to the types of knowledge needed by the system to avoid similar impasses in the future. The system generalizes from observed instances of user intervention to create new operators and preferences. In this manner, the programming knowledge of the system is automatically

 PRECEDING PAGE BLANK NOT FILMED

increased through its use as a software development tool by experienced application developers. This form of knowledge acquisition through the observation of user intervention in the design process allows us to characterize the Bauhaus as a learning system[4], similar to the VEXED VLSI design system [6]. Implementation of the Bauhaus is currently underway using ART running on a Symbolics Lisp machine under the Genera 7.1 environment, integrated with the Symbolics Ada programming environment.

References

1. Allen, B.P. and Holtzman, P.L. Simplifying the Construction of Domain-Specific Automatic Programming Systems: The NASA Automated Software Development Workstation Project. Proceedings of the Space Operations Automation and Robotics Conference, NASA/U.S. Air Force, August, 1987.

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

3. Laird, J.E., Newell, A. And Rosenbloom, P.S. "SOAR: A Architecture for General Intelligence". Artificial Intelligence 33, 1 (1987).

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

5. Soloway, E., Bachant, J. and Jensen, K. Assessing the Maintainablity 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.

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

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

N88-16437

AUTOMATIC PROGRAM GENERATION FROM SPECIFICATIONS USING PROLOG

Alex Pelin
Florida International University
School of Computer Science
Miami, FL 33199

Paul Morrow
AFWL / SCR
Kirkland AFB, NM 87117

ABSTRACT

We present an automatic program generator which creates Prolog programs from input/output specifications. The generator takes as input descriptions of the input and output data types, a set of tests, a set of transformations and the input/out relation. We use abstract data types as models for our data. The tests, the transformations and the input/out relation are also specified by equations.

The program generator creates a Prolog program which takes as input a data item of the input data type, item1, and outputs a data item, item2, of the output data type such that the input/out relation is satisfied by the items item1 and item2. In building the program the generator uses only the tests and the predicates given as input. The program generator was written in Prolog.

We implemented descriptions of the data types array, list, natural number and record. Our generator was able to generate correct Prolog programs for sorting lists with and without eliminating duplicate elements. We are currently working on developing heuristics for handling arrays and records.

We present a method for validating the data type descriptions based term rewriting systems and first order logic. Since the specifications of the data types can be quite involved we are building a natural language interface to our system. This way the user can enter commands in the English language and use the predefined data types. At the same time the sophisicated user can define his/her own data types.

We will also discuss the heuristics used by the automatic program generator in building Prolog programs. We will show the advantages and the disadvantages of writing automatic program generators in Prolog. We investigate problems associated with the use of term rewriting systems and of the theorem prover ITP in validating data types and in proving the generated programs correct.

We relate our work to the approaches used by Prywes, Manna and Dershowits in building automatic program generators.

N88-16438

ON ACQUISITION OF PROGRAMMING KNOWLEDGE

Ashok T. Amin
Computer Science Department
The University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT Acquisition and judicious incorporation of programming knowledge into the programming environment is essential to support development of correct programs and thereby enhance the programmer productivity. A program may be viewed as an outcome of interaction between a programmer and his/her programming environment. The interaction may be supported at the program generation level, program design level, or at the program specification level. The higher is the level of interaction supported the greater is the knowledge base required to support this interaction, and greater are the programming skills required of the programmer. Knowledge acquisition techniques used range from formal-based on mathematical properties of programs to empirical, and from generic to application domain specific. In this paper, we review the recent developments in this area and suggest future directions.

1. INTRODUCTION

Knowledge based programming environments have an important role to play in facilitating the development of correct programs. Such environments can provide for rapid development of correct programs by guiding the programmer through the maize of design decisions and implementation alternatives, and automating low level programming tasks. Developments in this area may lead ultimately to Automatic Programming Systems.

The goal of automatic programming is to automate all the phases of the program development - namely, program specification, design, and implementation. There are two approaches to realization of an Automatic Programming System. In one, the system proceeds with given correct specification of the program and through application of an appropriate sequence of valid transformations transforms the specification into a program. The program specification may be assumed to have been provided by the programmer or developed by the programmer through interactions with the system. As opposed to this, the other approach is more of an evolutionary approach, and is based on extending the automation of programming tasks from lower levels to higher levels. In either case, acquisition and codification of programming knowledge is an essential task. It has been observed [3] that large body of programming knowledge exists and that codification of this knowledge remains most limiting factor to ultimate development of automatic programming systems.

Knowledge acquisition and incorporation for an evolving

PRECEDING PAGE BLANK NOT FILMED

discipline is more problematic than for a mature one. One finds that programming is variously described as art, craft, and science. In the sense that parts of programming process are at various stages of evolution each of the terms may reasonably be used to describe the programming process. The evolving formalization of programming process by Computer Scientists and the the proven and empirical techniques of programming craft used by the practitioners must be used as the sources of programming knowledge and the knowledge from these source be suitable integrated to realize the gains in automation of programming process.

To further complicate the matter, not only the knowledge bases relevant to various phases of program development need to be integrated, but knowledge bases that address the development of correct programs must be integrated with one that addresses the efficiency concerns in a manner that changes in the one has minimal impact on the other [6]. Often specialization of a programming system to a specific application opens up opportunities for improving the programming process for that application by judicious incorporation of domain specific knowledge [1]. An important aspect of any system is to allow for acquisition and integration of additional knowledge and refinement of existing knowledge during its use by the programmer.

2. PROGRAMMING KNOWLEDGE

Fundamental research in Computer Science deals with formal approaches to understanding of programs and the programming process. The knowledge so gained may be used to develop exploratory techniques for program development. Those techniques that gain acceptance and wide usage then become proven techniques for program development. There are a number of issues that require more empirical approaches for its resolution. Such issues relate to reliability, maintainability, efficiency, human interfaces, etc.

There are two views in relation to approaches to automatic program development. One assumes that formal approaches will ultimately permit automatic development of correct programs. The other view accepts that writing correct programs is hard, and therefore it is accepted that programs may contain errors and that through iterative process of error detection and correction one arrives at a program in which one has high degree of confidence that it is correct. It may be noted that these two approaches complement each other in the sense that formal approaches are necessary if we are to build correct programs for critical applications, and at the same time empirical approaches are necessary if we are to develop any worthwhile program of reasonably large size in timely fashion.

Formal Approaches:

Formal approaches to the understanding of the programs and the programming process demands high degree of proffiCiency

in the creative application of mathematical skills. Formal approaches do not always yield results that may be used to develop practical techniques. Often techniques based on formal approaches are time consuming particularly when it is to be carried out manually. Further, since these techniques require specialized skills, they are subject to subtle and hence hard to detect errors. For example, it may be very difficult to detect error in an erroneous proof of program correctness. None the less knowledge gained from formal inquiries have yielded many useful results.

Laws of programming [5] and formal program design methods [4] are examples of formal approaches to the understanding of programs and the programming process, respectively. It is investigations of these types that will allow us to resolve the questions of program equivalence by suitable representation of programs in a canonical form. And allow development of programs through better understanding of issues that are important at various stages of program design and to the development of languages with suitable degrees of freedom to represent a program through various stages of development.

Exploratory approaches to program development are typically based on some break through in the formal understanding of programs and the programming process. Formal representation of program design information in Programmer's Apprentice represents an exploratory technique [7] for use in semi-automatic program development. Acceptance and wide usage of these techniques leads to proven techniques which then become common knowledge.

Common Knowledge:

Vast body of programming knowledge exists in the form of textbooks and reference books, especially related to algorithms, data structures, and structured programming. The impact of the developments in these areas from formal inquiries in 50's and 60's is now visible in development of programming languages and programmer training. While programming language can not enforce structured programming it can and have facilitated development of structured programs by providing suitable constructs and support for abstract data types. It is believed that advances in programming will require development of languages that support the major paradigms of their user communities [2].

The paradigms for algorithm development, such as divide-and-conquer, provide a systematic approach to development of algorithmic solution to a problem. A unified view of application of this paradigm for the development of algorithms and its implementations for a class of problems, say sorting, is needed to extricate the technique in details enough for the knowledge to be represented in machine processable form [3,8]. To this end, one can recognize that merge sort, quick sort, insertion sort, and selection sort are examples of divide-and-conquer paradigm with differences in the methods of partitioning of the problem and the composition of the solution from the partial solutions. In addition, a number of paradigms need to be developed for the tech-

niques of implementation to be suitably represented, such as recursion-to-iteration transformation, etc.

Empirical Approaches:

There are a number of issues of practical importance that defy formal approaches and in fact may not be properly be subjects of formal inquiry. For example, when do we stop testing a program? A practical answer is when the resources allocated for the testing have been exhausted. None the less, formal inquiries on this questions have yielded results that are awfully inadequate. Other issues involve human interfaces, or measures of complexity. It has been remarked that programming deals with managing complexity. There are rules, based on psychological studies, that say a human can deal with seven things at a time. A number of software methodologies use this as a guide to help manage complexity. How large a problem should be for a technique to more efficient than a simpler one? A number of implementation issues have this characteristics. In such cases it much better to arrive at a resolution based on interaction with the user. The studies involving observations of expert programmer at work also yield useful information involving 'good' programming practices [9].

Acquisition and Incorporation:

The approach used for knowledge acquisition and incorporation is important. Knowledge bases that addresses specific aspects of programming must be integrated in a manner that allows common interface without making it more complex to use or update. The knowledge must be represented so that its refinement and addition can be accommodated gracefully. Lastly, means must be provided so that user may add and modify the programming knowledge.

3. CONCLUSION

For the evolving discipline of programming, acquisition of programming knowledge is a difficult issue. Common knowledge results from the acceptance of proven techniques based on results of formal inquiries into the nature of the programming process. This is a rather slow process. In addition, the vast body of common knowledge needs to explicated to the low enough level of details for it to be represented in the machine processable form. It is felt that this currently impediment to the progress of automatic programming. Importance of formal approaches can not be overestimated since its contributions lead to quantum jump in the state of the art.

REFERENCES

[1] D. Barstow, "Domain-Specific Automatic Programming," IEEE-TSE, 11.11(1985)1321-1336.

[2] R. W. Floyd, "The Paradigms of Programming," Communications

of ACM, 22.8(1979) 455-460.

[3] C. Green and D. Barstow, "On Program Synthesis Knowledge," Artificial Intelligence, 10(1978) 241-279.

[4] C. A. R. Hoare, "An Overview of Some Formal Methods for Program Design," COMPUTER, 20.9(1987) 85-91.

[5] C. A. R. Hoare, I. J. Hayes, et.al, "Laws of Programming," CACM, 30.8 (1987 672-686.

[6] E. Kant, "Efficiency in Program Synthesis" UMI Research Press, 1981.

[7] C. Rich, "A Formal Representation For Plans In The Programmer's Apprentice," Proc. of Seventh International Conference on Artificial Intelligence, 1981.

[8] D. R. Smith, "Top-Down Synthesis of Divide-and-Conquer Algorithms," Artificial Intelligence, 27(1985) 43-96.

[9] E. Soloway and K. Ehrlich, "Empirical Studies of Programming Knowledge," IEEE Trans. of Software Engineering, 10.5(1984) 595-609.

N88-16439

APPLICATION OF ARTIFICIAL INTELLIGENCE TO IMPULSIVE ORBITAL TRANSFERS

by

Rowland E. Burns

NASA
Marshall Space Flight Center
Huntsville, AL

ABSTRACT

A generalized technique for the numerical solution of any given class of problems is presented. The technique requires the analytic (or numerical) solution of every applicable equation for all variables which appear in the problem. Conditional blocks are employed to rapidly expand the set of known variables from a minimum of input. The method is illustrated via the use of the Hohmann transfer problem from orbital mechanics.

INTRODUCTION

Although many papers deal with the use of rule based systems, few have applied that logic to strictly mathematical systems. Mathematics programs in artificial intelligence tend to provide the solution to a very specific problem such as evaluation of an integral.

The techniques presented here are for a very different class of problem. The user requires the numerical solution to an extended problem that could involve any number of equations from differing fields. While some of these equations are critical, others are inapplicable to the specific problem at hand; some of the critical equations are usable only after other equations are applied in a specific order. The non-expert cannot be expected to know all of these equations, the order in which they must be employed, or even whether or not there is enough information at hand to solve the assigned problem. This paper suggests a method to eliminate such difficulties. The method has been reported in one other paper (Ref. 1); that paper was discovered after the present effort was well under way.

THE PROGRAM LOGIC

Virtually every (numerical) computer program can be regarded as a map from mandatory inputs to invariable outputs. The goal of this program is to relax this one-to-one map and provide a complete set of outputs from any sufficient set of inputs. To begin the process, we initially establish the value of all constants and then set the values of all variables to nil.

The heart of the technique is to solve every potentially applicable equation for every variable which occurs in that equation. This should be done analytically, if possible, but we allow for the possibility that iteration, numerical integration, etc., may be required. If, for example, an equation such as

$$W=F1(X,Y,Z)$$

is applicable then we would also solve for

PRECEDING PAGE BLANK NOT FILMED

X=F2(W,Y,Z),

Y=F3(W,X,Z),

and

Z=F4(W,X,Y).

Then, for each of these equations we can write statements (LISP is convenient) such as

```
(COND ((AND X Y Z (NOT W))
   ...
(COND ((AND W Y Z (NOT X))
   ...
```

etc. The first condition would be met if and only if X and Y and Z are not nil while W is nil. The second would be met if and only if W and Y and Z are not nil while X is nil. Note that, in non-trivial systems, we may have several equations that yield W as a consequent from varying antecedents so that a specific condition statement for W may never be fulfilled. Once W is produced from any equation, it then becomes available to help produce, say, X or Y or Z from these equations. A rapidly increasing base of known variables results from this approach.

After a condition block has been satisfied the body of the block is filled with various tools that govern the operation of the program. Most important is the evaluation of the consequent which is then added to the accumulating knowledge base; this also guarantees that this block will never again be activated (nor will any other block with the same consequent). All physical units are "conditioned", if necessary, within the block (to allow the user to work in any units that are convenient). An array value is stored to record the order in which a given block was accessed. (This provides a "derivation" of the answer.) Another important function is to set an event-flag to indicate that a new variable has been evaluated. At the beginning of the condition block subroutine, the event-flag is set to nil and, if a new variable is added to the store of known values, the flag is set non-nil. At the end of the condition block subroutine, a non-nil event-flag forces subroutine looping until no new variables are evaluated or until the program terminates. If there are no new values and no termination, further input is requested. (Termination occurs when all variables have non-nil values.)

The mechanics of presenting the flow of information to the user can be handled in many ways. One technique which has proven to be valuable is to present the user with two dynamic menus. One of the menus lists the variables which are presently known (and their numerical values) while the other gives a mouse-sensitive list of variables that have yet to be specified. The "known" menu grows at a very rapid pace because one input often yields many new variable values. The variables which are known, and others derived from them, are removed from the "as yet unspecified" listing on that dynamic menu.

THE HOHMANN TRANSFER

Although page limitiations preclude non-trivial examples, the classic Hohmann transfer is illustrative. This maneuver involves a rocket vehicle leaving a circular orbit by adding an impulsive velocity along the tangent to the orbit, coasting on an ellipse to a higher circular orbit and entering into that orbit via a second tangential impulse.

For circular orbits, if we define the gravitational parameter as

μ, the distance from the attracting primary as R, the magnitude of the velocity vector as V, and if we use subscripts o and f to indicate the initial and final orbits then we have, from the two body problem (Ref. 2),

$$Vo = \sqrt{\mu/Ro} \tag{1}$$

$$Vf = \sqrt{\mu/Rf} \tag{2}$$

From these come immediately

$$Ro = \mu/Vo^2 \tag{3}$$

$$Rf = \mu/Vf^2 \tag{4}$$

The velocity which must be gained at the two end points of the transfer ellipse are given by

$$\Delta Vo = \sqrt{\mu(2/Ro-1/A)} - Vo \tag{5}$$

$$\Delta Vf = \sqrt{\mu(2/Rf-1/A)} - Vf \tag{6}$$

where "A" is the semi-major axis given by

$$A=(Ro+Rf)/2 \tag{7}$$

From (5), (6), and (7) come the relationships

$$Vo = \sqrt{\mu(2/Ro-1/A)} - \Delta Vo \tag{8}$$

$$Ro = 2ARo/[A(Vo+\Delta Vo) + \mu] \tag{9}$$

$$A = \mu Ro/[Ro(Vo+\Delta Vo) - 2\mu] \tag{10}$$

$$Vf = \sqrt{\mu(2/Rf-1/A)} - \Delta Vf \tag{11}$$

$$Rf = 2ARf/[A(Vf+\Delta Vf) + \mu] \tag{12}$$

$$A = \mu Rf/[Rf(Vf+\Delta Vf) - 2\mu] \tag{13}$$

The "total" velocity increase, V , can be written as the simple sum

$$\Delta V = \Delta Vo + \Delta Vf \tag{14}$$

which rearranges to give

$$\Delta Vo = \Delta V - \Delta Vf \quad (15)$$

$$\Delta Vf = \Delta V - \Delta Vo \quad (16)$$

If we now introduce M as mass, then the final mass, Mf, is related to the initial mass, Mo , ΔV, and the rocket exhaust velocity, C, as

$$Mf = Mo * \exp(-\Delta V/C) \quad (17)$$

which yields the subsidiary relationships

$$Mo = Mf * \exp(\Delta V/C) \quad (18)$$

$$\Delta V = C * \log(Mo/Mf) \quad (19)$$

$$C = \Delta V/\log(Mo/Mf) \quad (20)$$

Ro, Rf, Vo, Vf, A, Mo, Mf, ΔV, ΔVo, ΔVf and C are thus related by a total of twenty equations. Vo, for example, can come from either equation (1) or equation (8) or from user input. A may be generated from equation (7), equation (10), equation (11) or from input. The only source for Mo is from equation (15) or via input. Etc.

The user may now specify input variables in any order which is convenient. The solution for all variables in all forms from the applicable equations allows a free form input wherein any set of inputs will necessary lead to the output of all unspecified variables.

It is also possible to aid the user by providing help in determining which variables must still be specified to obtain a desired result. By tracing the dependence of the output variable on input variables the user may well recognize a sub-set of inputs that can lead to the required solution. This could allow a solution for smaller problems without having to solve for all possible answers. Graphics aids also have obvious applications.

It must be realized that the above set of equations is so extremely limited that it is doubtful that the corresponding program would hold any users interest for more than a very brief time. To expand to a full design tool requires including options for elliptical initial and target orbits, full three dimensional capability, equations which relate the rocket mass to payload mass, propellant mass, and structural mass, phasing infomation (Gauss' equation requires iteration to isolate the eccentric anomoly), etc. These extensions are presently under way at Marshall Space Flight Center and a useful program is expected within about six months.

CONCLUSIONS

The foregoing has provided only a minimal outline of an important concept in the field of artificial intelligence. Even so, the potential uses of such a system are enormous.

It is apparent that the technique is not limited to mathematics because, for example, it is possible to replace the condition blocks which deal with equations with, for example, chemical synthesis procedures. Once the elementary reactions are available it is possible to begin synthesizing more and more exotic compounds using those

building blocks. The system would indicate how to construct extremely complex organic compounds beginning with the elements; help screens could be productively employed to describe the exact laboratory procedures to be followed as well as required equipment.

In another area, it is also possible to "computerize" entire text books and produce programs that would be sold to students as a pony for a given text. With such help it would be impossible to assign a problem to the student which could not be solved using the equations that are supposed to be at hand. The next step would be to combine several texts in differing fields in order to cross-reference knowledge. One would expect that equations and concepts which occur in different fields could be used in unexpected and surprising ways; the end of such an endeavor is not predicable.

REFERENCES

1. Elias, Antonia L. "Knowledge Engineering of the Aircraft Design Process", Chapter 6 of "Knowledge Based Problem Solving", Januez S. Kowalik, Ed., Prentice-Hall, 1987.

2. Burns, Rowland E. "Ascent from the Lunar Surface", NASA TN D-1644, August, 1965. Appendix B.

N88-16440

A FRAMEWORK FOR REAL-TIME DISTRIBUTED EXPERT SYSTEMS: ON-ORBIT SPACECRAFT FAULT DIAGNOSIS, MONITORING AND CONTROL

Richard L. Mullikin

ARINC Research Corporation
11770 Warner Avenue, Suite 210
Fountain Valley, CA 92708

ABSTRACT

Control of on-orbit operation of spacecraft requires retention and application of special purpose, often unique knowledge of equipment and procedures. This information must be available for application in controlling routine as well as critical operating conditions. Expert or knowledge-based systems technology offers the capability of retaining this valuable knowledge under a variety of conditions. Even current expert systems (ES) have limitations however. A real-time, distributed intelligence system is a highly complex, human-machine system where an artificially intelligent component operates the system and the human manages it. Real-time distributed expert systems (RTDES) permit a modular approach to a complex application such as on-orbit spacecraft support. One aspect of a human-machine system that lends itself to the application of RTDES is the function of satellite/mission controllers; the next logical step toward creation of truly autonomous spacecraft systems. This system and application is described in this paper.

INTRODUCTION

Numerous complex and sophisticated human-machine systems exist which are beyond the scope and capability of current applications of artificial intelligence (AI) technology, yet these systems could significantly benefit from the application of some form of machine intelligence. This potential forms the basis for the application of AI and ES technology to real-time problems and distributed problem solving in development of RTDES. No one of the domains (real-time applications, distributed AI and expert systems) are sufficient for future autonomous systems as independent entities. One approach which embraces these domains in system development mimics the real-time problem solving ability of a human operator, using real-time paradigms in a heuristic structure.

Distributed AI is the technique by which several physically or logically separated AI programs cooperate to achieve a system-wide goal [4]. Distributing the decision-making AI programs in a real-time application has the potential to achieve the following benefits:

1. Allow autonomous operation of field units
2. Greatly enhance system fault tolerance
3. Allow modular increases in system capability

PRECEDING PAGE BLANK NOT FILMED

4. Significantly decrease communications required between field units and a central decision maker.

The application of a distributed intelligence system provides a transition from totally manual to totally automated systems; a hybrid system of sorts.

DIFFERENT METHODOLOGY

The methodology and structure used to create autonomously operating individual AI programs in a nondistributed environment differs from the methodology used to produce AI programs that have to interface with one another. A decision-making program operating in a nondistributed environment is usually structured to achieve a specific goal or set of goals using a limited number of plans to reach these goals. These plans operate in response to input events and actions occurring in the environment. AI programs in a distributed environment dynamically modify their goals and plans in response to decisions made by other AI programs in the system [7].

In a distributed AI system, decisions made by each AI program (which could be considered a "module") depend on decisions made and actions taken by the other modules in the system. Both concurrent and non-concurrent events affect other events. Each AI program cannot have a knowledge base that provides for all combinations of occurrences that might happen system-wide. Each individual intelligent module must have the capability to dynamically alter its course of action based upon inputs from other modules. Each intelligent module must also have some degree of knowledge regarding the types of objects and events (i.e., new information) by the other modules. A critical issue yet to be solved for distributed AI is coordination of distributed AI modules.

Perhaps the most important feature of having AI modules reside within the individual entities (or sites) is the ability to intelligently determine what information should be transmitted to a central control center. If site intelligence were not available, entire streams of data would be transmitted each time new information was sensed at the individual sites. This selective information is shown as compressed data in Figure 1. The local intelligent sites are "intelligent" only in terms of local decision-making, and do not possess any mission or system strategies. Figure 2 depicts a distributed AI adaptation of the centralized control scenario. This distributed architecture places the greatest emphasis upon mission performance.

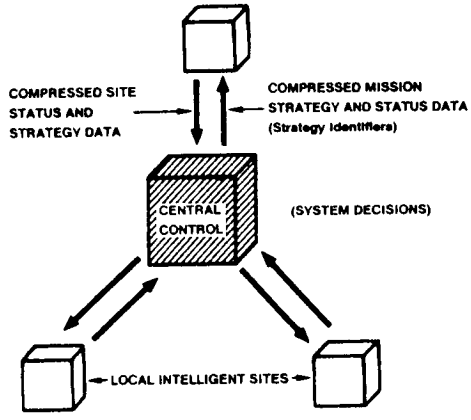


Figure 1. Centralized Control Structure

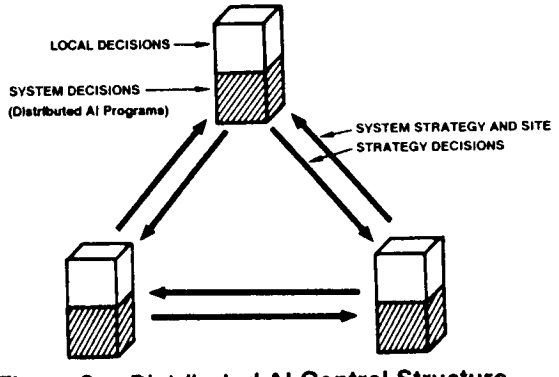


Figure 2. Distributed AI Control Structure

It is clear that diagnostic problems tend to make good applications of ES techniques [5]. This is because many diagnostic problems that are fully understood cannot be written algorithmically. For example, if the analysis is limited only to single component failures (as in even the simplest spacecraft system), there are still too many permutations for all cases to be deciphered explicitly. This combinatorial explosion is avoided in expert systems since the search tree is built at execution time and explores only the most promising branches.

SPACECRAFT OPERATION AND CONTROL

While an expert system can be used to consolidate a number of functions that analyze and monitor data during spacecraft passes or normal on-orbit operation, inherent limitations still remain. These limitations form the basis for the application of different structures and methodologies.

On-orbit spacecraft operation, including fault diagnosis, requires special purpose knowledge. This body of knowledge is acquired over decades, and is one of the most valuable resources of any large-scale program [6]. Unfortunately, this knowledge is most often lost with human attrition. ES technology offers the capability of retaining this knowledge, and in addition performing critical tasks faster and more reliably than humans, such as battery reconditioning, launch operations and resolution of anomalies.

The facilities and methods established for satellite command and control are quite basic and reliable. Raw data (telemetry) acquired via the assigned or scheduled tracking station is transmitted to a control center for processing. The "housekeeping" data is processed within a multisatellite data processing (DP) hardware and software complex for ultimate display to a spacecraft controller console. A variety of interfaces are available (e.g., CRTs, computer graphics, strip chart recorders, etc.). The "payload" data is processed specifically according to mission user requirements and disseminated appropriately. Processing of this data is usually affected off-line and incorporates the necessary corrections due to attitude and orbital influences [1, 3]. Alarm conditions triggered from digital and analog parameters exceeding pre-defined limits usually invoke a manual response.

A human-machine system such as satellite operations is an outstanding candidate for an RTDES application, based upon the telemetry, tracking and command functions described above. The advantages of this application include autonomous site operation, a high degree of fault tolerance, the ability to increase system capability modularly, and a decrease in real-time communications (due in part to data compression) between a central unit and local sites. The incorporation of one or more AI modules into such a system could enhance operational success because the controller would be freed from his/her time-sharing responsibilities, and thus have more time available to deal with a variety of more difficult, complex problems which might develop. This implementation process is evolutionary and iterative in nature, and could be a step towards the eventual elimination of a satellite control center.

It has been postulated that the definition of real-time processing is application dependent [2, 4]. Real-time in a satellite control application

might be considered to be a data processing response on the order of hundreds of milliseconds. It is this sort of high-level real-time application that must be addressed.

PERFORMANCE EVALUATION

The overall effect of incorporating one or more AI modules into a complex system can enhance system performance. One effect which can cause this result is the application of AI modules which makes the human's job easier.

Design of RTDES by incorporating artificially intelligent components into portions of a human-machine system does not guarantee enhanced system performance with the same or less input. The humans who remain involved with the resulting RTDES must be adaptable to work with the new creation. There are also a number of other compatibility considerations which must be addressed in the design phase, all beyond the scope of this paper.

Another consideration is that of cost effectiveness. It may prove difficult or impossible to isolate which subtasks can be supported by AI modules. This seems likely since it can be difficult to extract valid data on controllers regard for each subtask as part of the overall task. This difficulty arises because some subtasks are not independent but highly interrelated with other subtasks, and cannot be analyzed separately [7].

Overall communications requirements in RTDES are greater because system strategy decisions made by each site or unit must be relayed and coordinated with all other units.

CONCLUSION

Although significant progress has been made in artificial intelligence and expert systems technology, workable methods have not yet been developed for sophisticated real-time distributed AI-based systems.

The use of artificially intelligent local modules in a complex, distributed, real-time system offer the potential for significantly reducing the communications required to and from a centralized decision-maker, as shown in Figure 2. These modules can also provide a high degree of fault tolerance [2, 4, 7]. The associated benefits as a result of this application were outlined in the introduction and mentioned throughout.

The AI technology required to perform the communications, coordination, and control between the distributed AI modules is in the early research stage and many researchers in distributed AI are not addressing the real-time nature of systems [4] such as presented in the satellite control problem. Technology transfer from various AI disciplines will be required as a base for developing methods to implement real-time distributed AI. There is a need for these applications in a variety of disciplines.

The design of RTDES and similar types of systems are in the developmental stage. A number of relevent research issues exist which should be addressed in the immediate future:

1. Now that AI and expert systems are maturing, primary emphasis should be placed upon real-time implementations.

2. Further investigation is needed into the problem of communication and coordination among distributed AI programs.

3. Similarly, investigation into the criteria used to intelligently determine what information should be transmitted to other AI components needs to occur.

4. Research is necessary into the isolation of subtasks to be supported by AI components and subsequent interrelationships.

ACKNOWLEDGMENTS

The author wishes to acknowledge that the concept and much of the research for this paper was performed while working for the Jet Propulsion Laboratory in Pasadena, CA, while involved in the development and integration of the Space Flight Operations Center project.

REFERENCES

[1] Barret, M., "Impact of Spacecraft Design on Remote Control of Satellite Operations", IFAC Automatic Control in Space, Noordwijkerhout, The Netherlands, 1982, pp. 563-7.

[2] Hawkinson, Lowell B., Levin, Michael E., Knickerbocker, Carl G., and Moore, Robert L., "A Paradigm for Real Time Inference", 1st Annual Artificial Intelligence and Advanced Technology Conference, Long Beach, California, 1985, pp. 51-56.

[3] Kornell, Jim, "A Satellite with a Good Attitude: An Expert System for Stationkeeping", Third IEEE Conference on Artificial Intelligence Applications, Denver, CO, December 1986, pp. 111-117.

[4] McDaniel, Bonnie G., "Interface Requirements For Real-Time Distributed Artificial Intelligence", Proceedings, IEEE International Conference on Data Engineering, Los Angeles, CA, April 1984, pp. 241-245.

[5] Mullikin, Richard L., "The Commercialization of Artificial Intelligence: Identification, Assessment and Implication of an Integrated Decision Support System and Expert System", Master Thesis, Loyola Marymount University, Los Angeles, California, 1986.

[6] Scarl, E.A., and Delaune, C.I., "LOX Expert System", Proceedings, 21st Space Congress, Cocoa Beach, Florida, April 1984, p. 2-16.

[7] Smith, Leighton L., "The Distributed Intelligence System and Aircraft Pilotage", Artificial Intelligence and Simulation, Simulation Councils, Inc., San Diego, CA, 1985, pp. 26-28.

N88-16441

TES - A MODULAR SYSTEMS APPROACH TO EXPERT SYSTEM DEVELOPMENT FOR REAL TIME SPACE APPLICATION

Brenda England
Ralph Cacace
Hamilton Standard
MS 1A-2-5
Bradley Field Road
Windsor Locks, CT 06096

ABSTRACT

A major goal of the Space Station Era is to reduce reliance on support from ground based experts. The development of software programs using Expert Systems technology is one means of reaching this goal without requiring crew members to become intimately familiar with the many complex spacecraft subsystems. Development of an Expert Systems program requires a validation of the software with actual flight hardware. By combining accurate hardware and software modelling techniques with a modular systems approach to Expert Systems development, the validation and the software program can be successfully completed with minimum risk and effort. The TIMES Expert System (TES) is an application that monitors and evaluates real time data to perform fault detection and fault isolation as it would otherwise be carried out by a knowledgeable designer. This paper disucsses the development process and primary features of the TES, the modular systems approach, and lessons learned.

PRECEDING PAGE BLANK NOT FILMED

N88-16442

PROTOTYPE SPACE STATION AUTOMATION SYSTEM DELIVERED AND DEMONSTRATED AT NASA

DR. ROGER F. BLOCK
HONEYWELL SPACE AND STRATEGIC AVIONICS DIVISION
13350 US HWY 19 S.
CLEARWATER, FL 34624-7290

Abstract

The Automated Subsystem Control for Life Support System (ASCLSS) program has successfully developed and demonstrated a generic approach to the automation and control of Space Station subsystems. The automation approach was recently delivered and demonstrated by Honeywell and Life Systems Inc. for NASA JSC. The hierarchical and distributed real-time controls system places the required controls authority at every level of the automation system architecture. As a demonstration of the automation technique, the ASCLSS system automated the Air Revitalization Group (ARG) of the Space Station regenerative Environmental Control and Life Support System (ECLSS) using real-time, high fidelity simulators of the ARG processes. This automation system represents an early flight prototype and an important test bed for evaluating Space Station controls technology including future application of ADA software in real-time control and the development and demonstration of embedded artificial intelligence and expert systems (AI/ES) in distributed automation and controls systems.

Introduction

Early in 1983, NASA initiated a technology development program entitled, Automated Subsystem Control for Life Support System (ASCLSS, contract NAS-9-16895). The national commitment for a permanently manned Space Station was growing and NASA recognized that to be a successful, operational Space Station would require highly reliable and fully automated subsystems. NASA OAST (code R) and the Crew and Thermal Systems Division of NASA JSC set out to address this vital first step for the Space Station.

Following a competitive procurement, Honeywell and Life Systems, Inc. initiated a four year technology program to define, develop, and demonstrate a generic or common approach to automation and control for all Space Station subsystems including ECLSS, thermal (TPS), power (EPS), and guidance, navigation, and control (GN&C). Because of the critical importance of life support, the generic automation technique was to be demonstrated against the air revitalization group (ARG) of the regenerative ECLSS subsystem using high fidelity real-time simulators of the ARG O_2 generation, CO_2 removal, and CO_2 reduction processes.

ASCLSS Demonstration System

An early ASCLSS applications study indicated that a hierarchical architecture of distributed controllers would fulfill the automation and control requirements across a majority of Space Station subsystems. Additional automation system drivers included maximum operational autonomy from the ground, automation of routine subsystem operations and redundancy management, strong emphasis on commonality of hardware and software, accommodation for on-orbit maintenance,

PRECEDING PAGE BLANK NOT FILMED

use of accepted standards, and ease of subsystem performance growth and insertion of new technology.

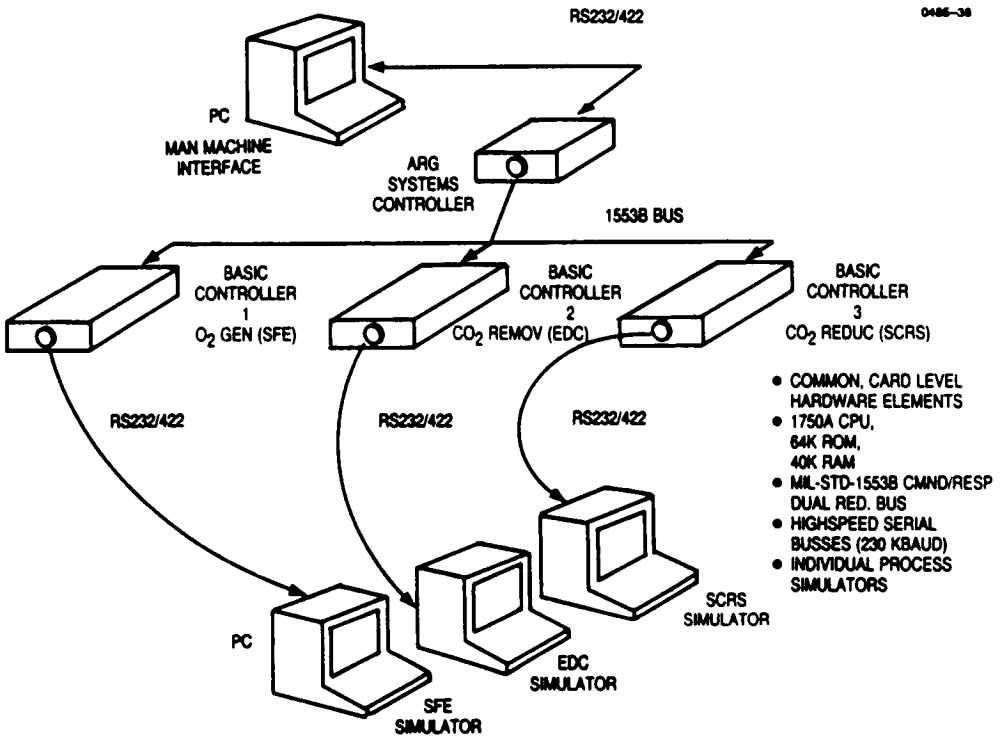


FIGURE 1. ASCLSS Automation and Control Demonstration System

The resulting ASCLSS automation system shown in figure 1 consisted of a two level hierarchy of distributed controllers implemented with MIL-STD-1750A microprocessors tied together by a high speed MIL-STD-1553B bus network. All four controllers at the basic or local level and system level are implemented in completely common hardware which supports maintainability and lower life cycle cost. The ARG processes are implemented by real-time simulations in individual personal computers. An important man-machine interface (MMI) is included in the automation system to provide the required supervisory control authority for the Space Station crew. Through the MMI, which represented the multi-purpose application console (MPAC), the crew can monitor system status and performance, and when necessary, exert override control authority or conduct system fault evaluation and maintenance procedures. The MMI also provided an effective demonstration of the control authority allocated between the crew, the system level controller and the basic or process level controllers.

The ASCLSS automation system implemented an innovative layered software architecture written in Pascal which emphasized significant levels of common software modules. Figure 2 illustrates the layered software structure developed for the automation and controls system. The real-time operating system (OS) is fully common to all controllers and is written in 1750A assembly. The OS performs all network communications and I/O data transfers and schedules/deschedules all controller tasks. The ASCLSS system control software acts as the agent for the application software between the I/O data base and the generic operating system. The system control software is the same in all local controllers and largely common in the system level controller. The only unique software in each controller is the application software and its associated table driven I/O data base. Thus, the non application-dependent software in each controller in the ASCLSS automation architecture is common.

ORIGINAL PAGE IS
OE POOR QUALITY

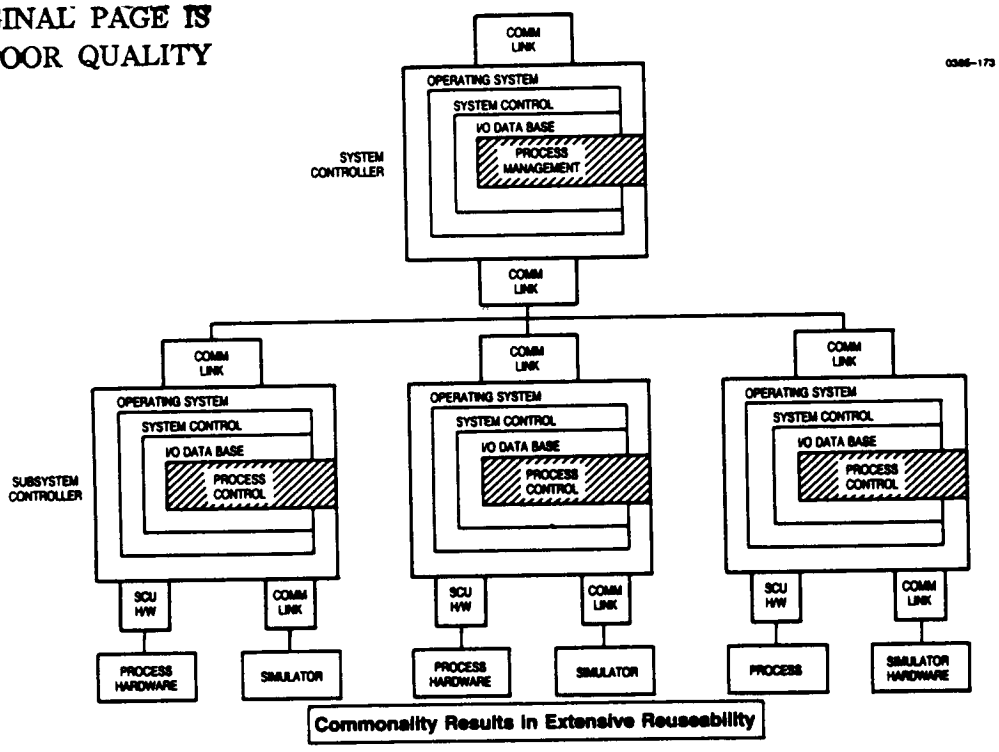


FIGURE 2. Layered Software Architecture Establishes Software Commonality

Many other important benefits are achieved by this layered software approach:

- The application software is completely independent of the automation system target hardware and operating system. This fosters software reusability and portability to any target microprocessor based system and accommodates software testing independent of the automation and controls system development.

- The ASCLSS software architecture was designed to have the application software developed by the subsystem or process expert. Thus, process control responsibility and accountability are retained by the subsystem or process developer, an important Space Station requirement.

- The subsystem controls authority is distributed across the automation hierarchy with maximum controls authority pushed to the lowest or local level which enhances system and process operational autonomy.

The use of real-time simulators of the ARG processes represent an early "proof-of-concept" of TAVERNS (Test And Verification of Remotely Networked Systems) being planned for Space Station. The simulators have the capability to run in real-time and at ten times real-time. They also can program in failures, abnormal performance conditions, and unique operational scenarios with no risk to the process hardware. They fully test the automation and controls system logic before the process hardware is integrated.

In summary, the delivered ASCLSS system demonstrated the automation of 1)three complex life support processes; 2)the monitoring and reporting of ARG system and process status, warning, and alarms; 3)the system event logging; 4)the fault detection and system safing; and 5)the

calculation and evaluation of the current system operational performance parameters and efficiencies.

Distributed/Embedded AI/ES

Space Station's ultimate success will depend heavily on AI/ES. As the Space Station evolves beyond IOC, application of AI/ES will be essential to support:

- Crew operations and scheduling activities.
- Enhanced real-time control procedures and subsystem performance management.
- Fault prediction, detection, isolation, system reconfiguration, and recovery.
- On-orbit maintenance and repair.

The capture of design knowledge and operations history will be required to reduce on-orbit and ground personnel training. The AI/ES will monitor Space Station subsystems, conduct performance trend analyses, and notify the crew of degrading or unsafe conditions well in advance of catastrophic failure. Maintenance will be scheduled when convenient and productive for the crew operations and/or compatible with the logistics schedule.

To meet the cost constraints of Space Station IOC, expert systems and limited levels of artificial intelligence can be embedded in the existing DMS architecture of conventional controllers using conventional software languages such as ADA. These initial AI/ES functions would be used as "controls advisors" and historic data base generators.

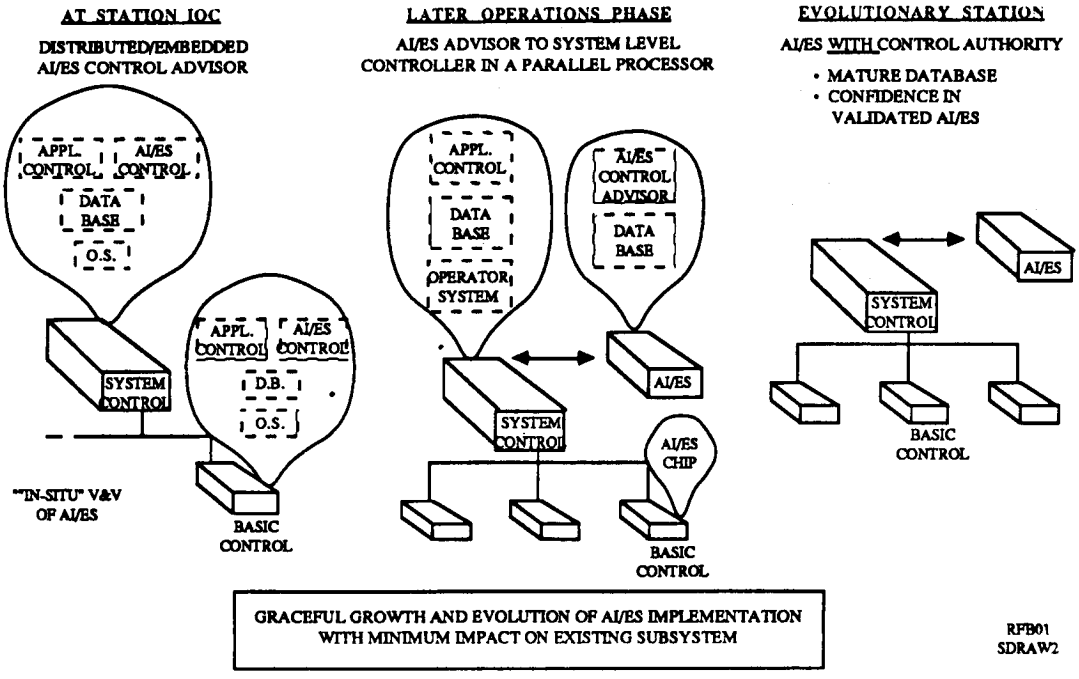


FIGURE 3. Evolutionary AI/ES Implementation on Space Station

ORIGINAL PAGE IS
OE POOR QUALITY

The AI/ES advisory functions would be tested and verified by using TAVERNS simulators. Figure 3 describes an approach to implementing AI/ES on the Space Station at IOC and gracefully evolving the level of AI/ES sophistication, control authority, and technology of implementation. As confidence is developed in the AI/ES controls advisor which is being validated "in-situ" in the Space Station operational environment, the AI/ES would be given ever increasing levels of control and command authority. As the level of functional sophistication grows, the AI/ES would eventually be hosted in parallel special purpose processors or AI chips.

The existing ASCLSS automation system offers a unique opportunity and test bed to demonstrate these principles of distributed, embedded AI/ES in a conventional real-time controls system architecture using conventional languages such as ADA. The real-time ASCLSS controllers have 40% of their CPU throughput in each 100ms minor cycle and 80% of their RAM and ROM memory available for embedded AI/ES "controls advisor" functions such as comparison of process state models, monitoring performance trend data, fault prediction, diagnosis, system reconfiguration, and recovery. The table driven, layered software is uniquely structured to accommodate the AI/ES functions at each controller in the hierarchy as merely another task in the application software program. The distributed, embedded AI/ES provides a high level of autonomy and a very responsive advisory function to the real-time process controllers.

Conclusions

The ASCLSS automation system has extended the proven approaches to industrial and commercial automation and control into the unique environment and high performance requirements of the future permanently manned Space Station. The recently delivered ASCLSS has successfully pioneered many of the important features planned for Space Station including commonality of hardware and software, implementation of standards, incorporation of high levels of operational autonomy, and the placement of the crew operator into supervisory control. The system also represents an early Space Station test bed which will support evaluation of subsystem controls requirements, application of ADA software to real-time control, and development of embedded AI/ES in a distributed automation and controls system.

Ten years from now, as astronauts and civilians are living and working on the Space Station, the Earth bound world will undoubtedly take this significant achievement for granted. The automation principles demonstrated by the ASCLSS system represent an early benchmark and significant first step to this successful operational Space Station in the mid-1990's.

PRECEDING PAGE BLANK NOT FILMED

N88-16443

APPLICATIONS OF EXPERT SYSTEMS FOR SATELLITE AUTONOMY

A. Ciarlo
ESA-ESTEC (WDI)
P.O. Box 299
2200AG Noordwijk
The Netherlands

P. Donzelli
LABEN SpA
SS Padana Superiore 290
20090 Vimodrone (Milano)
Italy

ABSTRACT

This paper discusses some aspects of the on-board application of Expert Systems (ES's) in artificial satellites. The ideas presented are mainly based on the experience gained during a study performed by LABEN, CRI, DORNIER and CRISA under European Space Agency (ESA) contract; the activities of the study, which include the implementation of two prototypes on a dedicated AI machine, are described. The more general implications of the experience are then discussed. These concern firstly, the interrelationship between the ES and the architecture of the satellite and its impact on the mission definition phase of the satellite lifecycle. Secondly, the main obstacles that need to be overcome before operational use of ES's on-board can take place, and namely the matters of testing, knowledge collection, and availability of computing resources. Finally, the activities that appear to be required in the near future to prepare the way for the full exploitation of this technology for satellite autonomy are briefly outlined, together with a brief description of an ongoing work studying the application of AI techniques for the management of the Cassini Titan probe; the probe will not have a telecomand link and will therefore have to manage autonomously its descent on Titan.

INTRODUCTION

Lately increased satellite autonomy has become a more pressing necessity; the reasons are various, but all have their roots in the fact that current technology allows very complicated missions to be implemented, the control of which, with the methods developed for satellites of previous generations, is extrememly costly. The recent availability of powerful development environments for Expert Systems (ES) has simplified the implementation of these to the point of making them viable tools for the achievement of increased spacecraft autonomy, and some activities have been initiated by the ESA to investigate the possibilities of applications of ES's to its own purposes.

ESA SATELLITE ARCHITECTURE

Over the last twenty odd years an architecture and a relatively fixed break-down into major functional subsystems have become generally accepted. The subsystems usually include Structure, Thermal, Power, Data Handling, Attitude and Orbit control, Telemetry/Telecommand, Antennas, and Payload(s). This break-down in functional subsystems represents the best compromise among often contrasting requirements, of which a significant one is the need to comply with geographical distribution i.e. assignment of tasks to companies in all participating member-states

In this architecture the Data Handling (OBDH) represents the core of the whole spacecraft from the point of view of control and monitoring; it performs

all the functions required to decode and distribute telecommands, gather and format spacecraft data for telemetry, and to provide a general purpose on-board data processing facility. The design of the OBDH is based on a distributed architecture of (intelligent) units connected by a serial bus; specialised units provide interfaces to satellite subsystems and payloads, while a Central Unit controls bus data traffic and general subsystem operation. The fact that this subsystem has access to all satellite data and that it includes a number of computing resources makes it the "natural" host of an on-board ES for autonomy.

STUDY OF EXPERT SYSTEM FOR SPACECRAFT MANAGEMENT

One of the studies initiated by ESA had as main purposes to verify the feasibility of an on-board ES for the management of an autonomous spacecraft, to identify the general requirements of such an ES, to assess the resulting on board complexity, and finally to identify areas for future research in the field. The study was carried out by a consortium of four companies: LABEN (Italy), CRI (Denmark), Dornier (W. Germany) and CRISA (Spain).

The first task was to identify a satellite which could be used as reference for the study. It was as decided to define a "hypothetical" satellite, rather than rely on a real design, because it was thought that this would allow it to be tailored to fit the purposes of the study; this in fact resembles the "toy problem" approach. The definition of the reference satellite served its purpose, because it provided a stable, logically well defined test case of reasonable complexity, but the whole procedure introduced some difficulties that will be explained later. The architecture of the reference satellite was based on the functional breakdown usual for ESA satellites, and the ES was supposed hosted in the OBDH.

Although the eventual goal of the study was to increase satellite autonomy by the exploitation of ES technology in an a priori unrestricted fashion, it was also clear it would be necessary to identify a limited application domain. The second task of the study was therefore a review of the possibilities within the general areas of mission, health and failure management; of these, the mission and health management were discarded because these tasks consist essentially of resource and activity scheduling, and therefore require reasoning about time. The remaining one, fault management, was chosen also because diagnosis is one of the historically successful applications of ES's. For what regards the object of the fault management activity, the Power and Data Handling subsystems were selected because the complexity of their design and operation matched well with the goals of the study, the relevant expertise is well established within the Consortium, and finally because their continuous operation is vital for the safety of the spacecraft.

The conventional method of knowledge acquisition through face to face consultations between the domain experts and the ES builder was difficult due to the geographical distribution of the personnel involved. Consequently, after an initial study of the problem area, a "knowledge specification formalism" was produced. This was then submitted to the domain experts, who wrote down the problem solving knowledge, producing "paper knowledge bases". These, together with the actual descriptions of the target systems (OBDH and Power) were the input to the coding of the ES. There was good correspondence between the knowledge specification formalism and the ES architecture.

Consequently, the actual coding of the ES's using the "paper knowledge bases" was fairly easy. However, it turned out that the problem solving strategy used by the ES deviated in many cases from that of the experts. These deficiencies had their roots in an insufficient problem analysis at the outset of the study, or at least in the failure to identify the implicit assumptions about diagnosis strategy within the specification formalism and to discuss them with the domain experts. The problem solving techniques should have been identified better before the knowledge specification formalism was constructed. Although the prototype systems seldom arrived at erroneous diagnoses, the failure in capturing correctly the experts' strategic problem-solving knowledge is serious. Obviously the solution adopted was too optimistic, and more code-evaluate-update cycles are required even before the formalism is fixed; afterwards it can be used by the experts to supply inputs to the knowledge base in a reasonably simple yet consistent form.

The architecture of the ES was based on a representation which models the fault propagation by means of a causal associational network. Such a network allows certain sets of decision strategies to be implemented retaining an explicit representation of the diagnostic knowledge. This knowledge is structured in three different layers: observation layer (symptoms and test), causal layer (failure states) and diagnostic layer (diagnosed states). The observation layer contains all the information that can be obtained by the subsystem organised in symptoms and tests: the first ones are used to activate the diagnostic process, the second ones are used by the inference engine, during the diagnostic process, to discriminate among failure hypotheses. The causal network and diagnostic layers are represented by a directed acyclic graph of nodes where each node identifies a state representing deviations from the normal behaviour. Three types of nodes are used:

- Failure states: nodes which conjecture the occurrence of a failure in a certain system component
- Diagnosed states: end nodes of the network containing the identification of the failed system component
- Starting nodes: pointed at by the associational arcs as result of the identification of a known symptom.

The nodes are connected by means of two types of arcs:

- Associational arcs that link the symptoms with starting nodes identifying the weight of the association;
- Causal arcs that link couple of states identifying the weight of the causality.

The development environment used in the study was the Intellicorp's KEE running on on a Texas Instruments Explorer workstation. The failure states (diagnosis hypotheses) were implemented as units (frames), and the determination of the causal pathways was effected through inferences over rule-sets. Tests and actions were associated with the units representing failure states as methods. The debugging facilities provided by the system enhanced the "visibility" of the diagnosis process considerably, especially compared to what would have been possible with a "conventional" software development environment.

Two prototype expert systems were constructed, one for the Power S/S, and one for the OBDH S/S. The user interfaces are similar in design for the two systems.The graphical facilities of the development environment are used to display information about knowledge bases, subsystem, and diagnosis process.

The user's communication with the prototypes is based on menu selection. He may specify the symptoms of a failure among those that are considered likely by the experts; when tne setting of the symptoms chosen for the test case is completed, the diagnosis process is started. No emulation of the subsystems has been implemented, so the user must supply the ES with the outcome of the actions it attempts to carry out, be these tests or activation of redundant units. When more than one failure can conceivably cause the same symptoms, the diagnosis process implemented in the ES will choose as first hypothesis the most critical one on the basis of a criticality value that is associated to each failure symptom.

IMPLICATIONS OF STUDY RESULTS

From the early stages of the study a difficulty arose in the definition of the corrective actions that were to be executed by the on-board ES: quite simply the choice was restricted to some redundancy switching. The difficulty was initially attributed to the lack of detail in the satellite that had been chosen as target; as more insight into the matter was acquired, it was recognized that the problem was more basic than it seemed and that it was also caused by the idea at the base of many a system design: to make each component as much a "black box" as possible. This was compounded by the fact that, to avoid the difficulties related to the representation of time and the reasoning about it, sequences of actions were considered only in atomic form. The bottom line is that simplification of the interfaces among units reduces integration and control problems, but severely limits the choice of action to correct a failure, to the point of making questionable the advantages of an ES compared to conventional algorithmic or table driven software. In the future the complexity of spacecrafts will increase and so will integration problems; in addition, the functional partition of the satellite in subsystems is closely mapped into areas of competence of european industries which are by now well established and not likely to be easily changed. Therefore, it is to be expected that the same type of problem will appear in the future.

Another conclusion that can be drawn from the previous paragraph is that it is difficult and not necessarily advantageous to use an ES in a satellite designed without this technology in mind. This point of view is however very much in contradiction with a ratner general attitude which is more or less expressed by "first mission requirements must be defined, and from these the need for an ES". In this respect it is interesting to note that even within this study the flow of activities effectively followed this pattern: first a reference satellite was defined on the basis of an architecture which was developed with completely different priorities, and then an ES was designed to take up the tasks that were meant to be carried out by simple on board HW or SW. One should of course accept that this represents an "initiation rite" for most new technologies; at the same time, one should be aware of its existence and of its limiting effects on the appreciation of the technology. Requirements are inevitably influenced by what is known to be feasible or available and therefore mission definition studies will take into account ES technology; however, it is also clear that to rely only on this mechanism will imply an unnecessary delay before a potentially extremely useful technology can be applied to its full capabilities.

OPEN ISSUES

A number of problems remain to be solved before ES's can be accepted for routine on board use. The first is the matter of validation, probably the biggest obstacle that needs to be overcome before on board ES applications become accepted; current on-board SW for unmanned satellites is orders of magnitudes less complex and "brute force" testing can achieve sufficient coverage. This is certainly not going to be possible with ES's, and in fact, although the problem has been identified and described, a satisfactory solution is not available; nevertheless, work is being carried out in this area, and one can feel reasonably confident that an acceptable solution will be found. However, much of the testing of ES's will continue to rely also on the assessment by human experts of the "reasoning" that generated a specific output. The reasoning is accessible only through SW facilities that are not justified in a satellite out of ground contact; therefore there is a definite risk of having to choose between two equally undesirable alternatives: installing on board a system which is overburdened by unnecessary facilities, or performing the tests without adequate support. A possible solution would be the development of a standard run time environment together with a tool for the automatic porting from the development system to the run time environment. In this way the testing and validation problem could be split in two parts, the first being the "conceptual" testing, to take place in the full ES development environment, and the second being done once and for all by the validation of a tool for the automatic translation of the ES to a "streamlined" run time version. The translation would consist essentially in the removal of the display, explanation, modification, etc. facilities and, when applicable, of a compilation.

The second problem is that space qualified HW which is currently available or in sight does not provide the resources that are needed to run in acceptable time ES's that have been developed with powerful shells. The previously described combination of run time environment and automatic translation tool could provide a meaningful contribution to the reduction of on-board resource requirements without performance penalties.

Finally, within the european space industry the problem of knowledge collection is compounded by the fact that experts are widely dispersed among different companies and countries; in this respect, procedures and tools to aid the process of knowledge collection and formalization would be particularly beneficial. The method used in this study can be seen only as a first step.

CONCLUSIONS

The two prototypes that have been implemented proved that an ES that performs a meaningful subset of the functions required for satellite autonomy is feasible in the european space industry with current technology. The line of activity has been extended to implement an ES to manage the descent of the Cassini probe on Titan. This mission has been chosen as target because no TC link will be available and because, given its relatively early stage of definition, it is hoped that the conceptual problems described earlier could be overcome. The same consortium will implement a prototype on an AI workstation and then transfer it to some HW more representative of on board resources; the purpose of the additional step is to acquire some dimensioning information on the related difficulty and to be able to run some performance tests.

INDEX BY AUTHORS

AUTHOR	PAGE
Allen, B. P.	423
Ali, M.	121
Amin, A.	427
Arima, H.	143
Bailey, David	279
Basile, Lisa	181
Belau, W.	131
Bell, Benjamin	123
Beyer, David S	303
Beyer, J.	261
Bharwani, S. S.	195
Block, Roger F.	447
Bond, W. E.	9
Bosworth, Ed	403
Brady, Michael	161
Brewster, L. T.	47
Bridges, Susan	273
Britt, Daniel L.	371
Brown, R. M.	83
Burns, Rowland E.	433
Bush, Joy	377
Byrd, J.	383
Cacace, Ralph	445
Campbell, William J.	359
Carnahan, Richard S.	255
Carnes, James R.	187, 297, 353
Castillo, D	361
Chang, Kai-Hsiung	315
Chen, Z.	165
Choudry, A.	151, 271, 401
Ciarlo, A.	453
Critchfield, Anna	377
Cruse, Bryant G.	101
Culbert, Chris	1, 53
Cutts, D. E.	187, 353
Daughtrey, Rodney	321
Dietz, W. E.	121
Donzelli, P.	453
Dudziak, Martin J.	227
Dunham, Larry	95
Dwan, W.	243

PRECEDING PAGE BLANK NOT FILMED

Eddy, Pat 103
England, Brenda 445
Erickson, W. K. 79

Feinstein, Jerald L. 227, 279
Fiesler, E. 401
Flachsbart, B. B. 9
Follin, John F. 269
Fox, B. R. 47, 309
Franke, H. 143
Freeman, M. S. 89
Friedland, P. 85

Garcia, Jr., Raul C. 213
Geoffroy, Amy L. 371
Gohring, John R. 371
Goode, Wayne 73
Gregg, H. R. 81
Guarro, Sergio 193

Hack, E. 81
Han, C. Y. 41
Harrand, V. 271
Harrington, James B. 33
Hays, Dan 333
Healey, K. J. 81
Heffelfinger, H. L. 171
Heher, Dennis 15
Hofacker, S. 59
Holtzman, P. L. 423
Holzer, J. 193
Homsley, T. 115
Hughes, Angi 383
Hyatt, Larry 193

Ihrie, D. 361

Jackson, M. E. 195
Jacobus, C. 261
Janssen, P. 151
Johannes, J. D. 221, 273, 297
Johnson, J. V. 87
Jones, J. 115

Kao, Simon A. 95, 155
Kawamura, K. 143
Kellner, A. 131, 415
Kelly, Angelita C. 181
Kesler, L. O. 65
King, James A. 409
Krishnamorthy, C. 21

Laffey, Thomas J. 95, 155
Land, Ken 65
Lazzara, Allen V. 345
Lewallen, P. 115
Littlefield, Ronald G. 303
Liu, H. 143
Liuzzi, R. 345
Lopez, F. 53

Marinuzzi, John 421
MacDonald, J. 171
McDaniel, M. 361
McIntire, G. 389
McLean, Daniel R. 303
Mendler, Andrew P. 255
Mitchell, B. 261
Modesitt, Kenneth L. 203
Morrow, Paul 425
Mullikin, Richard L. 439

Narayanan, S. 143
Newman, P. A. 249
Nguyen, Thinh 263

Otaguro, W. S. 65
Ozkan, M. 143

Padalkar, S. 21
Pelin, Alex 425
Pooley, J. 115
Potter, Andrew 327
Purves, R. B. 21, 353

Read, Jackson Y. 95, 155
Rhoades, Don 65
Richardson, K. D. 85
Riley, Gary 1, 53
Rochowiak, D. 285
Rolofs, L. 359
Ross, J. B. 41
Ryan, P. 395

St. Clair, D. C. 9
Savely, Robert T. 1, 53
Schaefer, Phillip R. 371
Schielow, N. 131
Schmidt, James L. 95, 155
Schwartz, M. R. 79
Schroer, B. J. 59, 243
Selig, W. J. 221

Shah, B. 395
Short, Jr., Nicholas 359
Smith, Richard 421
Stamps, M. E. 121
Sztipanovits, J. 21

Tanner, Steve 383
Teoh, W. 115
Tepfenhart, W. 345
Thompson, W. 115, 279
Tilley, R. 361
Toth-Fejel, Tihamer 15
Tsai, L. C. 41
Tseng, F. T. 243

VanderZijp, J. 401
Verwer, Ben J. H. 153
Villarreal, James A. 339, 389

Wakefield, G. Steve 137
Walls, J. T. 195
Wallace, Peter 193
Wang, C. K. 233
Wattawa, S. L. 359
Wee, W. G. 41
Weeks, Dave 109
White, R. 345
Wong, C. M. 81, 85, 87

Yeager, Dorian P. 291
Yee, Robert 83
Yost, R. A. 87

Zhang, G. 143

TECHNICAL REPORT STANDARD TITLE PAGE

1. REPORT NO.	2. GOVERNMENT ACCESSION NO.	3. RECIPIENT'S CATALOG NO.
NASA CP-2492		

4. TITLE AND SUBTITLE	5. REPORT DATE
Third Conference on Artificial Intelligence for Space Applications	November 1987
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S) Compiled by J. S. Denton, M. S. Freeman, M. Vereen	8. PERFORMING ORGANIZATION REPORT #
9. PERFORMING ORGANIZATION NAME AND ADDRESS George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812	10. WORK UNIT NO. M-575
	11. CONTRACT OR GRANT NO.
12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington, D.C. 20546	13. TYPE OF REPORT & PERIOD COVERED Conference Publication
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES Conference Coordinator - Thomas Dollman, Information and Electronic Systems Lab, Marshall Space Flight Center

Co-sponsored by The University of Alabama in Huntsville

16. ABSTRACT

Proceedings of a conference held at Huntsville, Alabama, on November 2 and 3, 1987. This Third Conference on Artificial Intelligence for Space Applications brings together a diversity of scientific and engineering work and is intended to provide an opportunity for those who employ AI methods in space applications to identify common goals and to discuss issues of general interest in the AI community.

17. KEY WORDS	18. DISTRIBUTION STATEMENT
Artificial Intelligence Computer Vision Design Data Capture Robotics Space Station Automation	Unclassified/Unlimited Subject Category: 61

19. SECURITY CLASSIF. (of this report)	20. SECURITY CLASSIF. (of this page)	21. NO. OF PAGES	22. PRICE
Unclassified	Unclassified	476	A-20

MSFC - Form 3292 (Rev. December 1972)

For sale by National Technical Information Service, Springfield, Virginia 22151

*U.S. GOVERNMENT PRINTING OFFICE: 1987-441-530