10

Second Conference on Artificial Intelligence for Space Applications

NASA

Second Conference on Artificial Intelligence for Space Applications

NASA

National Aeronautics
and Space Administration

Scientific and Technical
Information Division

1988

CONFERENCE ON ARTIFICIAL INTELLIGENCE
FOR SPACE APPLICATIONS
TABLE OF CONTENTS

SPACE STATION AS A VITAL FOCUS
FOR ADVANCING THE TECHNOLOGIES OF
AUTOMATION AND ROBOTICS[1]

Giulio Varsi[2]
and
Daniel H. Herman

NASA Headquarters, Washington, DC 20546

ABSTRACT

A major guideline for the design of the United States' Space Station is that the Space Station address a wide variety of functions. These functions include the servicing of unmanned assets in space, the support of commercial laboratories in space and the efficient management of the Space Station itself; the largest space asset. For the Space station to address successfully these and other functions, the operating costs must be minimized. Furthermore, crew time in space will be an exceedingly scarce and valuable commodity. The human operator should perform only those tasks that are unique in demanding the use of the human creative capability in coping with unanticipated events.

The technologies of Automation and Robotics (A&R) have the promise to help in reducing Space Station operating costs and to achieve a highly efficient use of the human in space. The use of advanced automation and artificial intelligence techniques, such as expert systems, in Space Station subsystems for activity planning and failure mode management will enable us to reduce dependency on a mission control center and could ultimately result in breaking the umbilical link from Earth to the Space Station. The application of robotic technologies with advanced perception capability and hierarchical intelligent control to servicing systems will enable us to service assets either at the Space Station or in situ with a high degree of human efficiency.

This paper presents the results of studies conducted by NASA and its contractors, at the urging of the Congress, leading toward the formulation of an automation and robotics plan for Space Station development.

KEYWORDS

---

[1] It has become customary to refer to cognitive and manipulative tasks by the terms "automation" and "robotics," respectively. This convention will be used throughout this paper.

[2] On assignment to NASA Headquarters from the Jet Propulsion Laboratory, California Institute of Technology.

INTRODUCTION

The United States' Space Station is a permanent multipurpose facility, with an initial crew of six to eight astronauts, that will serve as a research laboratory, a permanent observatory, a transportation node, a storage depot, and a base for staging missions to higher orbits, the planets and beyond. It is also a facility for assembling complex payloads and for servicing satellites and instruments. To fulfill this variety of functions, the Station is designed as a very complex system consisting of a modular manned base and several unmanned free flyers provided by several nations, Fig. 1.
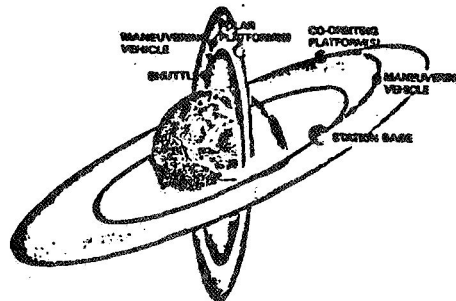
Fig. 1. Initial Space Station Complex

In the words of the National Commission on Space (NCOS, 1986), the Station is the first of 12 technological milestones in space toward a bridge between worlds and the beginning of the Earth Spaceport: for such a "port" we can easily envision a rich and always growing ferment of varied activities. Because of the size of the investment in the Space Station and the expected long life of this facility, versatility of architecture and capability to add new features must be provided from the beginning in the design of the system and its subsystems.

ROLE OF AUTOMATION AND ROBOTICS

Two resources that are critical for the construction and operation of the Station are: payload in orbit and crew time. Because of the legacy of the destruction of Challenger and the limitations of current technology, both are becoming more precious as we impose stricter limitations to enhance safety. While the payload limitations may be overcome in the not too distant future by the development of heavy-lift vehicles and performance improvements to

the shuttle, the availability of crew time will always be at a premium.

The judicious application of the technologies of automation and robotics can overcome the limitation of this vital resource as shown in Fig. 2, which summarizes one result of a recent study (Boeing, 1986). The study indicates that, by implementing a series of A&R applications consistent with the advancement of technology and the scope of the station program, a given crew can increase the number of its members devoted to productive functions by a factor of two to three. (Although the Skylab technology[3] used in the comparison is not a realistic option for the Station, it is the only U.S. long term experience and, therefore, a useful baseline for comparisons). As we shall see again later, such a productivity increase by a factor of two to three has been noted by other analysts.

The issue of which specific A&R technologies have the greatest leverage on Space Station productivity was studied by the Automation and Robotics Panel (ARP), a group of over 30 leading technologists. The results of their six-month analysis are summarized in Fig. 3 (ARP, 1985). For each one of the three broad classes, the panel indicated the specific technologies where NASA should, respectively, lead, leverage, and exploit. As we examine this table, we find that, in the range of autonomy development shown
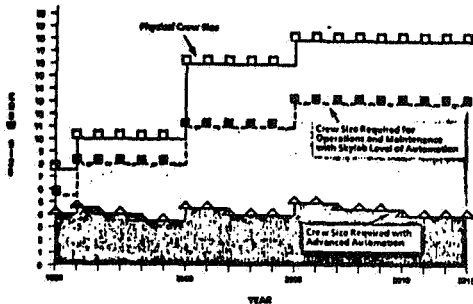
Fig. 2. Space Station Productivity Projection

in Fig. 4, the Space Station thrust is in teleoperated and supervised control. These technologies are at the leading edge for applications to flexible manufacturing and information management, the major frontiers in terrestrial applications.

CHARACTERISTICS OF SPACE STATION

There are four characteristics of the Space Station that render it an excellent setting for developing the technologies described.

Costs and Benefits

The cost of implementing any one of the A&R applications currently considered for the Station is comparable to that of implementing the equivalent terrestrial one and is quite substantial, ranging from the low 10^6 dollars to the high 10^7 dollars. However, on Space Station, the ensuing productivity gains have extraordinary value: $10^4 - 10^5$ dollars/work-hour saved, depending on whether it is for IVA or EVA, and also depending on the valuation method. Thus, the high initial investment is much more

[3]For example: Open-cycle life support system; semi-active thermal control; inertial, solar, and z-vertical attitude control. No self-checking, trend analysis, etc. (NASA, 1977).

Manual	Unaided IVA/EVA, with simple (unpowered) hand tools
Supported	Requires use of supporting machinery or facilities to accomplish assigned tasks (e.g., manned maneuvering units and foot restraint devices)
Augmented	Amplification of human sensory or motor capabilities (powered tools, exo-skeletons, microscopes, etc.)
Teleoperated	Use of remotely controlled sensors and actuators allowing the human presence to be removed from the work site (remote manipulator systems, teleoperators, telefactors)
Supervised	Replacement of direct manual control of system operation with computer-directed functions although maintaining humans in supervisory control
Independent	Basically self-actuating, self healing, independent operations minimizing requirement for direct human intervention (dependent on automation and artificial intelligence)

Fig. 4. Range of Autonomous Operation

Class	Man/Machine-Robotics	Information Management	Communication & Mechanical Infrastructure
NASA Leads	Space Manipulators Materials Handling Technologies Robot Mobility in Space Man/Machine Interfaces	Expert System Mission & Payload Control, Planning & Directing "Smart" Simulations Fault-Tolerant, Reliable Software Tools CAE Standards Software Language Standards Automation Design for Integrating Technology	Lightweight Structure & Assembly Spare Parts & Repair Technology Fluids Transfer Technologies
NASA Leverages	Robot Sensors/Integration Reconfigurable & Repairable Robots High-Level Robot Programming Language	Space-Related Custom Hardware Communications Networks Distributed Large-Scale Databases CAD-Directed Programming Knowledge-Based System Development Sensing Algorithms Real-Time Systems Facility "Seed" Funding	New Fabrication Technologies
NASA Exploits	Terrestrial Robots and Manipulators Lightweights Motors	Computer Architecture Chip Technologies Speech Technologies	Local Area Networking Display Technologies High Bandwidth Technologies Communication Technologies

Fig. 3. Space Station Classes of A&R Technology

2

readily justifiable in space than in terrestrial applications where a typical work-hour is worth $10 - 10^2$ dollars.

Environment

The space environment is relatively well structured and foreseeable, by comparison with that of an urban or an office setting, where a multitude of living entities interact unpredictably, and thus, it lends itself to the modeling that is indispensable to autonomous systems. It is also very complex, with millions of parts, intersecting control loops, cascading interactions from subsystem to subsystem, and multi-layered hierarchies of functions that are very taxing for humans, specially when working under pressing time constraints. Finally, because of its complexity and notwithstanding its relative predictability, this environment has a sufficiently large variety of possible configurations that preprogrammed automation or even detailed procedural prescriptions for all foreseeable eventualities are not feasible. Thus, humans can be very effectively elevated to the supervisory role, once the machines are endowed with autonomous local sensors and feedback, with sufficiently comprehensive declarative models, and with overrides for untested or unmodeled circumstances.

Users

The exceptional qualifications of the space users, who are all expert and trained technologists and scientists able and interested in contributing to the development of the technology, are of significant advantage in developing the operator interfaces, often one of the most difficult and little understood areas.

State of the Art

The situation presented in Fig. 3, which shows that there are areas where NASA must lead, others where it can adapt and leverage current advances, and others yet which it can exploit, indicates the breadth of the range of approaches available for R&D and the ample possibilities for advancing the state of the art.

HISTORICAL SUMMARY

The Congress of the United States sought to take advantage of this unique setting and of the historic opportunity to foster A&R by requesting in 1984, that the Office of Technology Assessment conduct a workshop (March 1984) to explore the relationship between the Space Station Program and advanced A&R (OTA, 1985) and by including A&R in the Space Station enabling legislation.

Public Law 98-371 mandates that NASA identify Space Station systems that would advance A&R technologies beyond what is in use in current spacecraft. Congress further requested that an Advanced Technology Advisory Committee (ATAC) be established to fulfill the mandate and to report to Congress every six months on NASA's progress.

At NASA's direction, five aerospace firms examined, "without regard to cost," the A&R applications that might be included in the Station as it evolves. The contractors' six-month studies focused on these areas:

Boeing Companies: Operator-system interface (BOEING, 1984);

General Electric Company: Manufacturing in space (GE, 1984);

Hughes Aircraft Company: Subsystem control and ground support (HUGHES, 1984);

Martin Marietta Aerospace: Autonomous systems and assembly (MARTIN, 1984);

TRW: Satellite servicing (TRW, 1984).

In addition, NASA funded SRI International to conduct an assessment of the studies from the viewpoint of artificial intelligence technology (SRI, 1985) and the California Space Institute to organize the ARP - mentioned earlier - and conduct an independent study. This panel confirmed that advanced A&R would improve productivity on the Station and yield benefits nationwide; recommended a major NASA investment in related R&D (climbing rapidly to between 100 to 200 million dollars/year); and stated clearly the requirement that the Station program must be designed for growth from the beginning (ARP, 1985).

The findings of all these studies were consolidated by the ATAC in its first report to the U.S. Congress (ATAC, 1985a). The committee, recognizing the difficulty of accommodating the ambitious proposals and the projected budgets articulated its position in 13 recommendations segregated into two groups. The first group of eight - to be implemented within the nominal budget - focused on: A&R as a significant element of the Space Station Program; maximum adoption of current R&D; the requirement to design for growth; the importance of verification and validaton; and the use of A&R for the management process. The second group of five recommendations focused on aggressive development of advanced A&R, conditional on budget augmentation.

In November 1985, the U.S. Congress expressed desire that greater and faster progress be made in A&R than what would be possible within the normal Space Station budget and provided, in successive augmentations, additional funding for a flight telerobot. This will be a versatile system to be used as an aid in Station assembly and maintenance, and in payload servicing tasks, and it will be transportable in space by a variety of carriers based on Station and on shuttle, Fig. 5.

Additional insight into the need and role of A&R is provided by a White Paper from the Astronaut Office (ATAC, 1985b), which recommended: application to repetitive, time consuming, time critical, taxing, hazardous, boring tasks; performance of early flight tests; provision for human override.

More recently, the need for A&R was also present in the call for the development of telescience by the Space and Earth Sciences Advisory Committee (SESAC, 1986) through its Task Force on Scientific Uses of Space Station. Specifically, the task force recognized the projected evolution from space science (principally observatory in character) to laboratory science in space (principally experimental). Telescience is described as the ability to conduct

experiments and reprogram them quickly - and
remotely - as current results are understood and
new opportunities uncovered. This also re-
quires advanced forms of telecommunications and
automation using supervisory control.
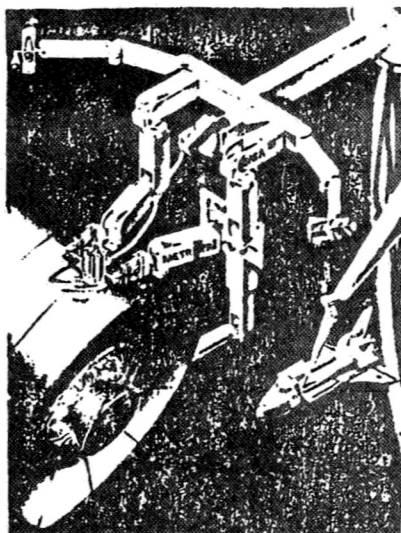
Fig. 5. Flight Telerobotic System: A Concept

APPROACH TO AUTOMATION AND ROBOTICS AND STATUS

The Space Station is now nearing the end of the
definition and preliminary design phase which
will be completed in December 1986. The four
Space Station Lead Centers: Marshall Space
Flight Center, Johnson Space Center, Goddard
Space Flight Center, and Lewis Research Center
and their eight industrial contractors have
been performing preliminary designs and eval-
uations of over a hundred different A&R concepts
for specific applications. Evaluation criteria
include: reduction of crew time devoted to
operations and maintenance; initial costs and
operation savings; system availability; safety;
terrestrial spin-offs; design risk; and impact
on ground operations.

In addition, experimental and theoretical R&D
efforts are led by the Ames Research Center (for
cognitive functions) and by the Jet Propulsion
Laboratory (JPL) (for manipulative functions).
Demonstrations of their R&D are planned at
approximately two to three year intervals in
collaboration with the Space Station Lead Cen-
ters. The first demonstration is scheduled to
be by JPL, on telerobotics, in 1988.

NASA contractors (BOEING, 1986; GE, 1986; MAR-
TIN, 1986; MCDONNELL, 1986; RCA, 1986; ROCKET-
DYNE, 1986; TRW, 1986; ROCKWELL, 1986) have
identified over 100 useful A&R applications for
the initial configuration of the Station and
ATAC (1986) has culled the list to about 18
cognitive, eight manipulative and four addi-
tional applications requiring more advanced
techniques. Of principal interest in the first
category, we find: system management and crew
activity planners; data base management; power
system control and management; and monitoring
and fault detection for life support systems.
In the second category: Space Station assembly,

inspection and repair; payload servicing; and
docking.

The studies conducted at the NASA Centers, and
at the contractors, confirm the selection of
this approach to A&R for the Space Station: the
human operator, who is in charge of the task at
all times, assigns to the machine, directly or
by default, those operations that, in his jud-
gement, can be well performed automatically at
that time. The operator reserves the option to
resume control during the execution and com-
plete the operation directly. The "level of
abstraction" of the operator's actions can be
adjusted dynamically: the machine functions
similarly to the apprentice of a master crafts-
man. Thus, the terminology: "astronaut assis-
tant" or "aide." This approach is well suited
to the traditions and needs of the manned space
endeavors and is conceptually and technically
different from that generally followed in the
unmanned space flight (Pivirotto, 1986; Varsi,
Man, and Rodriguez, 1986), where machines are
given complete autonomy, but within narrower
bounds, specified in advance (e.g., Viking's
landing sequence).

Technically, the general area of intelligent
autonomy is being pursued vigorously with spe-
cial focus in the areas of sensors and world and
knowledge representation. As a consequence of
the approach chosen, the key technology of
shared or "traded" control between operator and
machine is being developed and, within it,
particular emphasis is given to the control
architecture and the man-machine interfaces.

Programmatically, the guidelines to the con-
tractors for the next phase of work: design,
development and construction, are expected to
emphasize the themes discussed here and require
a plan for the implementation of A&R applica-
tions. Information from the preliminary plans
available now indicates that levels of autonomy
that are technically achievable by 1993 - 1994
can increase productivity on the Station by a
factor of two to three and allow recovery of the
investment in about two to three years for the
majority of applications. It has been shown
(THURIS, 1984) that it is necessary to pay
particular attention to the sequence according
to which applications are developed and imple-
mented: in financial terms, the investment
hurdle of an application can be reduced by
several-fold if that application is implemented
as a part of a group of related applications.

AUTOMATION AND ROBOTICS BENEFITS

We shall now review three specific applications
and summarize the analyses, albeit preliminary,
of expected benefits.

System Management

This application is variously conceived and
called "System Manager," "Station Coordinator,"
"Operation Manager." In its broad conception,
it has the function of translating Station
performance and scheduling requirements into
task sequences for subsystems. It can be
considered an "expert system" hierarchically
controlling other expert systems.

It contains a representation of the Station and
of the systems it interacts with, receives real

time information on the status of the crew, the hardware and software of the Station, and the operation and science requirements, and constructs and updates schedules of activities. One configuration of this application (MCDONNELL, 1986) is expected to require about 8,000 "rules" and to cost 40 million dollars. Its yearly benefits have been estimated at 90 EVA hours, 1,350 IVA hours, and 32,000 work-hours on the ground: with a total value of about 60 million dollars.

Power Management

Three options can be considered: a ground based or "manual" system; a "conventional automation" (e.g., on-board load shedding on the basis of preprogrammed priorities); and a more advanced management (e.g., on-board resources optimization and failure recovery), based on expert system technology. On a uniform basis, the initial costs are estimated at: 45, 75, and 85 million dollars, respectively; and the operating costs at: 30, 13, 10 million dollars/year (NASA/LeRC. 1986). This example illustrates vividly the programmatic dilemma offered by A&R applications: on one hand, the 20 million dollars/year difference between the extreme cases allows recovery of the 40 million dollar difference in initial costs in only two years; on the other hand, that 40 million dollar difference represents about a doubling of the initial costs - hardly affordable program-wide. Thus, the need for a judicious selection of applications based on a careful Station-wide system analysis.

Telerobot

The Station telerobot is an evolvable system that will include the capability for both pure teleoperation with telepresence, as well as full autonomy of selected functions, the repertoire of which is designed to expand greatly during the useful life of the Station. In addition, its architecture will allow for graceful sharing of control between operator and machine. The analysis summarized here (GRUMMAN, 1986) compares EVA with the teleoperation capability only for an assembly task. This analysis uses EVA work-hours as a unit of account and sidesteps, in part, the difficulty of assigning costs when experience is uniformly lacking and, as it is the case with marginal costs, even the methods for determining them are speculative. It is generally assumed that present technology permits about 50 EVA hours per week-long shuttle flight, on the basis of two days of space adaption, two teams of two EVA and one IVA astronauts each, working six hour/day on alternate days, and one day of cleanup. The investigators found, experimentally on the ground, that, with comparable training, the execution time increases by a maximum of a factor five in teleoperation, for tasks requiring dexterity; however, the telerobot could be operated at least 16 hours/day, in shifts, for almost six days. The combination of these factors for the expected mix of assembly tasks would produce an increase of over 50 percent in production per astronaut applied to the task. If autonomous operation, even to a very limited degree (e.g., the alignment and locking steps) were considered, the advantages would be even greater.

CONCLUSIONS

The Space Station Program provides a high payoff

environment conducive to the development of advanced A&R technology in the areas of: hierarchical architectures, artificial intelligence tools and prototyping technology, adaptive controls, rapid planning and replanning, and verification and validation. Notwithstanding the exceptional conditions for a high return on investment, the cost of this technology is high, both in absolute and relative terms (as shown in the example above, it can double the cost of a subsystem) and, therefore, a continuing commitment to A&R is required on the part of the program and the U.S. Congress.

The effort and the investment will not only enjoy a relatively rapid return, but, in advancing the technologies mentioned above, will specifically foster greater versatility, flexibility and "intelligent behavior" in machines and hasten the departure from preprogrammed automation, which requires very long production runs to justify its adoption. This is now the main form of automation in industrial applications: about 50 percent of all robots are used in automotive production. The need and the potential impact of the more advanced forms of A&R, which the Space Station will foster, can be gaged by the fact that over 75 percent of the total value added in manufacturing is attributable to non-mass production methods (Miller, 1986). The potential impact of the introduction of flexible automation for assembly has been analyzed recently by Funk (1984) and one representative estimate is displayed in Fig. 6. The costs are derived for a "standard" product on the basis of industrial statistics for a variety of assembly tasks.
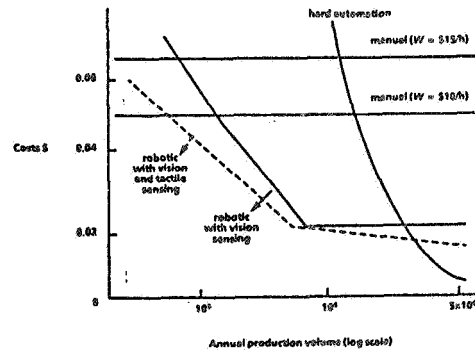
Fig. 6. Impact of Advanced Robotics on Manufacturing (Adapted from Funk, 1985)

REFERENCES

ARP (1985). Automation and Robotics for the National Space Program. Automation and Robotics Panel. California Space Institute Report CSI/85-01.

ATAC (1985a). Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy. Advanced Technology Advisory Committee (ATAC) Report to the U.S. Congress. NASA TM 87566.

ATAC (1985b). Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy. Advanced Technology Advisory Committee (ATAC) Report to the U.S. Congress. Progress Report No. 1, October 1985. NASA TM 87772.

ATAC (1986). Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy. Advanced Technology Advisory Committee (ATAC) Report to the U.S. Congress. Progress Report No. 2, March 1986. NASA TM 88785.

BOEING (1985). Space Station Automation and Robotics Study. Boeing Aerospace Company and Boeing Computer Services Company. D483 10027-1.

BOEING (1986). Automation and Robotics Plan. Boeing Aerospace Company. DR-17 Data Package WP-01, D483-50055-1, June 30, 1986.

GE (1984). Space Station Automation Study: Automation and Requirements Derived from Space Manufacturing Concepts. General Electric Space Systems Division, Contract No. NAS5-25182.

GE (1986). Space Station Definition and Preliminary Design. DR-17 Automation and Robotics Plan. WP-03. June 30, 1986, General Electric Company. DR-17, Contract No. NAS5-29300.

GRUMMAN (1986). Space Station Assembly Study. March 1986. Internally funded by the Space System division of Grumman Corporation. Private communication from Mr. G. Fischer.

HUGHES (1984). Automation Study for Space Station Subsystems and Mission Ground support, Hughes Aircraft Company, Contract No. 82-14F.

MARTIN (1984). Space Station Automation Study. Martin Marietta Aerospace, MCR84-1878.

MARTIN (1986). Space Station Definition and Preliminary Design. Martin Marietta Aerospace. DR-17 Data Package WP-01, SSP-MMC-00022, rev. A, June 30, 1986.

MCDONNELL (1986). Automation and Robotics Plan. McDonnell Douglas Astronautics Company, DR-17 Data Package WP-02, MDC H2036A, June 1986.

Miller (1986). Private communication.

NASA (1977). Skylab, Our First Space Station. NASA SP-400.

NASA/LeRC (1986). Preliminary Cost Estimate for Automation Alternatives for the Electric Power System. NASA Lewis Research Center. Private communication from J. Dolce.

NCOS (1986). Pioneering the Space Frontier. The report of the National Commission on Space. Bantam Books, Inc., New York, p. 17.

OTA (1985). Automation and Robotics for the Space Station: Phase B Considerations. Office of Technology Assessment staff paper.

Pivirotto, D. S. (1986). Unmanned Space Systems. 13th Annual Symposium Association of Unmanned Vehicle Systems. July 21-23, 1986.

RCA (1986). Space Station Definition and Preliminary Design, DR-17 Automation and Robotics Plan. WP-03. RCA Astro Electronics. Contract NAS5-29400. June 30, 1986.

ROCKETDYNE (1986). Interim Automation and Robotic Plan, DR-17. WP-04. Rocketdyne Division. Contract No. NAS3-24666. June 30, 1986.

ROCKWELL (1986). Automation and Robotics Plan. Rockwell International DR-17 Data Package WP-02, SSS85-0190A, June 13, 1986.

SESAC (1986). Space Station Summer Study Report. Space and Earth Sciences Advisory Committee (SESAC) Task Force on Scientific Uses of Space Station (TFSUSS).

SRI (1985). NASA Space Station Automation: AI-Based Technology Review. SRI Project 7268.

THURIS (1984). The Human Role in Space. McDonnell Douglas Astronautics Company, MDC H-1295.

TRW (1984). Space Station Automation Study - Satellite Servicing. Z410.1-84-160.

TRW (1986). Space Station Definition and Preliminary Design WP-04, Electrical Power Systems (EPS) - DR-17 Automation and Robotics Plan. TRW. Contract No. NAS3-24655. June 30, 1986.

Varsi, G., G. Man, and G. Rodriguez (1986). Automation of Planetary Spacecraft. Submitted to Unmanned Systems.

A Relational Data-Knowledge Base System

And

Its Potential in Developing a Distributed Data-Knowledge System

by

Eric N. Rahimian and Sara J. Graves*

This paper describes a new approach used in constructing a rational data-knowledge base system. The relational database is well suited for distribution due to its property of allowing data fragmentation and fragmentation transparency. This paper will formulate an example of a simple relational data-knowledge base which may be generalized for use in developing a relational distributed data-knowledge base system. The efficiency and ease of application of such a data-knowledge base management system will be briefly discussed. The paper will also discuss the potentials of the developed model for sharing the data-knowledge base as well as the possible areas of difficulty in implementing the relational data-knowledge base management system.

I. INTRODUCTION

Distributed database systems have been developing during the last ten years. The knowledge bases have also been growing at a fast pace during the same period. A group of knowledge bases are considered as supportive resources to enrich databases and to facilitate the end-users utilization of databases. So far, the study of the (distributed) databases which are supported by knowledge bases has been a very technical area of research. Nevertheless the interest in the subject has produced a large amount of literature under different titles including Expert Database Systems, Intelligent Database Assistants, and Data-Knowledge Base Systems.

This paper briefly reviews the nature of the relational models, knowledge bases, expert systems, and the integration of knowledge bases and databases into a data-knowledge base system.

---

*Rahimian is with Alabama A & M University and a graduate student in Computer Science at UAH. Graves is with the Computer Science Dept. at UAH.

II. The Complete Integration of Knowledge Bases and Databases: A Dream Which May Come True

As Korth and Silberschatz [2] observe, usually the reasoning (rule processing) part of the expert system interacts with a standard database system.

"In its simplest form, the expert system interacts with the database system as a casual user. The expert system submits queries in a database language such as SQL and awaits an answer from the database system". [2] This form is simple but not an optimal design since the rules in the knowledge base are not available for use in processing a database query. In addition when the expert system poses a series of related queries, the database system can not take advantage of the similarity of the queries and must process each one individually. Due to these inefficiencies, several alternatives have been considered, including:[2]

(1) Loading into the expert system that part of the database that is needed for processing rules. The database itself is stored and maintained separately from the expert system and periodically, the expert system's copy of the data is updated.

(2) Implementation of a database system within the expert system.

(3) Translating into the relational algebra certain logical queries posed to the expert system. The resulting algebra expression is passed to the database system for optimization and execution.

Finally, Korth and Silberschatz assert that the ultimate solution is probably the integration of database and expert systems into a single system in which the rule processing component has access to low-level internal database system components.

III. A New Approach in Integrating A Relational Database and A Knowledge Base

Han-lin Li has recently (April 1986) proposed and tested a new approach for integrating data and knowledge bases which is very promising and simple [3]. His system uses one integrated relational data-knowledge base system under the control of a relational database management system (RDBMS). More specifically, the human knowledge is translated into relational entities (tables).

By this translation, the knowledge files can achieve the same structure as relational data files. Then, both kinds of files can be integrated as a data-knowledge base system and may be manipulated by a relational database management system. Naturally, this system, when proved possible, is easier to understand, manipulate and expand.

Human knowledge may be represented by production rules. The basic form of production rule is "IF condition, THEN action with certainty factor CF." "AND" and "OR" operators may be used in either condition or action part. The following table provides some example of production rules. "D" is denoting a decision or action and CF(D) is the certainty factor associated with "D".

Rules 1 and 2 of the table imply that for the same IF statement there might be different THEN actions. Rule 3 shows that different combinations of IF conditions might have the same THEN actions. Two important questions are how the production rules are translated to relations, and whether such a translation is justified.

8

TABLE 1. Some Example of Production Rules

Rule 1: IF A = a and B = b and C = c

    THEN: D = d1 with CF(D) = cd1

Rule 2: IF A = a and B = b and C = c

    THEN: D = d2 with CF(D) = cd2

Rule 3: IF A = a and (B = b or C = c)

    THEN: D = d3 with CF(D) = cd3

Rule 4: IF A = a' or B = b' or C = c'

    THEN: D = d4 with CF(D) = cd4

Rule 5: IF A = a.

    THEN: D = d5 with CF(D) = cd5

    Otherwise: D = d6 with CF(D) = cd6

Rule 6: IF A = a' and B = b and C = c

    THEN: D = d7 with CF(D) = cd7 and

    E = e with CF(E) = ce

TABLE 2: R1, a relation containing rules 3, 4, 5 and 6

Rule #	IF:			THEN:			
	A	B	C	D	CF(D)	E	CF(E)
3	a	b		d3	cd3		
	a		c	d3	cd3		
4	a'			d4	cd4		
		b'		d4	cd4		
			c'	d4	cd4		
5	a			d5	cd5		
	ā			d6	cd6		
6	a'	b	c	d7	cd7	e	ce

TABLE 3: R2, a relation containing rules 1 and 2

Rule #	IF:				THEN:		
	A	B	C	D	CF(D)	E	CF(E)
1	a	b	c	d1	cd1		
2	a	b	c	d2	cd2		

TABLE 4: Comparison of Some Major DDBMS Systems

Systems	Define Fragments	Define Replicas	Optim Cost Factors	Compile Query	Query Planning	Concurrency Control	Deadlocks	Data Model
DDM	Horizontal	Yes	Comm, CPU	Yes	Central-ized	Locking	Centralized Detection	Functional
DDTS		Yes			Yes	Semicon-trolled	Locking	Prevention	Entity-Relationship
Distributed INGRES	Horizontal	No	Comm, CPU	No	Central-ized	Locking		Relational
ENCOMPASS	Horizontal	No		No	Central-ized	Locking	Time-out	Relational
POLYPHEME	No	No	Comm	No	Central-ized	None		Relational
POREL	Horizontal	Yes	Comm	Yes	Central-ized	Locking	Avoidance	Relational
R*	No	No	Comm, CPU, I/O	Yes	Semicon-trolled	Locking	Distributed Detection	Relational
SDD-1	Horizontal Vertical	Yes	Comm	No	Central-ized	Timestamp	Avoidance	Relational
SIRIUS-DELTA	Horizontal Vertical	Yes	Comm	Yes	Central-ized	Locking	Prevention	Relational
VDN	Horizontal Overlap OK	Yes			Central-ized	Locking by User		Relational

TABLE 5. Relation S, (DF1) Supplier Data

S#	Sname	Srept	Place
S1	General	Excellent	AL
S2	Phillips	Good	IN
S3	Sunco	Bad	MA
.	.	.	.
.	.	.	.
.	.	.	.

TABLE 6. Relation Q, (DF2) Parts' Quality and Price

S#	P#	Price	Quality (Qulty)
S1	P1	105	High
S1	P2	54	Low
S1	P3	9	High
S2	P1	96	Medium
S2	P2	56	Low
S2	P3	6	High
.	.	.	.
.	.	.	.
.	.	.	.

TABLE 7. Relation P, (DF3) Parts' Name and Type

P#	Pname	Type
1	LS	Micro
2	P11	Mini
3	3M	Main
.	.	.
.	.	.
.	.	.

Li suggests that production rules with the same or similar set of IF conditions be grouped as a relation. To assure that the resultant relation is in the fourth normal form, all "IF conditions" are converted into the key of the relation. Then the given set of actions are transformed into non-key attributes and are fully functionally dependent on the primary key.

Following this method, the production rules in Table 1 may be transformed into two relations R1 and R2. Tables 2 and 3 show the two relations.

Notice that using "OR" operators in the condition part of a production rule will produce additional tuples in the produced table (relation). Also notice that in Table 3, the condition part is modified to include the decision D. This is because rules 1 and 2 have the same IF condition and inclusion of D in IF part allows creation of two tuples with unique identifiers.

The architecture of Li's relational data-knowledge base system is illustrated by the following figure:[3]
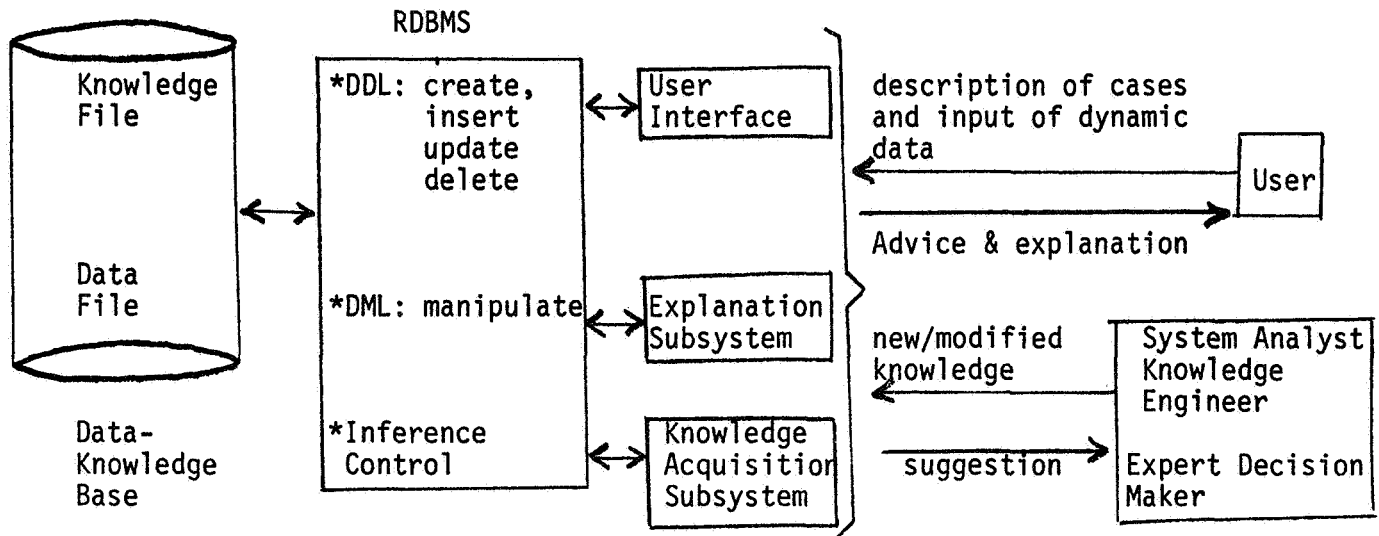
Fig. 1: The Architecture of the Data-Knowledge Base Managment System.

Using three relations or data files (DF1, DF2 and DF3) and a rule file (RF1), Li provides an empirical evidence for the running time efficiency of his approach as compared with other methods. More specifically, he shows that if at least one of the relations in the database has more than 500 tuples or when the knowledge file is not small, his approach of converting the knowledge file into 4NF relations is more efficient in running time [3].

IV. Query Processing in the System for Distributed Databases (SDD-1)

C. Mohan has provided a brief comparative study of important distributed data base systems. He has summarized the features of the major distributed data base systems which is reproduced in table 4.[4] Using the results of this comparison, SDD-1 is chosen as an appropriate distributed data base system for this study.

10

SDD-1 is a relational and homogenous distributed database management system. SDD-1 allows both horizontal and vertical fragmentation (select and project operators). Replication is possible and communication cost factor may be optimized. An important feature of SDD-1 is the use of semijoins, which when properly utilized, will reduce the cost of query processing. In SDD-1, queries are expressed in Datalanguage and are translated by the system to QUEL for performing optimization [5]. Datalanguage may be embedded in host programs in essentially the same manner as QUEL or SEQUEL. This means the host program issues self-contained Datalanguage commands to SDD-1 which processes these commands as if they were entered by an end-user. Each Datalanguage command is called a transaction [5]. The user transactions are unaware of data distribution or redundancy. It is the responsbility of SDD-1 to translate from relations to logical fragments, and then to select the stored fragment to be accessed, when processing a given translation.

In SDD-1, optimization of a query begins by translating the query which is in Datalanguage into a relational calculus form called an envelope (E). An envelope is an aggregate-free QUEL query. Envelopes are processed in two phases. First, relational operations at various sites of the distributed database are done. This delimits a subset of the database, called a reduction of database, that contains all data relevant to the envelope. The second phase consists of transmitting the reductions to one designated site, where the query is executed and the optimization is completed.

V. Combining the Query Processing Techniques of SDD-1 and The Proposed (Li's) Relational Data-Knowledge Bases System

This section provides an example of a data-knowledge base model where relations are stored at different sites. The example will show that the optimization features of SDD-1 may be combined with the data-knowledge base management system proposed by Li.

Example: Consider the following four relations. The numbers below each entity represents, its cardinality. NA means not applicable/available.

S	(S#	Sname	Srept	Place)	Where S# is the key attribute
10000	10000	NA	3	50	
Q	(S#	P#	Price	Quality)	Where (S# P#) is the key
100000	1000	1000	NA	3	
P	(P#	Pname	Type)		Where P# is the key
10000	10000	NA	5		
PAUX	(P#	Uprice	Wprice	Lprice)	Where (P# Uprice Wprice) is
10000	10000	-	-	3	the key

In relation P, there are 5 possible types where one of them is assumed to be micro. There are three possible price levels (Lprice): expensive, average, and cheap. There are three possible qualities: high, medium, and low. There are three level of reputation: excellent, good, and bad. There are fifty possible suppliers' places (states).

Notice that the values of Uprice and Wprice in PAUX relation are set by decision makers to determine the price level for each product in the market. Li has used this relation as one of his data files. But the reader may notice that its structure resembles the rule file structure (if condition, then action). These data files may be conceived as tables 5, 6, 7 and 8. In addition to these data files,

11

TABLE 8. Relation PAUX, (DF4) Parts' Price Level

P#	Uprice	Wprice	Lprice
P1	109	100	Expensive
P1	99	90	Average
P1	89	80	Cheap
P2	69	60	Expensive
P2	59	50	Average
P2	49	40	Cheap
.	.	.	.
.	.	.	.
.	.	.	.

TABLE 9. The Initial Rule File (RF1)

Rule 1   IF srept=bad THEN priority=no consid. and cf(prty)=1

Rule 2   IF qulty=low THEN priority=no consid. and cf(prty)=1

Rule 3   IF srept=excel. and qulty=high and lprice=average
          THEN priority=1st and cf(prty)=.7

Rule 4   IF srept=excel. and qulty=high and lprice=expensive
          THEN priority=1st and cf(prty)=.3

Rule 5   IF srept=excel. and qulty=medium and lprice=cheap
          THEN priority=2nd and cf(prty)=1

Rule 6   IF srept=excel. and qulty=medium and lprice=average
          THEN priority=2nd and cf(prty)=1

Rule 7   IF srept=good and qulty=high and lprice=cheap
          THEN priority=1st and cf(prty)=.6
          OR priority=2nd and cf(prty)=.4

Rule 8   IF srept=good and qulty=high and lprice=average
          THEN priority=2nd and cf(prty)=1

Rule 9   IF srept=good and qulty=high and lprice=expensive
          THEN priority=2nd and cf(prty)=.5
          OR priority=3rd and cf(prty)=.5

Rule 10   IF srept=good and qulty=medium and lprice=average
          THEN priority=3 and cf(prty)=.2 OR...

Rule 11   IF...
          .
          .
          .

TABLE 10: RF2

No. of corresp. rule in RF1	IF: (conditions)			THEN: (recommendations)	
	Srept	Qulty	Lprice	priority	cf(prty)
[1]	bad			no consid	1.
[2]		low		no consid	1.
[3]	excel	high	average	1st	1.
[4]	excel	high	expensive	1st	.7
	excel	high	expensive	2nd	.3
[5]	excel	medium	cheap	2nd	1.
[6]	excel	medium	average	2nd	1.
[7]	good	high	cheap	1st	.6
	good	high	cheap	2nd	.4
[8]	good	high	average	2nd	1.
[9]	good	high	expensive	2nd	.5
	good	high	expensive	3rd	.5
.					

TABLE 11: KF1

Srept	Qulty	Lprice	status	priority	cf(prty)
bad			1	no consid.	1.
	low		1	no consid.	1.
excel	high	average	1	1st	1.
excel	high	expensive	2		
excel	medium	cheap	1	2nd	1.
good	medium	average	1	2nd	1.
good	high	cheap	2		
good	high	average	1	2nd	1.
.	.	.	.	.	.
.	.	.	.	.	.

TABLE 12: KF2

Srept	Qulty	Lprice	priority	cf(prty)
excel	high	expensive	1st	.7
excel	high	expensive	2nd	.3
good	high	cheap	1st	.6
good	high	cheap	2nd	.4
good	high	expensive	2nd	.5
good	high	expensive	3rd	.5
.	.	.	.	.
.	.	.	.	.

TABLE 13. Response to An Example Query

S#	Sname	P#	Priority	Cf(priority)
1	General	1	1	.7
			2	.3
2	Phillips	1	3	.2
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

there exists a rule file which is adopted from Li's example. For the sake of clarity, however, it is reproduced in Table 9. Prty is an abbreviation for priority.

The rules of RF1 may be rewritten in tabular form as Table 10. However, whenever the cf is not 1, the IF condition in Table 10 is not unique. Therefore, Li has suggested the creation of two knowledge files (KF1 and KF2) which are in forth normal form. Notice that the added attribute "status" in KF1 is used to indicate the number of THEN action(s) following the IF condition. Hence, when status $\neq 1$, the search will go to KF2. In KF2 priority is included in the condition part (primary key). This technique assures us that both KF1 and KF2 provide a unique identifier (primary key) and are in 4NF.

Now our model has four data files and two normalized relational knowledge files. Assume the three relations S, Q, and P are stored at three different sites. Also assume that PAUX, KF1, and KF2 are either at a central site or they are replicated and available in all sites. Notice that the techniques of optimization by reduction may not be needed for PAUX, KF1, and KF2 because these relations may not be susceptible to useful reduction. In other words the structure of these relations are such that the action (recommendation part) do not have predefined selectivities [1] and may not produce useful reductions. The usefulness of reductions will be discussed later.

One may then use the techniques of query processing in SDD-1 on possible queries made. Consider the query:

List the priority of purchases from suppliers in Massachusetts (MA) who provide parts of type Micro.

The cardinalities of relations and their attributes were given in the beginning of this section. The envelope for this query is:

Retrieve into S (S#, Sname, Srept, Place) where qual.
Retrieve into Q (S#, P#, Price, Qulty)   where qual.
Retrieve into P (P#, Pname, Type)   where qual.
    qual: S.Place = "MA" $\cap$ P.Type = "Micro"
        $\cap$ S.S# = Q.S# $\cap$ Q.P# = P.P#
    Where qual. stands for qualification.

The analysis of Fig. 2 shows the computation of the cost and benefits for processing the semijoins related to the above query and envelope. The semijoin selected in each stage is denoted by a circled letter on the left side. The same letter is used in the flowgraph of the reductions (Fig. 3). The numbers in the modes of Fig. 3 are cardinalities of the (reduced) relations and their attributes.

Notice that site Q has the largest amount of data for the query. If site Q is selected as the place where the query is executed, then the reduction indicated by d can be eliminated from this process. SDD-1 is mainly concerned with communication cost optimization, but the data of Q remains in Q, where the query is executed.

It is assumed that all other relations (i.e., PAUX, KF1, and KF2) are all stored and available in Q. Therefore if the data needed from sites S and P are transmitted into Q, the query is executed by the relational database technique and the result will appear as in Table 13. [3]

As this example has illustrated the application of optimization technique on the relational data-knowledge model proposed by Li is very natural. However one may still be skeptical about the above example,

13

Semijoins	Cost	Effect	Benefit
S <S# = S#]Q	1000	C(S.S#) = C(S) * 20	180*4
P <P# = P#]Q	1000	C(P.P#) = C(P) * 200	1800*4
(b) Q <S# = S#]S	200	C(Q.S#) = 20, C(Q)*2K C(Q.P#) = 1K, C(Q.Price)=1K C(Q.qulty) = 3	98K*4
Q <P# = P#]P	2000	C(Q.P#) = 200, C(Q)=20K C(Q.S#)=1K C(Q.Price)=200 C(Q.qulty) = 3	80K*4

Semijoins	Cost	Effect	Benefit
S <S# = S#]Q	20	C(S.S#) = C(S) = 20	180*4
(c) P <P# = P#]Q	1000	C(P.P#) = C(P) = 200	1800*4
Q <S# = S#]S	200	None	None
Q <P# = P#]P	2000	C(Q.P#)=200 C(Q)=400 C(Q.S#)=20 C(Q.Price)=200 C(Q.qulty) = 3	1600*4

Semijoins	Cost	Effect	Benefit
S <S# = S#]Q	20	C(S.S#) = C(S) = 20	180*4
P <P# = P#]Q	1000	None	None
(d) Q <S# = S#]S	200	None	None
Q <P# = P#]P	200	C(Q.P#)=200 C(Q)=400 C(Q.S#)=20 C(Q. qulty)=3	1600*4

Semijoins	Cost	Effect	Benefit
S <S# = S#]Q	20	C(S.S#) = C(S) = 20	180*4
P <P# = P#]Q	1000	None	None
(e) Q <S# = S#]S	200	None	None
Q <P# = P#]P	200	None	None

Figure 2
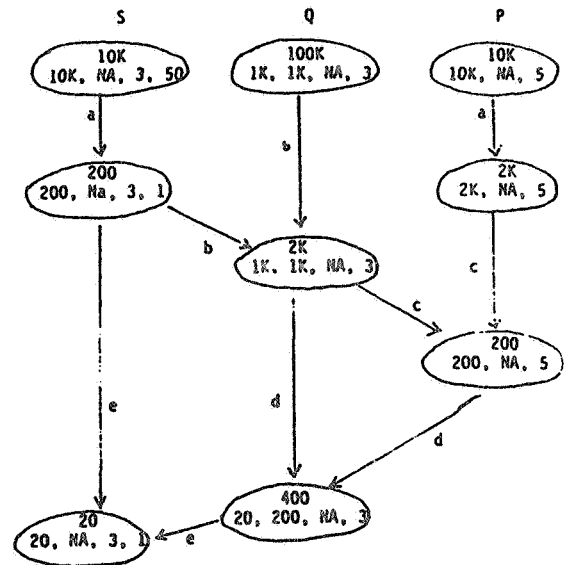
Fig. 3. The flowgraph of reductions of S, Q, and P.
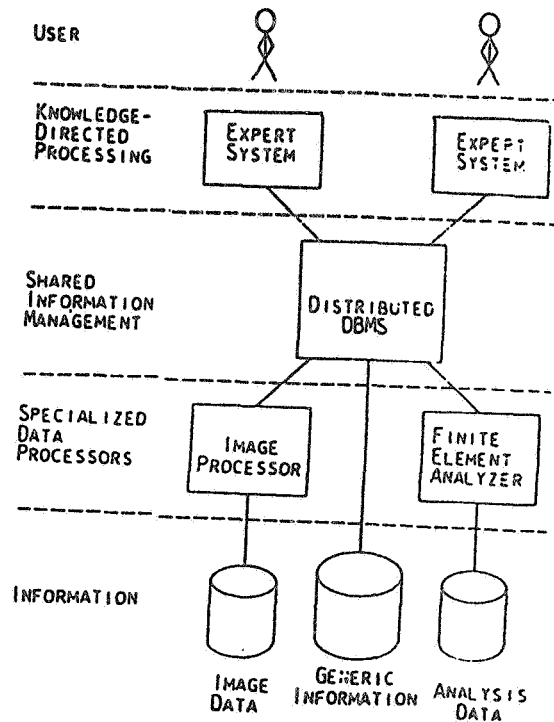
Fig. 4. Expert Database System Architecture

because its query did not involve any reduction of the knowledge files. It was pointed out that the main question is the usefulness of such queries. Indeed it may be useful to make queries such as:

List all the supplies from the suppliers who have received first priority.

But, without any information on what proportion of supplies or suppliers fall into this category, a priori the cost and benefit of the reduction can not be determined to allow further analysis. Also, it is probably true that the knowledge base (compiled information from the experts) are available in all potential sites where queries are executed and hence the reduction of the size of the knowledge files, which may be implicit in query processing, may not reduce communication costs.

As another example consider the query:

List the suppliers who provide parts whose price level is average. In this case, again, if there are three price levels for each part listed in the table and if the prices of the suppliers' parts are uniformly distributed into low, average, and high ranges, then the proportion of the reduction may be estimated to 1/3. But, if this data file is available in all sites or at the site where the query is executed then its reduction does not contribute to any communication cost reductions.

Finally, one should remember that SDD-1 is a distributed relational data-base system and it can handle any queries supported by the relations stored at its sites. Query planning in SDD-1 is, however, centralized. This implies that the knowledge files, though treated as relations, must be available for making queries. Therefore their availability in the central site if any, or in all sites used for query execution will reduce the communication costs of the system. The trade-off between the storage cost and communication cost is important in the design of the system architecture.

VI. A Word About Architecture and Some Concluding Remarks

A typical expert database system has an architecture similar to Fig. 4. [6]

As J. M. Smith has explained: "...An EDS (expert database system) is essentially the composition of two powerful search engines. One engine will be searching knowledge rules to solve an applicaton problem and, in so doing, generating queries over shared information. The other engine will be searching the shared information to answer the queries. The result could be a multiplicative explosion in processing time."

"Performance should be the criterion for the dispersion of functionality between ES's and DBMS. A function should be allocated to the system where it can be executed most efficiently. The two primary performance concerns are the secondary storage search patterns of the DBMS and the search cycle of an ES." [6]

Smith also has discussed the question of whether the DBMS can take over some of the loads of the Expert Systems (ES).

Looking at the EDS architecture, Li's approach will reduce the load of the Expert System, because the knowledge files are transformed into relations and hence are stored and manipulated by DBMS.

Li has made another point:

"most of current logic-based programs, as Micro-Prolog, are recursive search processes which interpret each fact or rule one by one

15

following its order in the program. Many unrelated facts and rules needed to be executed under the same process. Therefore the size of facts or rules affect prolog's executing time definitely. However, in relational database system, numerous large data or data-type knowledge is decomposed into many normalized relations under the consideration of data dependency. A RDBMS is then convenient to search a specific relation or to join some relative relations directly based on their indexed primary key(s)" [3].

Li later adds that for the decision problem with rules which are translatable into relations, his method is one of the best systems to be used.

If one accepts that most of the human (expert) knowledge may be expressed by production rules and transformed into relations, then, the reduction in the expert systems load will be large enough to encourage new research on this subject.

As conclusion, this paper was motivated by both the potentials of distributed database systems in concurrent and parallel execution, as well as by the growing literature on expert database systems. The limited time and resources availed to this paper has not allowed the empirical examination of the proposed approach in a distributed database environment. Nevertheless, based on the empirical test rendered by Li and the example used in this paper, a case has been made which illustrates the potential advantage of the proposed method in developing a relational distributed data-knowledge base system.

At this stage of the study, it is difficult to discuss the potential problems which may arise in implementing Li's approach in a full-fledged distributed data-knowledge systems. Nonetheless, it seems proper to assume the merits of such implementation will exceed the cost of its development. In cases where large knowledge files exist the translation of production rules into relational knowledge files may be automated. In addition, authentication and maintenance of knowledge files may require higher level of integrity and security.

REFERENCES

1. Bernstein, P. A., et al., "Query Processing in a System for Distributed Databases (SDD-1)," ACM Transactions on Database Systems, Vol. 6, No. 4, Dec. 1981, pp. 602-625.

2. Korth, Henry F., and Silberschatz, Abrahams, "Database System Concepts," MCGraw-Hill, 1986, pp. 475-477.

3. Li, Han-lin, "To Develop a Data-knowledge Base Management System by Utilizing Relational Database Management System, in Proceedings of Applications of Artificial Intelligence IV, (A Conference on 15-16 April 1986 in Insbruch, Austria).

4. Mohan, C., "Recent Advances in Distributed Data Base Management," IEEE Computer Society Press, 1984, pp. 7-13.

5. Rothnie, J. B., Jr., et al., "Introduction to a System for Distributed Databases (SDD-1)," ACM Transaction on Database Systems, Vol. 5, No. 1, March 1980 pages 1-17.

6. Smith, J. M., "Expert Database System," (edited by Larry Kerschberg) Addison-Wesley Publishing Company, A. Benjamin, Commings Publication, 1986, pp. 5-9.

Hologram Representation of Design Data in an Expert System Knowledge Base

S.G.SHIVA
COMPUTER SCIENCE
   DEPARTMENT
UNIVERSITY OF ALABAMA
HUNTSVILLE, AL 35899

PETER F. KLON
BOEING MILITARY
   AIRPLANE CO.
HUNTSVILLE, AL 35807

Abstract

This paper presents a novel representational scheme for design object descriptions. An abstract notion of modules and signals is developed as a conceptual foundation for the scheme. This abstraction relates the objects to the meaning of system descriptions. Anchored on this abstraction, a representational model which incorporates dynamic semantics for these objects is presented. This representational model is called a hologram scheme since it represents dual level information, namely, structural and semantic. The benefits of this scheme are presented.

I. INTRODUCTION

The clear unambiguous communication of design data and specifications is crucial to systems design which encompasses a multitude of disciplines. Today this need appears particularly apparent in electronic systems design. The evolution of electronic component technology from simple single device units to complex very large scale integrated (VLSI) circuits has greatly increased the design options available to computer engineers. This factor has led to the development and use of design automation tools such as silicon compilers, computer aided design (CAD) tools, and expert system based design aids. Each of these tools seeks to automate the routine tasks faced by the computer system engineer while freeing him to focus on strategic design issues. [1, 2] The integration of these tools is closely tied to the communication of design data among them.

Design data representation provides a medium for communicating and exchanging design data among the system tools. The development of design data representation must address the following issues:

1) How is the meaning of the design data represented?

2) How should this representation be interpreted?

3) Is the meaning predefined (static) or is it developed by the manner in which primitive components of the representation combine to form a definition (dynamic)?

18

This paper extends and expands upon the work of other researchers [3-9] to develop a generic design representation scheme that is free from static semantics and is thus adaptable to any design methodology and strategy.

In general, knowledge representations can be classified into seven distinct groups: logic, procedural representation, semantic networks, production systems, direct representation, frames, and scripts [3, 10]. A discussion of their applications and limitations can be found in [3, 10]. Figure 1 summarizes the applications and shortfalls of these representations.

None of the seven representational schemes by themselves are suitable for representing design data which reflect dynamic semantics, complex multidimensional relations and components, and modularity. To accommodate design data a combination of the basic schemes is required. For example, Green's prototypes for UHESS [3, 10] merge frames and production rules into prototypes in a manner similar to that of CENTAUR [9]. These prototypes provide modularity, a structure for logically organizing information, and a mechanism for organizing and collecting heuristic rules into related groupings. But a means to build meanings dynamically is not provided, nor are complex relations among objects supported. We seek, therefore, to develop a representational scheme that is suitable for representing design data. This scheme also will be based upon a combination of several selections from the seven basic representations.

We will develop a conceptual basis for a new representational scheme, the hologram, for design specifications and will introduce and illustrate the scheme itself. Section II analyzes the abstractions of design data in terms of the notion of module and signal abstractions. Modules represent functions, actions, activities, or geometrical objects. Signals represent carriers of information between modules. Modules interface through ports; and signals are associated with these ports for the transmission of information between modules. Hierarchical semantic structures are propagated through modules; and semantics of the connectivity between modules are propagated through signals. Section III introduces the hologram scheme which combines the features of prototypes and semantic networks into a single structure suitable for use in an expert system knowledge base. The hologram holds both design data and design knowledge and provides a generic tool for dealing with the design problem in various problem domains. The use of this scheme will be illustrated.

II. ABSTRACTION OF DESIGN DATA OBJECTS

In this section, we lay the conceptual foundation for our hologram scheme. An abstraction of design objects and the interfaces between them form the cornerstone of this foundation. We begin with a consideration of design objects.

Design objects may be divided into two classes, modules and signals. These classes flow from an analysis of the design

20

process. Designs typically reflect a hierarchical modular structure and the relationships among the modular elements. The modular structure provides a means to manage the design complexity. The modules represent functions, activities, or real objects. For example, in electronics modules represent circuits and circuit functions; in optics they represent optical components and optical transformations; in control system design they represent filters, limiters, summing junctions, etc. The relationships among modules on a particular hierarchy level are described in terms of their interfaces with one another.

Module interfaces consist of points of input and output. The points at which the inputting and outputting of data take place are called ports. These ports provide an interface mechanism between the module and its external environment. Signals provide a means for passing information from an output port of one module to the input port of another. From this point of view, signals are informational packets that possess the characteristics of a value and a meaning. The meaning is given by the type definition of the signal. This usage of signal terminology then encompasses any element that carries information such as types, attributes, and comments.

The semantics of a module provide a definition of the function or activity performed by the module. If we consider the ALU module shown in Figure 2, we find the meaning of the module given in terms of several submodules and the interconnection of signals among them. For this example the submodules are SHIFTER, TWOS_COMP, and EIGHT_BIT_ADDER. Each submodule in turn can be

21

similarly analyzed and can be described in terms of a hardware description language (HDL) specification. This process can be continued until the lowest level of the hierarchy is reached. The meaning propagates through the design hierarchy upward from the bottom to the top. The meanings of both the submodules and the interconnecting signals form the complete module meaning.

III. HOLOGRAM REPRESENTATION OF DESIGN DATA

This section applies the module and signal design data abstraction to knowledge base representation of the design data, develops the hologram representational scheme, illustrates its use, and discusses its advantages.

III.1. The Hologram.

We now seek to find a convenient scheme for representing modules and signals in a knowledge base that reflects the features of a substitution hierarchy. The requirements for such a representational scheme follows:

1) Module hierarchy must be displayed.

2) Terminal modules must have predefined static meanings.

3) Generic modules must be available for use as templates in growing a hierarchy tree.

4) Port structures must be provided in modules to allow for communication of signals between modules.

22

5) A hierarchy of signal semantics must be provided for developing signal meanings analogous to those of modules.

6) Terminal signal elements must have predefined static meanings.

7) Signal semantic hierarchy must be supportive of module hierarchy.

8) The representational scheme should focus on module hierarchy.

The first three requirements support the development of a meaningful module semantic hierarchy. The fourth requirement provides for the interface between modules. The fifth and sixth requirements provide for the development of signal meanings. The seventh and eighth requirements establish the priority relationship between modules and signals. Modules are primary since they specify functions performed on signals and hold the primary interest of the designer.

These representational requirements suggest the use of a structured representational scheme that reflects relations among structured elements. As we have seen in Section I, frames, prototypes, and semantic networks are such schemes. Frames are complex heterogeneous structures that reflect knowledge in a predefined manner. Each data element in a frame is a slot which identifies data values to be filled-in. Slots may reference other frames and thus can build a frame hierarchy. A prototype is a combination of a frame and a set

23

of relevant production rules. The production rules may provide a heuristic for filling-in slot data values. Semantic networks are graph structures in which nodes represent objects or concepts and arcs represent relationships between nodes.

A marriage of prototypes and semantic network structures provides a mechanism for satisfying the representational requirements. Prototypes are suitable for representing the hierarchical structure of modules. The semantic networks are suitable for representing meanings. Since the semantic networks do not reflect an ordering of knowledge nor do they easily propagate an inheritance of knowledge, they alone are not sufficient to represent modules and signals. Prototypes are suitable for representing modules which are hierarchical. We often find, however, that signals may possess a hierarchical structure due to a complex type specification. Prototypes and semantic networks are merged into a two dimensional structure that reflects knowledge ordering, heuristic direction, hierarchy, and object relationships for both modules and signals.

This perspective requires two kinds of prototypes, one for modules and one for signals. Module prototypes reflect hierarchical relationships and signal prototypes reflect semantic relationships and structures. Figures 3 and 4 present prototype models for modules and signals, respectively. Several types of slots are common to both kinds of prototypes. The header slots provide cataloging and revision information. The type slot identifies the prototype as either a module or a signal. The

24

modifier slot provides a usage indication for the prototype. These usage indications are predefined. They are germane to the particular problem domain in which they are used. For example, usage indications for the domain of electronic circuit design may include the terms interface, body, library, generic, etc. The condition slots hold isa_attribute relations which provide attributed information that help support and develop the semantic meaning of the prototype. These slots may be denoted by lists of identifier-prototype pairs which develop the meaning of the attribute. For the ALU example, the list (8, MA., MAX) reflects the attribute condition that the bus can handle a maximum current of eight milliamperes. The identifiers MA. and MAX reference their defining signal prototypes. Heuristic slots provide information in a production rule form. These production rules may relate to the usage, applicability, and limitations of the object being defined by the prototype.

Several slot types are peculiar to the module prototype. Input and output port slots identify the signals associated with the input and output ports, respectively. These slots hold isa_type relations denoted by an identifier-prototype pair lists which provide the associated signal typing and meaning information for the corresponding port. The pair (A, BUS) is an entry for an input port slot of the ALU example, Figure 2, where BUS is the type name for the eight bit bus structure. The local slots identify signals confined to internal usage within the module. Such local signals connect

25

the output of one component to one or more component inputs without a connection to the module input or output ports. Signal Y in Figure 2 is an example of a local signal. The function slot may provide a description of the function intended to be performed by the module. Information for explanations and justifications are also included in this slot. The element assignment slots hold isa_element relations which pair the components (circuit elements) that form the next lower layer of the structural hierarchy with modules that define the components and their function. These slot entries are denoted by circuit element and identifier-prototype pairs which develop the structural hierarchy. For the ALU example, (1, EIGHT_BIT_ADDER) is one of the element assignments. The netlist slots define the interconnection of ports among the submodules by isa_net_member relations. To denote these relations for each signal name, components and ports are paired together and listed with other identifier-port pairs to form a named list. The list (A; (1,1), (2,2)) reflects the connection of port 1 of the EIGHT_BIT_ADDER with port 2 of the SHIFTER and the association of the net with signal A for the ALU example. This interconnection network name is implicitly or explicitly typed. Implicit types are determined from the port typing. Explicit types are used to avoid ambiguities and are defined in the condition and heuristic slots.

Two slot types are peculiar to signal prototypes. First, information slots provide declarative descriptions of the signal and its use. Information for use in explanation and

26

justification may also be accommodated by these slots. Second, the descriptor slots identify the typing structure by isa_type relations. For the ALU example, the BUS can be given the descriptor (8, BIT). The identifier of each element of this structure is paired with a prototype that further develops its meaning.

Terminal elements of these prototype structures are predefined. Such terminal elements are also described by prototypes although no prototypes are referenced by them. Only the header, type, information, function, and heuristic slots hold information; the other slots are null. In addition, the modifier slot entry is filled-in with "terminal." The signal BIT is an example of a terminal signal.

Figure 5 presents abbreviated module and signal prototypes for the ALU example of Figure 2. For the ALU module, port names are paired with their typing identifier which relates to its signal prototype. Element assignments pair a circuit element number with the name of the component. The component relates to its module prototype. The netlists are listings of pairs of circuit element number and its port number. The element port pairs are chained together and anchored at the identifier of the signal that passes through them. The heuristics slot references usage rules. For the BUS signal the descriptor describes the bus as eight BIT. The type identifier itself references the BIT signal structure which is a predefined terminal. Each module and signal structure is similarly expanded until a terminal structure is reached.

27

This expansion process produces a network structure of module and signal prototypes. Figure 6 presents a graph of this network structure for the ALU example. The solid lines represent relations to modules and the broken lines relations to signals.

We find a name for this prototype-semantic network structure by borrowing a term from the discipline of optics. In optics a recording technique which represents amplitude and phase information of an object is called holography. The recording medium is a holographic plate. The image reconstructed from the recording is a hologram. Our knowledge base maintains structural and semantic information. The structural information is analogous to amplitude, and semantic information is analogous to phase. Both are needed to reconstruct the complete "picture." Thus, we name our knowledge base structure "hologram." To further recognize the appropriateness of this terminology consider the etymology for the word. Hologram comes from holo and gram. Holo comes from the Greek holos meaning whole. Gram means something recorded. Thus our hologram is a complete recording of an entire object.

Modules and signals are represented by a hierarchical tree of module prototypes and a semantic network of signal prototypes that underlie each module prototype. This dual level prototype scheme is called a hologram. It is the dual level nature of holograms which combines the common usage of prototypes with a semantic network structure that forms the distinguishing feature of this knowledge base representation. The hologram provides a

28

means for defining the semantics of a designed object in a dynamic fashion. Only terminal modules and signals have static semantics.

III.2. Benefits of the Hologram.

The hologram representation of design data objects in a knowledge base component is superior to other representational schemes, such as frames, semantic networks, and production rules. The hologram provides modularity and a logical structured organization of knowledge that corresponds to the design process itself. A mechanism for dynamically creating new meanings is provided. This mechanism supports the synthesis of new design objects having new meanings. Heuristic information for guiding the usage of objects is localized to the objects to which they pertain. The hologram unifies structure, semantics, and heuristics.

This hologram scheme holds several advantages for knowledge base representation over pure production systems, frames and prototypes, and semantic networks. Pure production systems do not provide mechanisms for easily grouping related heterogeneous knowledge. In order to determine that some module X has a port A for output and ports B, C for input requires that a directed search be conducted among a sequence of production rules. This approach requires a pattern matching search process that is time consuming with large knowledge bases. A very large knowledge base of production rules

would be needed for fully describing a complex electronic system for which an expert system would be a significant design aid.

Frames provide a structured representational scheme for grouping design information relating to some module X. The predefined slots in the frame can provide information on module X directly. However, if we ask a question like "for what circuit applications is module X suitable?," the frame structure cannot provide the answer. A production rule system, on the other hand, can address the question provided the required rules are created. Now if we combine the production rules into slots in the frames, thus creating a prototype structure, then we can address this question.

If we now ask a question such as "What is the type of signal at the output ports?", we find that production rules and frames are all ill-equipped to address the question. First, the typing information is not likely to be unique to module X. The input port to whatever this output is connected must also have this type associated with it. Second, the information that defines this type must be unique. This requirement will ensure, for example, that if the signal has a complex bus structure then its definition will be consistent for all instances of its usage. With this example we are dealing with the semantics of the signal, i.e. its meaning. A semantic network is more suited to representing this semantic information. The module structure also can be represented in a semantic network. But the semantic network structure cannot

provide information to answer the question "for what circuit applications is module X suitable?"

The hologram structure described above combines the key features and advantages of the prototype and semantic network schemes. The hologram structure can be viewed as a generalized semantic network in which the nodes have a prototype structure. The hologram has slots which propagate semantic meaning, slots which hold data values internally, and slots for production rules for specialized knowledge tailored to that particular module or signal. Thus the hologram structure provides for the development of semantics of modules and signals, for rules about the usage and development of the surrounding module or signal, and for direct retrieval of unique characteristics of the module or signal. Figure 7 presents an illustration of the hologram network structure.

These advantages of the hologram culminate in its generic character. The dynamic semantics of the hologram are not tied to any particular design problem domain. This is the hologram's key strength. For as Mitchell [6] has noted:

> ...relatively few knowledge-based systems have been developed for design tasks, and as a field we have yet to produce a generic model of design tasks and generic frameworks for developing knowledge-based systems for design.

The hologram provides a generic framework for representation of design data in knowledge bases of design oriented expert systems. Design problem domains that manipulate objects which have some kind of interface with one another are

31

prime candidates for interpretation in terms of module and signal abstractions. The application of the hologram to these domains is the subject of a future paper.

IV. CONCLUSION

Design data divide into two classes, modules and signals. Modules represent an activity or operation. Signals are carriers of information. Ports are module locations through which signals may pass from one module to another. Modules and signals are the chief carriers of semantic meaning in a design description. Modules and signals are the design objects. The meaning of modules and signals are propagated from terminal modules and signals upward through the hierarchy tree.

Modules and signals are represented by a hierarchical tree of module prototypes and a semantic network of signal prototypes that underlie each module prototype. This dual level prototype scheme we call hologram. It is the dual level nature of holograms which combines the common usage of prototypes with a semantic network structure that forms the distinguishing feature of this knowledge base representation. The hologram provides a means for defining the semantics of a designed object in a dynamic fashion. Only terminal modules and signals have static semantics.

The hologram representation of design data objects in a knowledge base component is superior to other representational schemes, such as frames, semantic networks, and production rules. The hologram provides modularity and a logical structured organization of knowledge that corresponds to the design process

32

itself. A mechanism for dynamically creating new meanings is provided. This mechanism supports the synthesis of new design objects having new meanings. Heuristic information for guiding the usage of objects is localized to the objects to which they pertain. The hologram unifies structure, semantics, and heuristics and provides a generic framework for representing design data in various problem domains, such as electronic systems, optical systems, control systems, and signal processing systems.

We are continuing our research into the hologram representation of design objects in knowledge bases for application to various problem domains. Our investigation is including study of the interaction of holograms with the synthesis process and users.

REPRESENTATION	APPLICATION	SHORTFALLS
LOGIC (PREDICATE CALCULUS)	MODULAR FLEXIBLE	UNORDERED. REQUIRES PRECISE INTERPRETATION.
PROCEDURAL	CREATES NEW FACTS	DOES NOT ADDRESS OBJECT DESCRIPTIONS.
SEMANTIC NETWORKS	TAXONOMIES	INEFFICIENT TO REPRESENT A COLLECTION OF SIMILARLY STRUCTURED OBJECTS SINCE REPETITIOUS TRAVERSAL OF SIMILAR NETWORK STRUCTURES IS REQUIRED TO OBTAIN OBJECT INFORMATION.
PRODUCTION SYSTEMS (ANTECEDENT- CONSEQUENT)	HEURISTICS	UNORDERED. AWKWARD FOR STRUCTURED KNOWLEDGE.
DIRECT	GRAPHIC DISPLAYS	DIFFICULT TO USE. CANNOT CLEARLY REFLECT MULTIDIMENSIONALITIES OF COMPLEX OBJECTS HAVING MULTIDIMENSIONAL STRUCTURES.
FRAMES	OBJECTS KNOWLEDGE STRUCTURING MODULAR	SEMANTICS ARE STATIC.
SCRIPTS	EVENT SEQUENCES KNOWLEDGE STRUCTURING MODULAR	SEMANTICS ARE STATIC.

Figure 1. Seven groupings of knowledge representation.
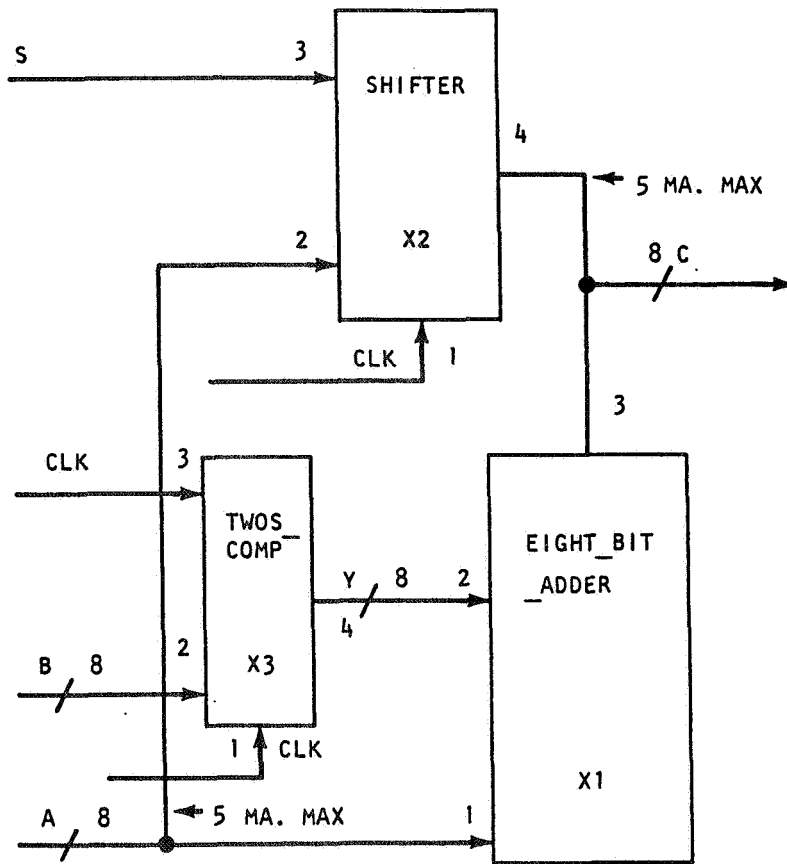
Figure 2.   Example ALU module schematic.

Header: Name
                     Author
                     Date
                     Revision date
                     Revision number
Type: Module
Modifier: Usage indication
Input ports: Signal prototype list
Output ports: Signal prototype list
Local ports: Signal prototype list
Function: Descriptive (May key to explanations)
Element assignments: Lists of circuit element
                       and module pairs
Netlists: Parameter interconnection list
Conditions: Signal prototype list for attributed
                     information
Heuristics: Production rules

Figure 3.   Module Prototype Model.

35

```
Header:          Name
                 Author
                 Date
                 Revision date
                 Revision number
Type:            Signal
Modifier:        Usage indication
Information:     Descriptive (may key to explanations)
Descriptor:      Signal type prototype list
Conditions:      Signal prototype list for attributed
                 information
Heuristics:      Production rules
```

Figure 4.   Signal Prototype Model.

```
NAME:  ALU                      NAME:  BUS
TYPE:  MODULE                   TYPE:  SIGNAL
IN:  A,  BUS                    DESCRIPTOR:  8,  BIT
     B,  BUS                    CONDITIONS:
     S,  BIT                            5,  MA.,  MAX
     CLK,  BIT                  HEURISTICS:  RULE062
     COMP,  BIT                              RULE065
OUT: C,  BUS
LOCAL:  Y,  BUS                 NAME:  BIT
ELEMENT ASSIGNMENTS:           TYPE:  SIGNAL
     1,  EIGHT_BIT_ADDER       MODIFIER:  TERMINAL
     2,  SHIFTER
     3,  TWOS_COMP             NAME:  EIGHT_BIT_ADDER
NETLISTS:                      TYPE:  MODULE
     A;  (1,1),  (2,2)         ...
     B;  (3,2)
     S;  (2,3)                 NAME:  SHIFTER
     CLK;  (2,1),  (3,1)       TYPE:  MODULE
     COMP;  (3,3)              ...
     C;  (1,3),  (2,4)
     Y;  (1,2),  (3,4)         NAME:  TWOS_COMP
HEURISTICS:  RULE071           TYPE:  MODULE
             RULE076          ...
```

Figure 5.   Abbreviated hologram prototypes for ALU example.
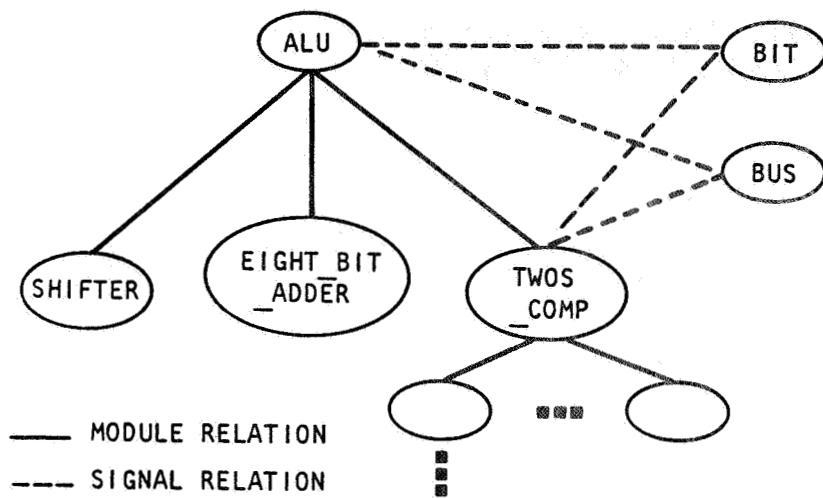
Figure 6. Abbreviated hologram network for ALU example.
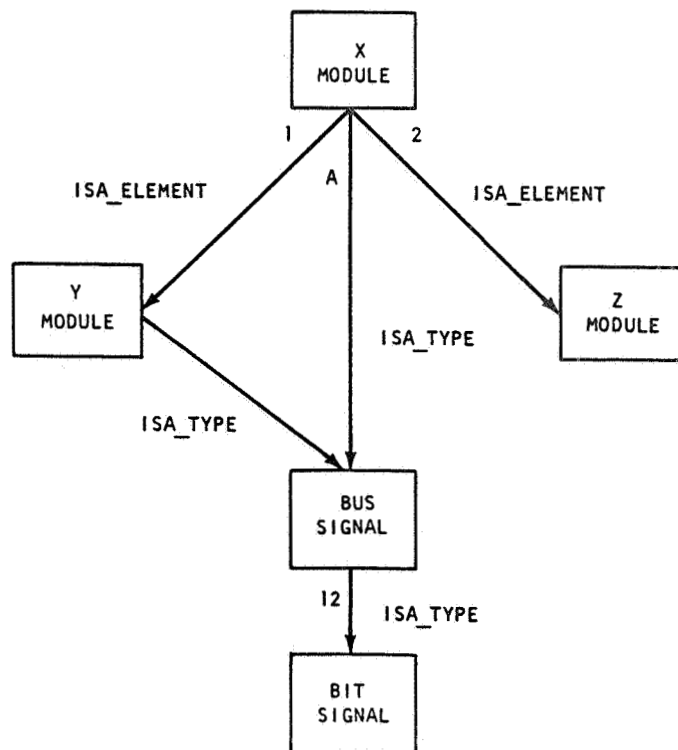
Figure 7. Hologram description of module X,Y,Z.

37

BIBLIOGRAPHY

[1]   Shiva, S.G. "Automatic Hardware Synthesis,"
       Proceedings of the IEEE, Vol. 71, No. 1, January
       1983, pg 76-87.

[2]   Muroga, S. VLSI System Design, Wiley-
       Interscience, 1982.

[3]   Green, C.R. Development of an Expert Hardware
       Synthesis System, doctoral dissertation,
       University of Alabama in Huntsville, AL.
       December 1985.

[4]   Kawato, N., Uehara, T., Hirose, S., Saito, T.
       "An   Interactive Logic Synthesis System Based
       upon AI   Techniques,"   19th Design Automation
       Conference. 1982.   Pg. 858-864.

[5]   Kowalski, T.J., Thomas, D.E. "The VLSI Design
       Automation Assistant: Prototype System." 20th
       Design Automation Conference. 1983. Pg. 479-
       483.

[6]   Mitchell, T.M., Steinberg, L.I., Shulman, J.S.,
       "A Knowledge Approach to Design," Pattern
       Analysis and Machine Intelligence, Vol. PAMT-7,
       No. 5, September 1985, pg 502-510.

[7]   Steinberg, L.I., Mitchell, T.M. "A Knowledge
       Based Approach to VLSI CAD. The Redesign
       System." 21st Design Automation Conference.
       1984. Pg. 412-418.

[8]   Park, H.-S., Kabat, W.C. "KnowPLACE --
       Knowledge-Based Placement of PCBs." 5th
       International Workshop, Expert Systems and their
       Applications. 1985. Pg. 1283-1294.

[9]   Aikins, Janice S.   Prototypes and Production
       Rules: A Knowledge Representation for Computer
       Consultation.   Doctoral Dissertation, Stanford
       University, Computer Science Department, 1980.

[10]   Klon, P.F., On Interfacing HDL to Knowledge Bases,
       doctoral dissertation, University of Alabama in
       Huntsville, AL. May 1986.

AN EXPERT SYSTEM
FOR RELIABILITY MODELING

ERNST GOSS
VANI SUNDARAIYER
ALAN WHITTEN

UNIVERITY OF ALABAMA IN HUNTSVILLE

AN EXPERT SYSTEM
FOR RELIABILITY MODELING

Introduction

Reliability researchers have traditionally used univariate models such as the exponential, Weibull, etc. to explain or predict failure in a device. Unfortunately the assumptions necessary to validate their use are quite restrictive and vitiate against their use in many cases. This study advances an alternative technique that provides a multivariate approach. The technique, logit regression, in combination with decision support software written by the authors, provides an expert system that will accomodate reliability modeling based upon a multivariate historical data base. This system will, based upon the "expert's" predicted usage of the device over the next mission or task and the "expert's" predetermined acceptable risk level, dictate whether the unit should be replaced.

Expert Systems
and Statistical Analysis

> As the AI representations become deeper and represent their roots, they will always be found to rest on measurement, which always has a random component, so there is a clear need for statistics in AI. (Gale, 1986, p.3)

Gale concludes that what the statisticians need to learn, the AI researchers know well and vice versa (Gale, 1986). It is contended that due to poor communications or interaction between the two groups, AI researchers often obtain less than robust results or obtain results that omit the critical random nature of data. This study attempts to bridge the gap between the sometimes disparate areas and provide an expert system that incorporates sound statistical techniques.

Nelder (1977) was the earliest to call for intelligence in statistical software. However, in general, AI work has paid insufficient attention to the problem of error in the input symbolic data. In other words, AI researchers assume that the variables are measured without error and represent population data rather than sample data. In the future, reasoning in AI must take into account the uncertainty of empirical relationships (Charniak, 1983). Furthermore, others argue that techniques that have been used to deal with uncertainty in expert systems have not always made use of the most appropriate statistical technique (Spiegelhalter and Knill-Jones, 1984).

According to Hahn (1985), the probability of success in developing a statistical expert system is greatly increased when the system is devoted to specialized applications. Further he recommends product-life analysis as an appropriate candidate. "For example, the end result of a product-life data analysis might be a number of different reliability projections based on; different assumed models, each of which seems reasonable on the basis of the available data." (Hahn, 1985, p. 5)

The objective of this study is not only to offer an expert system for reliability modeling but to provide a system that integrates statistical methods which are congruent with sound statistical theory.

Univariate Techniques

Engineering reliability measurements assume that "....reliability is the probability that a device will satisfactorily perform its specified function for a specified period of time under a given set of operating conditions (Naresky, 8-141)." Thus past studies assume that reliability or failure rates are time dependent measures of quality. In fact reliability is often defined as a "....time-dependent measure of quality (Naresky, p. 8-141)."

Of course the key to this analysis is the caveat "...under a given set of operating conditions." This requirement appears to be unrealistic in many evironments. This one dimensional analysis appears to omit many important factors which affect reliability but which are not captured by the variable, time.

For example, operating temperature, voltage fluctuations, maintenance expenditures and upkeep, number of starts, etc. will ordinarily affect system or unit reliability. The omission of relevant variables that affect failure rates produces a biased measurement of the impact of the included variable time upon failure rates. Thus there appears to be a need for alternative approaches under most operating environments.

Multivariate Techniques

The implicit assumption of homogeneity within the population, required by most operational models, appears to be untenable in most cases. Thus, it is suggested that in order to provide more realistic estimates of the probability of failure or time to failure, multivariate methods should be employed. Multiple regression is one recommended option (Bain p. 401). This method reduces to the determination of a least squares poloynomial fit to a set of data points. An advantage of this technique is that it is applicable to both censored and complete data.

Both the exponential and Weibull models can be generalized to regression models by allowing the failure rate to be a function of covariates (explanatory variables). However, multiple regression estimation provides no assurance that estimated probabilities will lie between zero and one. Thus, the regression may lead to nonsensical results, e.g. negative estimated probabilities or predicted probabilities greater than one. Of course, the dependent variable could be constrained to take on a value between zero and one. However, this procedure produces kinks at the end points of the distribution and there is no guarantee that the results will be unbiased (Pindyck and Rubinfield 1981, p. 241).

To circumvent these problems, estimation of these functions has increasingly employed alternative techniques. The purpose of this study is to integrate one of these techniques, logit analysis, into an expert system that will provide "expert" assistance in the determination of unit reliability. This technique has been employed successfully in past medical expert systems that have used probabilistic models to predict the existence of disease (Spiegelhalter and Knill-Jones, 1984).

This technique is particularly useful in those cases where the determination of risk over a time period is more valuable than the estimated time to failure. In the case where redundant or backup systems are provided, this is the most useful information since researchers are interested in the quantitative measurement of multiple failures on a particular mission or test.

41

Methodology

It can be demonstrated that the log-likelihood function is

$$\ln L = \sum_{i=1}^{T} \left[y_i \ln\{F(\underline{x}_i'\underline{\beta})\} + (1-y_i) \ln\{1-F(\underline{x}_i'\underline{\beta})\} \right] \tag{1}$$

Where:

T = No. of observations and $\underline{x}_i$ = explanatory variables;
y_i = a binary response variable which takes a value of 1 for failure and 0 for no failure

and where the cumulative density function (CDF) for the logistic variable is given by:

$$F(t) = \frac{1}{1 + \exp(-t)} \tag{2}$$

Since the maximum likelihood estimator of β cannot be obtained in a closed form, a numerical, iterative procedure must be used to obtain the maximum likelihood estimates. We are employing the Newton-Raphson method here. The recursive relation obtained by this approach is:

$$\underline{\tilde{\beta}}_{n+1} = \underline{\tilde{\beta}}_n - \left[\frac{\partial^2 \ln L}{\partial\underline{\beta}\,\partial\underline{\beta}'} \right]^{-1}_{\underline{\beta}=\underline{\tilde{\beta}}_n} \cdot \left[\frac{\partial \ln L}{\partial\underline{\beta}} \right]_{\underline{\beta}=\underline{\tilde{\beta}}_n} \tag{3}$$

where $\underline{\tilde{\beta}}_n$ is the n-th round estimate at which the matrix of second partials and the gradient vector are evaluated. For the logistic variable with the CDF given by (2), it can be shown that:

$$\frac{\partial \ln L}{\partial\underline{\beta}} = \sum_{i=1}^{T} \left[y_i F(-\underline{x}_i'\underline{\beta}) - (1-y_i)F(\underline{x}_i'\underline{\beta}) \right] \underline{x}_i$$

and   (4)

$$\frac{\partial^2 \ln L}{\partial\underline{\beta}\,\partial\underline{\beta}'} = -\sum_{i=1}^{T} f(\underline{x}_i'\underline{\beta})\underline{x}_i\underline{x}_i'$$

where $$f(t) = \frac{\exp(-t)}{\{1 + \exp(-t)\}^2}$$

is the probability density function of the logistic variable .

Using these derivatives and the recursive relation (3), we can obtain maximum likelihood estimators of β given some initial value of β . One can simply use the least squares estimates of β obtained by regressing y_i on the explanatory variables x_i as initial estimates. For the logit model, the choice of the initial estimates does not matter since it can be shown that the matrix of second partial derivatives is negative definite for all values of β . Consequently, the Newton-Raphson procedure will converge to the unique maximum likelihood estimates regardless of the initial estimates. Using OLS estimates as initial values simply hastens the convergence process.

The Data

The sample data was collected as an input into a reliability study for an aerospace manufacturing firm. This data represents historical data on transducers and contains both the number of starts and accumulated seconds on both failed and non-failed transducers. Much more data were supplied by the firm for the failed units. However, logit analysis does not allow missing values. Thus only data supplied for both failed and non-failed units could be used.

Empirical Results

A diagram of the decision support model is provided in Figure 1. The proprietary package Lotus 1-2-3 was used to implement the model.
The results of empirically testing the logit model in predicting transducer failure are displayed in Table 1. In each case the vector of measurements or explanatory variables will be:

```
STARTS  =    Engine hot-fire starts
TIME    =    Accumulated seconds  of hot-fire time exposure
TIMESQ  =    TIME*TIME
```

We expect both the number of starts and the accumulated seconds of hot-fire time exposure to positively affect the probability of failure. The variable TIMESQ was inserted since it is common for a unit to experience high rates of failure during the early and late time frames of usage. This of course suggests a quadratic relationship between the probability of failure and time.
Each variable is found to have a statistically significant impact on the probability of failure with each entering the estimated equation at the 90% confidence level or better. Additionally, the results indicate that the probability of a failure declines reaches a minimum and begins to increase for increases in time (parabolic). Thus holding the number of starts constant, one would expect to observe more failures during the early and late life of a transducer.

Conclusion

This study has provided a method for computing the probability of failure of a unit during the next usage or mission based sound statistical techniques. With the use of logit regression imbedded in Lotus 1-2-3 macros, this package computes the probability of unit failure based upon a historical data base. Using this computed probability of failure, the package then makes a replacement recommendation based upon the "expert's" acceptable risk parameters.

Table 1

Binary Logit Estimates of the Impacts of
Independent Variables on the
Probability of Failure of Transducer

Independent Variable	Coefficient	T-Ratio	Significance Level
STARTS	.12479	1.861	.063
TIME	-.68500E-03	- 2.392	.017
TIMESQ	.28984E-07	2.124	.034

LOG-LIKELIHOOD VALUE -69.159   (function converged after six iterations)

OPERATOR INPUT UNIT WHICH YOU WISH TO ANALYZE ?   3456

(Note: acccumulated
Starts and Time
= 8 and 5000)

OPERATOR INPUT ESTIMATED NUMBER OF STARTS NEXT MISSION? 5

OPERATOR INPUT ESTIMATED NUMBER OF SECONDS OF USAGE NEXT MISSION? 1500

OPERATOR INPUT THE CRITICALITY LEVEL (1, 1R, 2, 2R, 3)   3

PROBABILITY OF FAILURE OF THIS UNIT ON NEXT MISSION EQUALS 4.03%

BASED UPON YOUR INPUT ABOVE:

THIS UNIT SHOULD BE REPLACED

FIGURE 1
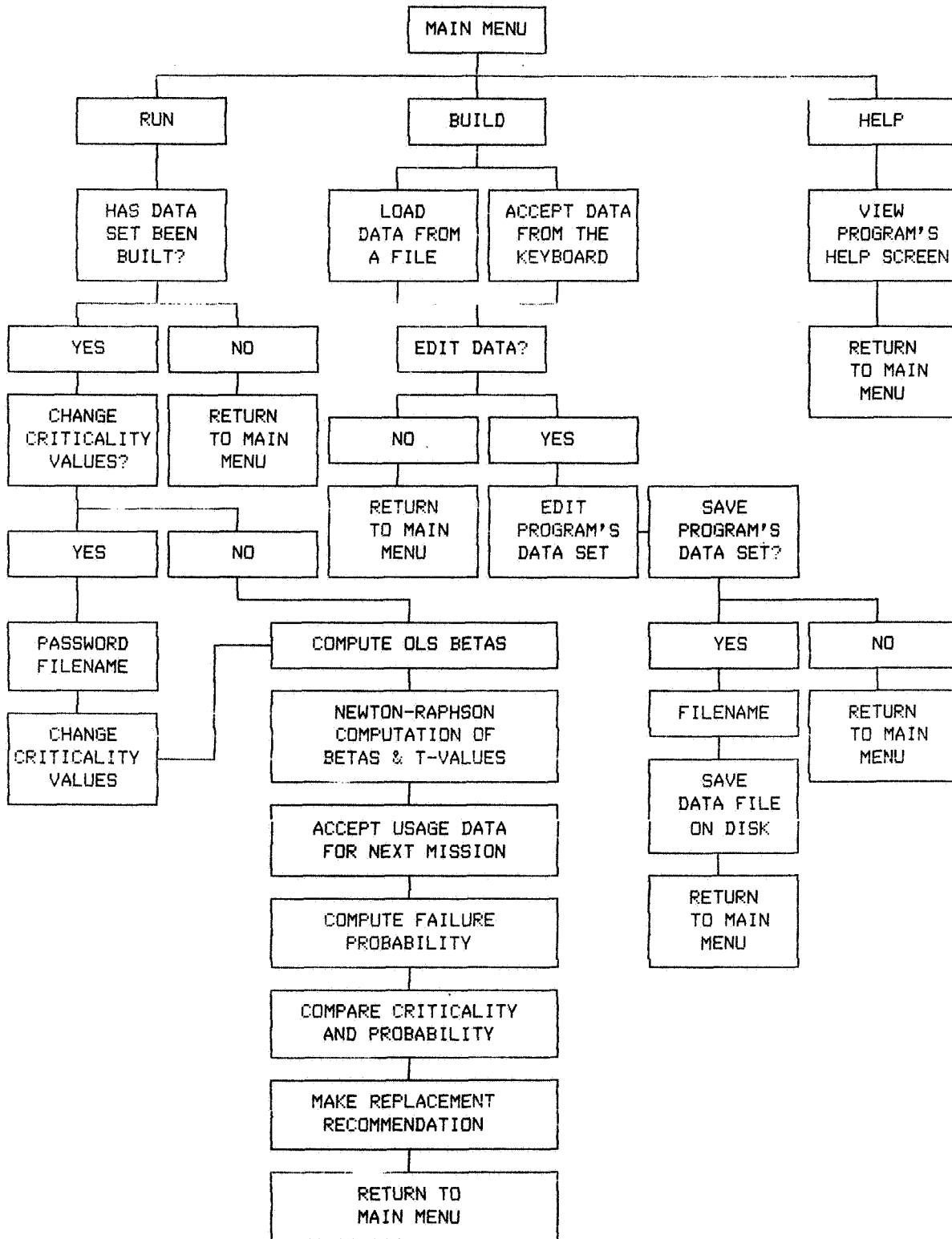
References

Bain, L. J. Statistical Analysis of Reliability and Life-Testing
  Models. Marcel Dekker, Inc., New York, 1978.

Charniak, E. "The Bayesian Basis of Common Sense Medical Diagnosis,"
  Proceedings of the National Conference on Artificial Intelligence,
  1983, pp. 70-73.

Gale, William. Artificial Intelligence & Statistics
  Addison-Wesley Co. 1986.

Gilbert, E. S. "On Discrimination Using Qualitative Variables,"
  Journal of the American Statistical Association Vol. 63, 1968,
  pp. 1399-1412.

Greene, W. H. LIMDEP. A statistical package for handling limited
  dependent variables, 1986.

Hahn, Gerald J. "More Intelligent Statistical Software and
  Statistical Expert Systems: Future Directions." The American
  Statistician, Feb. 1985 Vol. 39 (1), pp. 1-8.

Johnston, J. Econometric Methods McGraw Hill Book Co., Second
  Edition New York, 1972.

Kalbfleisch, J.D. and R.L. Prentice. The Statistical Analysis of
  Failure Time Data. John Wiley and Sons, New York, 1980.

Maynard, H. B. Industrial Engineering Handbook. Third Edition.
  McGraw Hill Book Co. New York, 1971.

Naresky, J. J. "Reliability Engineering," in Industrial
  Engineering Handbook, H.B. Maynard, Editor, McGraw-Hill Book
  Co. 1971, pp.8-140 to 8-167.

Nelder, J. A. "Intelligent programs, The next stage in statistical
  computing," in Recent Developments in Statistics, Barra (ed.)
  Amsterdam, North-Holland, 1977, pp. 79-86.

Norusis, Marija J. Advanced Statistics Guide-SPSS-X McGraw-Hill
  Book Co., New York, 1985.

Pindyck, R. S. and D.L. Rubinfeld. Econometric Models and
  Econometric Forecasts, McGraw-Hill Book Co., New York, Second
  Edition, 1981.

Spiegelhalter, David J. and Robin P. Knill-Jones. "Statistical and
  Knowledge-based Approaches to Clinical Decision-support Systems, with
  an Application to Gastroenterology," Journal Of the Royal Statistical
  Society, Vol. 147, 1984, pp. 35-77.

A SOFTWARE ENGINEERING APPROACH TO EXPERT SYSTEM DESIGN AND VERIFICATION

Daniel C. Bochsler
LinCom Corporation
18100 Upper Bay Road, Suite 208
Houston, Texas 77058

Mary Ann Goodwin
National Aeronautics & Space Administration
Johnson Space Center/FM2
Houston, Texas 77058

ABSTRACT

Software engineering design and verification methods for developing expert systems are not yet well defined. Integration of expert system technology into software production environments will require effective software engineering methodologies to support the entire life cycle of expert systems. Expert system designers are now implementing systems which are more sophisticated than many of the early prototypes, whose purpose was primarily to demonstrate the technology. Consequently, designers are becoming more aware of the need for efficient software engineering methods.

This paper discusses the software engineering methods used to design and verify an expert system, RENEX, which was developed for the NASA Johnson Space Center Mission Planning and Analysis Division. RENEX demonstrates autonomous rendezvous and proximity operations, including replanning trajectory events and subsystem fault detection, onboard a space vehicle during flight. The RENEX designers utilized a number of software engineering methodologies to deal with the complex problems inherent in this system. Many of these were adaptations from traditional software engineering methods.

This paper presents an overview of the methods utilized. Details of the verification process receive special emphasis. The benefits and weaknesses of the methods for supporting the development life cycle of expert systems are evaluated, and recommendations are made based on the overall experience with the methods.

INTRODUCTION

RENEX Overview

The RENEX program was developed for the NASA/Johnson Space Center Mission Planning and Analysis Division to demonstrate the concept of using an expert system to perform autonomous rendezvous and proximity operations onboard a space vehicle during flight. An expert system was implemented using the Automated Reasoning Tool (ART) Language, developed by Inference Corporation, on a Symbolics 3670 to simulate monitoring of vehicle hardware and software and planning of the trajectory. The flight software guidance, navigation, and control (GN&C) functions are simulated by a previously developed package, the Orbital Operations Simulator (OOS), which is programmed in C and runs on an HP9000. Relative motion data from OOS are sent to an Interactive Machines Incorporated (IMI) 500 graphics computer to provide animated graphics of onorbit motion between the active and target vehicles. This configuration is shown in figure 1. A voice interface can be connected to the Symbolics, and voice commands can be used to override the automatic trajectory planning capability.

The goal of the project was to develop technology which was transferrable to various types of trajectory control software to support future NASA projects, i.e., ground premission planning, ground real-time planning and monitoring, and onboard planning and monitoring. The rule base for the trajectory planning during rendezvous adapts Space Shuttle flight rules to a hypothetical Orbital Maneuvering Vehicle (OMV). The rule base for proximity operations is target-dependent. For example, if the user selects the Space Station as the target, the rule base is based on proposed guidelines for the Space Stations command and control zone operations.

Software Engineering Approach

The RENEX software engineering approach for the program needed to be flexible to augment technology transfer. The development team chose the traditional structured approach to software development used with conventional procedural languages as the approach most likely to provide this flexibility. Withing this structure, however, they utilized and benefitted from judicious prototyping.

The objectives, activities, and products of the definition, requirements, and design phases are summarized in Table 1 and discussed in detail in reference 1. The activities performed for the implementation and verification phases were related to the characteristics of previous phases. This is because the entire software engineering methodology has to considered as a whole, with activities and products of each phase complementing that of the others. In this section, we briefly revisit the results of

these early phases. The next section describes the goals, activities and results of the implementation and verification phases. Implmentation began earlier than verification and verification ended later than implmentation, but the two were inextricably interwoven. Finally, an evaluation is made and conclusions are presented.

RESULTS OF EARLY PHASES

Definition Phase

During the RENEX definition phase, we utilized the expert's time only as necessary to scope the system functions and develop a system architecture. This time familiarized the expert with the objectives of the expert system and gave him a context for understanding how the knowledge he provided would be used. The prototype system produced during this phase was based on the expert's preliminary input and supporting documentation. It gave the expert an opportunity to critique the approach to the expert system and created the foundation for communicating with the expert for the entire development.

Requirements Phase

A conscientious effort was made during the requirements phase to acquire detailed knowledge and to construct from it an accurate control structure (figure 2). Several knowledge forms were developed to assist in documenting the expert information. The payoffs were that later design efforts utilized a consistent control structure and that the expertise implemented was complete and correct. This reduced changes in later phases. Reference 2 describes the guidelines and requirements resulting from this effort.

Design Phase

The goal of software engineering is to create a design which "prevents" errors so that they do not have to be "treated" during verification. Design creates order which makes implementation and verification more manageable. For RENEX, allocating detailed information to the proper areas of the program was a matter of refining the control structure. For example, figure 2 represents the requirements phase control structure from which the more detailed design phase structure, figure 3, was derived. This refinement was accomplished through the use of the HIPO (Hierarchy, plus Input Process and Output) system analysis and design technique as the design specification approach. A sample of the HIPO documentation of the specifications is shown in figure 4. No major changes to the control structure of the system were required due to the consistency between successive

49

refinements of structural details. The RENEX HIPO software design is documented in reference 3.

IMPLEMENTATION AND VERIFICATION PHASES

It was assumed that the OOS and IMI software utilized were already verified and that the RENEX implmentation need only be concerned with verification of the expert system and the interface software created to connect it to the OOS.

The testing effort utilized the 5 levels of verification illustrated in figure 5, i.e.:

1) individual rules,
2) ordered and unordered groups of rules,
3) interface rules,
4) system tests, and
5) user tests.

Implementation activity followed this same order. This meant that verification could focus on testing new code as it was added.

The ART expert system shell has numerous features that assisted all levels of testing. Among them are capabilities for monitoring facts, rule executions, agendas, etc. associated with the inferencing process within RENEX. A test support requirement was identified and documented during the requirements phase and read as follows: "The expert system shall be capable of recording user interactions and appropriate internal process activity in order to provide process traceability and run repeatability. An option to enable or disable this feature shall also be provided to the user." An additional operational requirement not related to testing stated RENEX must execute in two modes. In the first mode, referred to as "closed loop," the OOS GN&C simulation provides feedback data. In the second mode, referred to as "open loop," the user provides the feedback data. The open loop mode allowed rapid, realistic testing of the system functions.

Rule Tests

Rule tests were designed to verify the accuracy and correctness of the individual rules. This was accomplished in two steps: inspection and compilation. Inspection consisted of comparing the rule code with a design specification statement in the design document. This permitted traceability from requirement to code and verified the intent of the rule. Compilation of each rule uncovered syntax errors and undefined patterns or functions. Individual rules could have been executed, but realistically there was little gain. Therefore execution was deferred to the lowest level of group testing.

50

Group Tests

Group testing is actually a hierarchical series of tests in which groups of rules are looked at, first, individually and, then, in combinations until finally the group is composed of the entire rule base. The control structure of RENEX provided well defined groups and a basis for combining them. Group tests were designed to verify the integration and functioning of the group components. Groups were defined as either ordered or unordered with respect to functionality. An ordered group is a one in which the components execute in a specific order with respect to each other, i.e., procedurally. In an unordered group the components execute opportunistically with respect to each other, i.e., randomly. Ordered execution was accomplished through the use of salience, control patterns, and/or declarative agendas.

Group tests were accomplished in three steps: inspection, compilation, and execution. First, inspection compared the rule code with the source requirements. Here, though, contrasted with individual rule inspection, the intent was to associate all design specifications with a rule or rules to ensure complete coverage of the specifications. Next, rules were compiled as a group instead of individually as in the rule tests. This uncovered any rule syntax interactions such as renaming of rules, patterns, schema definitions, or secondary language functions within a group. Third, each group of rules was executed. The objective was to verify the proper order of execution of rules in relation to each other as the heirarachy of testing progressed. Executions utilized predefined sets of input facts, conditions, data, etc. The intent was to verify the functionality of the group and make sure that each rule was fired with appropriate fact values.

Interface Tests

These tests were similar to group tests in terms of inspection, compilation and execution. But, the goal here was that interface functions between major rule groups or interfaces with users or other processes associated with the expert system as a whole should be verified explicitly. Significant system data flow is characteristic of functional interfaces and warrants special attention. Validating interface specifications were of particular importance during this type of testing.

Interface testing activities included not only RENEX system software, but also software written and/or modified in the OOS. OOS software changes included in this testing level consisted of four parts: 1) a software model added to the standard OOS configuration to gather and send simulation feedback data to the Symbolics, 2) a software program invoked at the Unix operating system level on the HP that served as a "pipe" for receiving

51

simulation and timeline information from the Symbolics and providing it to the OOS software, 3) a software model added to the standard OOS configuration to gather, calculate, and send simulation state data to the IMI for driving the movement of the graphics system objects, and 4) a software program invoked by the IMI to receive, manipulate, and move the graphic objects on the IMI display screen.

System Tests

System tests were designed to verify overall end-to-end system operation. The performance of the expert system was demonstrated by executing typical operational scenarios. Stress or robustness testing was accomplished by the operator selecting improper or unusual inputs or input combinations.

User Tests

User tests involved a series of unstructured executions. They allowed the user to test the effectiveness of the system for such things as input and output of information as well as to further test the system for handling typical user scenarios.

EVALUATION

If we apply traditional definitions of software development to expert systems, we can think of expert knowledge to be incorporated into the program as the program specification, i.e., the description of what the program should do. Program verification then consists of insuring that the implemented rules perform as the expert specified. Errors can therefore be introduced in the following ways:

1) The expert does not correctly or completely convey his expertise, or his expertise is itself in error;

2) The knowledge engineer does not correctly translate, or map, the expertise into rules that correctly relate the conditions and actions combinations which the expert specified;

3) The control structure embedded in the rules contains errors;

4) The syntax contains errors.

Based on these definitions, the software engineering methods used to design and verify RENEX appear to have been successful.

It is well known that errors detected early during software development can be corrected with less impact on the system and with fewer resources. It seems safe to assume that the same will be true of expert systems. It follows that the most costly and critical problems can be introduced in the knowledge engineering effort. There appears to be no recourse if the expertise is incorrect, since the incorrectness might not be determined until the system is operational or has even been operating for some time. Stimulating the expert to think critically about the problem during the knowledge engineering effort may result in uncovering limitations or reevaluating conclusions.

The "mapping" of the expert's knowledge into rules occurred during meetings between development team and expert personnel. These meetings reviewed available information and discussed operations and functional processes of the proposed expert system. Early information was captured in the form of generalized rules or heuristics. As this information was refined and expanded, many detailed rules were often necessary to clearly express the earlier rules. These rules were then documented in a format more consistent with the expert system implementation language. The key to proper mapping of expert information appears to be effective documentation which bridges the gap between analytical level descriptions and implementation level specifications, i.e., between rules and concepts as described by domain experts and rules and data as expressed in the expert system software format. Forms used during RENEX development served as an path from the expert's perspective to the implementation format. Rule validity was further enhanced by involving the experts in design reviews. These reviews proved very effective in confirming information as well as pointing out previously unnoticed shortcomings.

It should be realized that an expert system is not a "fruit salad" where one simply adds rules. The rule/fact control and management must be explicitly designed and built into an expert system rule base. A significant milestone for RENEX was the formulation of a model of the trajectory planning process that all participants felt was thorough, complete and accurate.

An number of considerations affect the ease with which the system can be implemented. Syntactic complexity of expert system development tools affect the ease with which rules are specified and the number of errors in the implmentation of the expert system. This complexity is paradoxical in that it lessens the chances for coding errors while at the same time increasing the difficulty of learning to use the tool properly and efficiently. The implementer's experience with the expert system development tool and familiarity with the software engineering methods affect the learning curve required to get the project underway. The organization of RENEX rules into small, manageable groups allowed implementation and testing to occur concurrently and scheduling and planning to occur without sacrificing continuity.

53

CONCLUSIONS AND RECOMMENDATIONS

Based on our experience with RENEX, we conclude that performing expert system development in a structured manner offers the following advantages:

- manageability of the knowledge engineering process,
- production of maintainabe code,
- manageability and effectiveness of verification,
- adaptability to accommodate varying project size,
- flexibility in sequencing development activities,
- support of configuration control,
- scheduling and controllability of the development process,
- option to blend the expert system development gracefully into a conventional software development environment

This leads to the expectation that expert system software development can and will become part of a "composite software engineering" process which supports the entire life cycle. This explanation is further supported by the fact that RENEX software engineering methods are being adopted for real-time expert systems development by the Mission Planning and Analysis Division. RENEX itself is serving as a base of several spinoffs, indicating that the goal of technology transfer is being accomplished.

Although RENEX software engineering was based on pragmatic, engineering principles rather than theoretically based principles, the authors believe that the strategy used can be an effective software engineering approach to expert systems design and verification.

REFERENCES

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

(2) "Guidelines and Requirements for the Development of a Rendezvous/Proxops Trajectory Control Expert System (RENEX)," Bochsler, Daniel C., and Mary Ann Goodwin, JSC Internal Note JSC-22024, February 1986.

(3) "Software Design for the Rendezvous/Proximity Operations Trajectory Control Expert System (RENEX)," Bochsler, Daniel C., and Mary Ann Goodwin, JSC Internal Note JSC-22250, August 1986.

	Definition Phase	Requirements Phase	Design Phase
OBJECTIVES	– SCOPE SYSTEM FUNCTIONS – DEVELOP AN ARCHITECTURE – DEVELOP SKILL WITH THE HARDWARE AND SOFTWARE TOOLS	– ESTABLISH DEVELOPMENT BOUNDARIES – DEVELOP DETAILED REQUIREMENTS	– DEVELOP A DETAILED SYSTEM DESIGN
ACTIVITIES	– KNOWLEDGE WAS OBTAINED INFORMALLY FROM EXPERTS AND DOCUMENTATION – A SMALL SUBSET OF SYSTEM FUNCTION WAS PROTOTYPED – PROTOTYPE EVALUATED	– GUIDELINES SPECIFIED FOR DOMAIN, DESIGN APPROACH, USER INTERFACES, ETC. – DETAILED MEETINGS WERE HELD WITH EXPERTS – "KNOWLEDGE FORMS" WERE DEVELOPED TO RECORD INFORMATION – PUBLISHED FORMAL GUIDELINES AND REQMTS.	– UTILIZED HIPO* SYSTEMS ANALYSIS AND DESIGN TECHNIQUE – SOME PROTOTYPING WAS DONE FOR LESS WELL DEFINED PORTIONS OF DESIGN – DESIGN WAS REVIEWED IN A MANNER SIMILAR TO A SOFTWARE DESIGN REVIEW
PRODUCTS	– PROTOTYPE – FUNCTIONAL STRUCTURE	– REFERENCE 2	– REFERENCE 3

* Hierarchical Input Process Output

Table 1 – Summary of Definition, Requirements and Design Phases

SYMBOLICS 3670

SYSTEM MONITORING & ASSESSMENT

EVENT PLANNING/ REPLANNING

FACT BASE

GN&C I/F

USER I/F

HP9000

FLIGHT SOFTWARE

HARDWARE

ENVIRONMENT

IMI500

GRAPHICS HARDWARE & SOFTWARE

USER
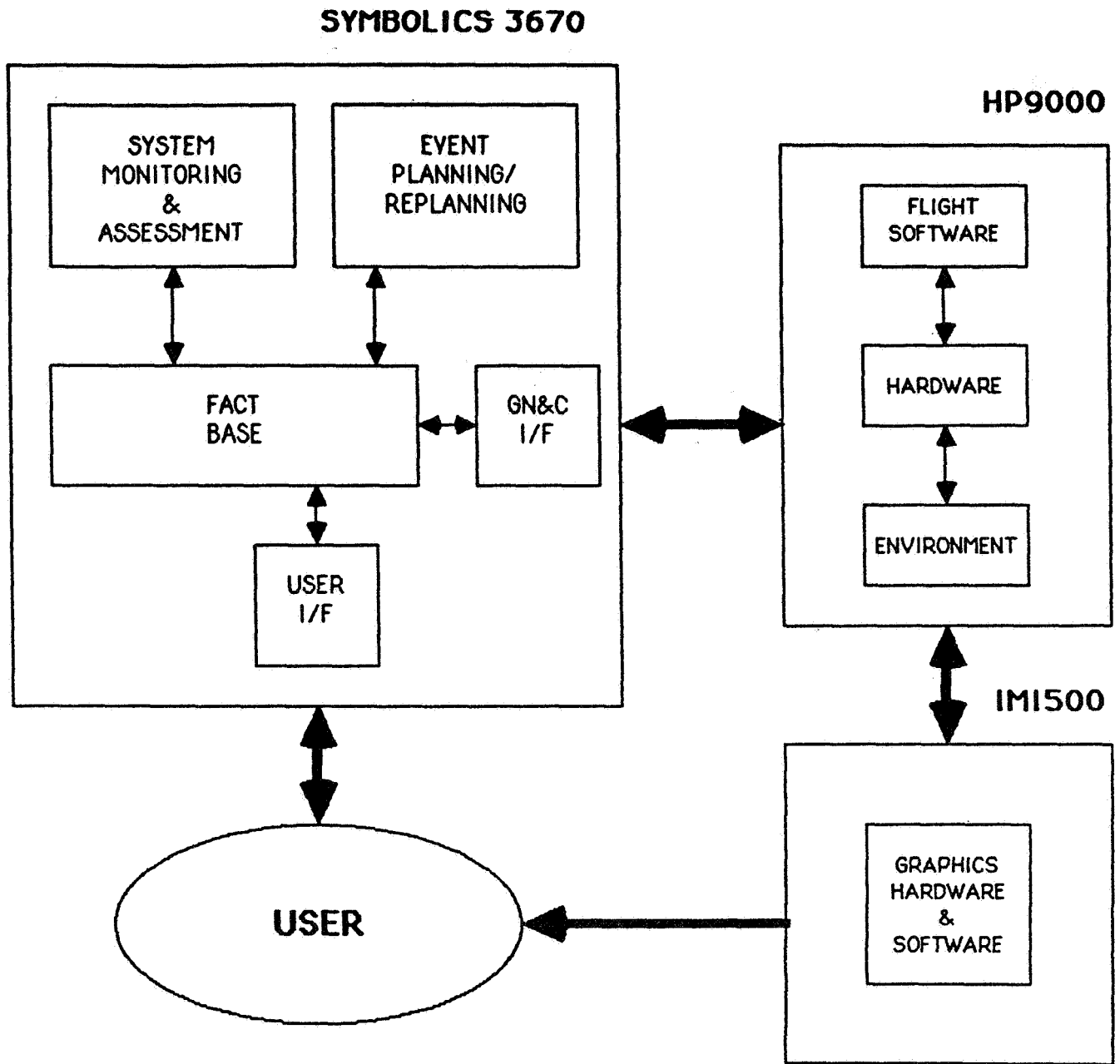
Figure 1 - RENEX System Configuration
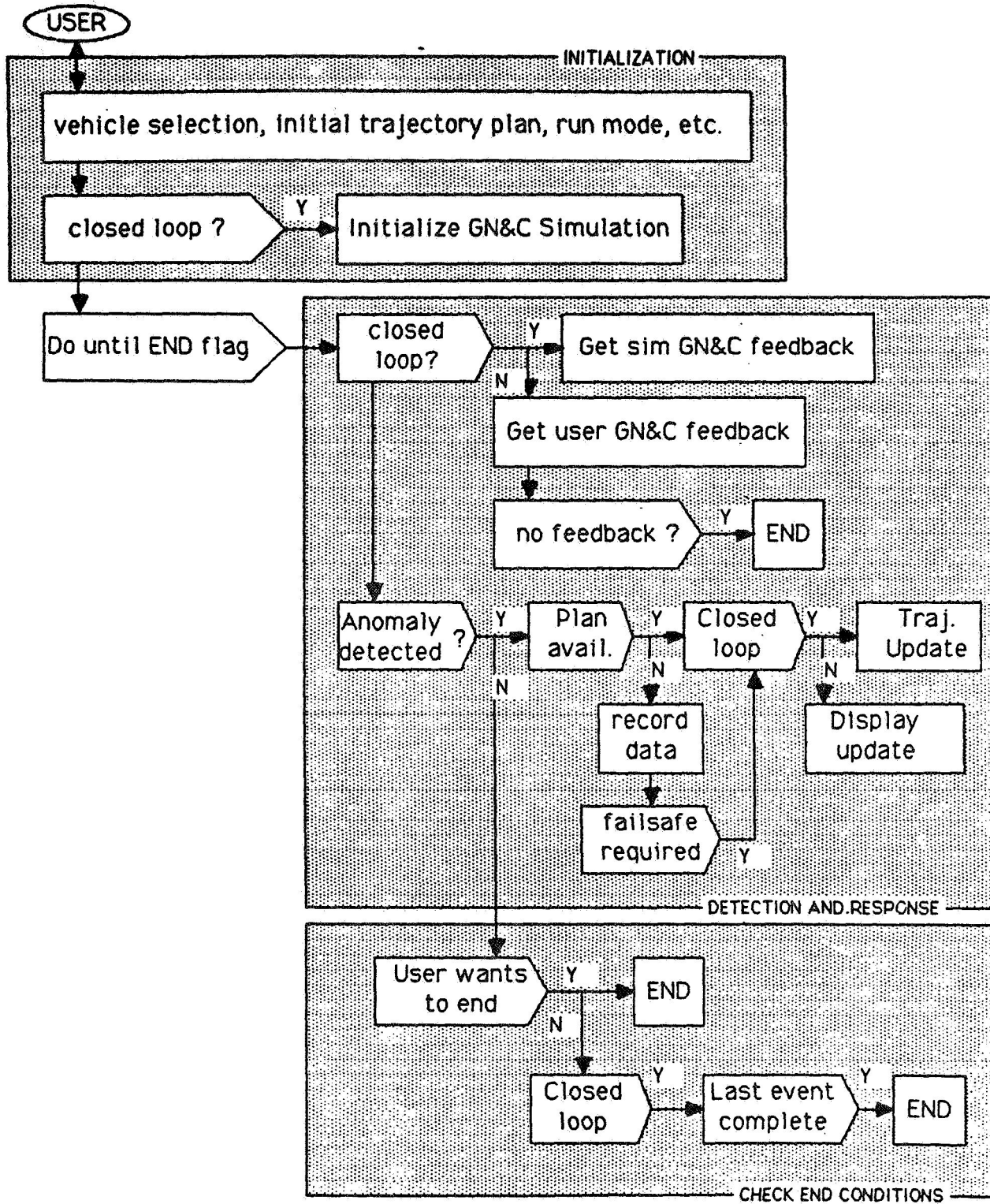
Figure 2 - Functional Structure of the RENEX Expert System

Figure 3 - RENEX Software Design Structure

Lowest Logic Level

INPUT

user selection

START

PROCESS

1.0 Display to user a menu of system over-
head options including:
- user response trace (on/off)
- speech synthesis (on/off)

2.0 Set user trace and speech status to
selected values, and set trace file
if necessary.

3.0 Display to user a menu of system
configuration options consisting of:
a) open-loop: meaning user provides
feedback.
b) closed-loop: meaning a simulation
provides the feedback.
c) terminate expert system funct-
ions: meaning halt system actions.

4.0 a) If choice = open-loop or closed-
loop.
Then 1) set run mode to the
users choice.
2) send user trace msg:
"user chose <user choice>
system configuration".
3) update mission summary
display to indicate user
choice.
b) If choice = terminate system,
Then 1) send user trace msg:
"user wanted to stop".
2) verify this choice by
asking the user "are you
sure – yes or no".
3) get user response.
4) if response = yes
then set halt status.
5) if response = no,
Then go to step #3 above.

OUTPUT

user display

trace option status fact

speech option status fact

trace file

run mode fact

trace information fact

halt status fact

A

TO 2.2

Figure 4 – HIPO Design Specification Example from RENEX
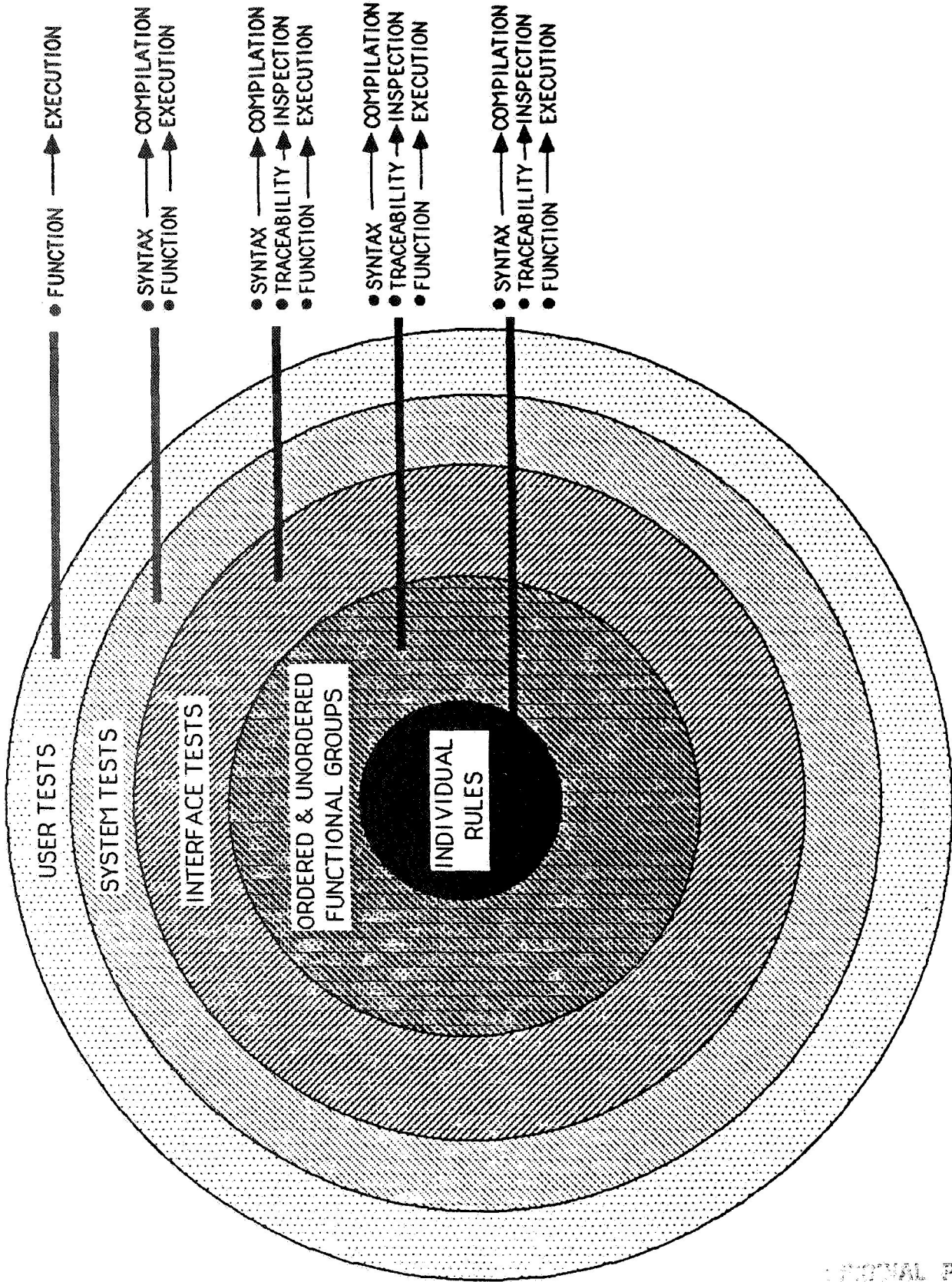
Figure 5 - Levels of RENEX Expert System Testing

USER TESTS

SYSTEM TESTS

INTERFACE TESTS

ORDERED & UNORDERED FUNCTIONAL GROUPS

INDIVIDUAL RULES

- FUNCTION ⟶ EXECUTION

- SYNTAX ⟶ COMPILATION
- FUNCTION ⟶ EXECUTION

- SYNTAX ⟶ COMPILATION
- TRACEABILITY ⟶ INSPECTION
- FUNCTION ⟶ EXECUTION

- SYNTAX ⟶ COMPILATION
- TRACEABILITY ⟶ INSPECTION
- FUNCTION ⟶ EXECUTION

- SYNTAX ⟶ COMPILATION
- TRACEABILITY ⟶ INSPECTION
- FUNCTION ⟶ EXECUTION

ISSUES ON THE USE OF META-KNOWLEDGE IN EXPERT SYSTEMS

Jon Facemire and Imao Chen

Computer and Information Sciences Department,
Alabama A & M University, Normal, Al 35762, U.S.A.

ABSTRACT Meta knowledge is knowledge about knowledge; knowledge that is not domain specific but is concerned instead with its own internal structure. Several past systems have used meta-knowledge to improve the nature of the user interface, to maintain the knowledge base, and to control the inference engine. More extensive use of meta-knowledge is probable for the future as larger scale problems are considered. A proposed system architecture is presented and discussed in terms of meta-knowledge applications. The principle components of this system: the user support subsystem, the control structure, the knowledge base, the inference engine, and a learning facility are all outlined and discussed in light of the use of meta-knowledge. Problems with meta-constructs are also mentioned but it is concluded that the use of meta-knowledge is crucial for increasingly autonomous operations.

1. INTRODUCTION

The Greek prefix "meta" means "among" or "after". Thus to Aristotle metaphysics is what is "after" physics. Gradually "meta" came to mean, in English, "beyond" in an abstract sense. So "meta-knowledge", then, means "beyond knowledge"; an abstraction of knowledge about knowledge. In expert systems meta-knowledge can be viewed as the incorporation into the program of knowledge about its own internal workings; of how much it knows and how it reasons.

Meta-knowledge is information one step removed from domain specific knowledge. Meta-knowledge is typically more general and less subject to change than is knowledge from one particular application area. The incorporation of meta-knowledge can make a system more adaptable and efficient. The use of meta-concepts can be viewed as a means to make the system more intelligent; gradually more and more self knowledge enabling the system to become a better problem solver and aware of its own limitations. Ultimately, meta-knowledge can make the system autonomous and free to act without the presence or intervention of human aid. It is this possibility of greater independent action that makes meta-knowledge especially appropriate to a space based endeavor.

2. BACKGROUND OF META-KNOWLEDGE USAGE

Ideas about meta-knowledge are hardly new and date back at least to McCarthy in 1960 although relatively little meta-programming was done until the mid 1970's (Genesereth, 1983). Since that time, several well known expert systems such as TEIRESIAS (Davis, 1982) and META-DENDRAL (Buchanan, 1978) have incorporated mechanisms to deal with meta-knowledge. This was done primarily to make the individual programs less domain specific so that new areas of expertise could be accomodated without extensive rewrites of the original programs. More recently the idea of meta-knowledge has been broadened a bit and used several ways, such as in GUIDON as part of a tutorial package (Clancey, 1981), in ANALOG to form abstraction models (Genesereth, 1980), and in JAUNDICE to discover new control rules (Fu, 1984).

The various applications of meta-knowledge have generally fallen into three main categories: control of the user interface, maintenance of the knowledge base, and control of the inference engine. Most meta-systems have used rules for the knowledge representation scheme although other mechanisms such as semantic nets or frames could be used. Rules seem to work well for reasonably sized problem domains and they are inherently incremental and comprehensible; no

doubt accounting for their popularity.

Meta-knowledge is used to control the user interface typically for both knowledge acquisition and explanation. In the case of acquiring knowledge, meta-knowledge is necessary about the permissible ways in which domain knowledge can be represented internally. Knowledge about the nature of the user can be useful, too, and both areas used to guide the user into supplying appropriately formatted information.

Once the user does enter data into the system, the meta-knowledge about the knowledge base itself can be used to insure integrity. Incoming information can be checked against existing knowledge for consistency and completeness. TEIRESIAS (Davis, 1977), for example, checks antecedent clauses and requests more information about new rules if the clauses seem incomplete based on prior rules.

When the information is finally complete, the system can then be used for actual problem solving. Here, the meta-knowledge can be used to control the inference engine and it is in this area that most work has been done with meta-contructs. Meta-rules can be used for ordering other rule firings, for resolving uncertainties, or for planning a general approach to the problem solution(Davis, 1980b). Sub-goals can be specified, too, and used to coordinate planning systems (Clancey, 1983).

After the inference engine has done its work, there is still the task of displaying the results to the user and explaining the logic of the solution in an explanation facility. Meta-knowledge can be used in this case to choose an appropriate explanatory mechanism and level of detail. Knowledge of both the problem itself and of the nature of the user is necessary for this output interface to work properly.

To date, there have been relatively few empirical studies on the level of improvements that meta-knowledge provides to expert system operation; but the studies that have been done (Fu, 1980) indicate sizeable savings in such areas as number of rule firings needed to achieve a solution. It seems reasonable to expect that more and more attention will be paid to meta- representations in the future. Truly large knowledge bases will be required for more sophisticated problems and meta-knowledge is one of the few software

mechanisms that offers any hope of reducing the combinatorical explosion involved in large domains. Hardware supports, in the form of multiple processing elements, may also be needed but the problems associated with coordinating such elements to one task also implies considerable self knowledge about the task and the mechanisms used to handle it. Users will also demand more "intelligent" interfaces to simplify the insertion of knowledge and the extraction of answers and this, too, suggests meta-knowledge. Therefore it seems inevitable that the design of expert systems must include features to handle meta-knowledge.

3. A PROPOSED META SYSTEM ARCHITECTURE

Figure 1 shows a block diagram of a suggested expert system architecture. This outline is general enough to represent a wide variety of possible applications and representation methods but detailed enough to reflect major functional components. There is no separate meta-system represented; instead the meta-components are present in each of the main subsystems and will be discussed in turn. The diagram is, in any case, more of a symbolic than a physical partitioning. Nevertheless, most of the meta-aspects would probably exist as distinct modules or files merely to simplify development from a software engineering point of view.
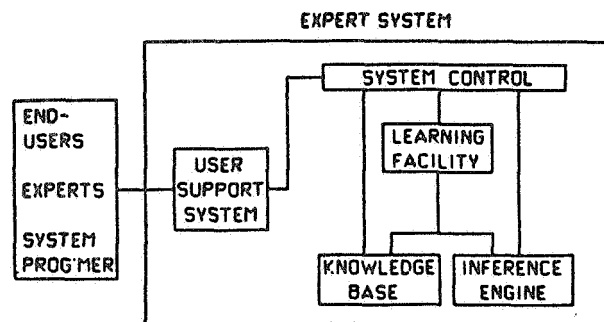

Figure 1. Expert System Architecture.

3.1 USER SUPPORT SYSTEM

First of all, the user support system would consist of all the actual communication software that would interact with the user. This would include

62

relatively low level subsystems, such as a simple file editor or graphics display program,as well as more complex parts where meta-knowledge would be of value.The interface would also include ways to explicitly locate, add, and edit the meta-features as being distinct from more procedural modules.

One necessary facet to any intelligent communications would be the ability to form a model of the user. A portion of the knowledge base could be used to keep specifics about the nature of the user and the past history of user interactions. The nature of explanations, for instance, would be considerably different depending on whether the user was relatively naive or was the actual domain expert checking the system performance. The explanation facility should have a good understanding of what the user should or should not know. The expert should not be bored with a belaboring of the obvious and the novice should not be confused by lack of detail. If the user is the program author, then a trace of the code involved might be in order as an explanatory mechanism, but this would be worse than useless to a non-programming end user. It seems that there are at least three kinds of users with completely different needs; namely the system programmer, the domain expert, and the end user. Each would have different needs and different allowable accesses to the system. Of course, the user might not be human at all but a sophisticated control or robotic system.

The situation, also, affects the kind of explanations given. Some kinds of data are better shown as a graph or flow diagram while other types of information should be shown alphanumerically. Simple meta-based rules and assumptions can make use of the system much less tedious.

The user support facility must also provide an editor, a security system, debugging aids, and routine file utilities. There is less application in these areas for meta-knowledge; although considerable effort could be spent here on a natural language capability. A typical menu driven input request involving meta-knowledge would resemble the following; taken from a stock investment expert system (Davis, 1977). This example shows simple knowledge about the nature of stocks and their associated attributes as a new stock type is entered.

Should this new item be added to -
1 - the list of common stocks, or
2 - the list of preferred stocks, or
3 - the list of cumulative preferred stocks, or
4 - the list participating preferred stocks.
Select one of the above.

3.2 SYSTEM CONTROL

Considerably greater possibilities exist for the use of meta-knowledge in the system control module. This module is analgous to an operating system and as such controls the interaction of the other component parts. This can include considerable control over both software and hardware resources, especially in a multi-processor architecture, and so a good deal of "meta" hardware knowledge might be necessary(Konolige, 1980). The numbers and kinds of available resources and their relationships to the process of problem solving are a natural application for meta-knowledge(Genesereth, 1983).

One key component in the control system is the command interpreter which parses the dialog coming from the user interface and executes accordingly. This could involve considerable meta-knowledge about the nature of the problem including the rather basic notion of whether the facilities are present to solve the problem or not. Clearly the problem could fall outside the domain areas of the system or the hardware might not be available to do the job in the time allocated. At the very least, meta-knowledge should be incorporated into any system to enable it to know the scope of its own abilities. This knowledge of limitations versus capabilities should be part of the command interpreter to provide timely intervention in the event the system is not capable of useful interaction.

Even if the problem is potentially doable there is still the problem of assigning the right processes to the job. The knowledge base may be logically partitioned into many areas and the control system needs to decide which context is applicable to the problem; the SPHERE system uses such a relevant context scheme(Filman, 1981). Then too, the control module must decide if time is a significant factor. If so, then the inference engine must be told how severely to limit its activities. Permissible risk is another related factor; the inference

engine would operate differently depending on how serious were the consequences of possible error. Again, this is a factor to be determined by the control mechanism using meta-knowledge.

The nature of the dialog with the user would also be partially the responsibility of the control system. Although the user support system would take responsibility for the actual display and input of dialog, it would be the control module that would decide what the nature of the dialog should be. There could be many possible requests for more information made of the user at any one time; the control system would have to decide which request to service. Meta-constructs have been used in this area(Slagle, 1983).

The control system would also be in charge of resolving any conflicts in output to the user. Consistency checking and other kinds of verification would be done here as many systems have already done(Goosens, 1979; Roach, 1984). Furthermore, there should be some validation of answers as well. As much feedback about system performance as possible should be accomodated by the control system. Statistics should be kept in the knowledge base and used to modify system parameters when possible to improve performance. Whenever possible, the system needs to be made aware of the accuracy of its past performance so that its ability can be incrementally improved.

A further task for the control system would be to detect and resolve any internal problems. Vicious cycles are always a possibility as are deadlocks or recursive control problems. The system control would be responsible for detecting and remedying any such happennings; this also involves meta-knowledge, this time of self diagnosis.

An example of a meta-knowledge based request from the user follows. This example, from TEIRESIAS(Davis, 1977), points out the program's ability to recognize a possible inconsistency or need for clarification.

I hate to criticize, but did you know that most rules about what the area of investment might be that mention the income-tax bracket of the client and how closely the client follows the market also mention - [A] -the amount of investment experience of the client. Shall I try to write a clause to account for [A]?

3.3 KNOWLEDGE BASE

The knowledge base itself might not have as many different functions as the control system; but the knowledge base is likely to be physically by far the largest unit and is sure to have a fairly complex internal structure. There may be several logical partitions in the knowledge due, for one thing,to different problem domains. The knowledge about pulmonary disease, for example, may be kept distinct from the knowledge of liver disease. A division by domain allows for an orderly review of the knowledge base during development and a means for limiting the consideration of applicable data during inference. As knowledge bases get to be very large, more and more kinds of subdivisions may be necessary for efficiency of search. These structures can not be too restrictive, however, as much practical knowledge will always be difficult to categorize in any schema. Storing a string of associated vectors with individual entities or an indexing of rules by antecedent clause are possibilities that could be used as an alternative to a physical division. A partitioning is likely, too, into domain specific, domain independent, and meta-subparts. The domain independent parts would represent general kinds of knowledge that would be universal to all application domains. This part of the knowledge would be available to the inference engine for all domains and would include some kinds of meta-knowledge such as the nature of available internal data structures. Some meta-knowledge, however, could still be associated with specific domains. Furthermore, a breakdown seems beneficial into declarative and procedural information(Gallanti, 1985; Balzer, 1977; Aikins, 1980). Declarative knowledge would mostly be associations of the is-a or is-a-kind-of type, while procedural knowledge would define the actions to be taken or the way new associations could be formed. It would also be desirable to represent knowledge in several different ways 'such as rules, nets, or frames so that whichever mechanism is most suitable could be used for a particular problem. However, multiple representations lead to serious complexity problems. In general, there is a trade off between making knowledge bases amorphous, which leads to a simplicity of structure but huge search space; or highly structured which may simplify control but make it difficult to represent all relevant facts completely.

3.4 INFERENCE ENGINE

Part of the nature of the inference engine is determined once the knowledge representation scheme is fixed. If the system is rule based, then some kind of forward or backward chaining mechanism is dictated. If the system is based on a semantic net, though, then other traversal procedures are called for. In any representation, however, there will be uses for meta-knowledge as a means of control. This meta-knowledge can be used mainly in two ways, namely in the control of what to do next and in the resolution of uncertain or incomplete information.

Control of the inference engine is indeed the largest area of research in meta-programming (e.g. Davis, 1980a; Friedman, 1985; Hudlicka, 1984) and promises the greatest gains in overall performance. In any expert system, there will be many occasions where there will be a conflict in terms of what the next action should be. For rule based systems there may be many possible rule firings at any given point and the choice of which to do is a meta-control issue. If time is critical then perhaps the rule with the fewest consequences should be done next or a decision might be made on the quality of competing rules with the one having more confidence associated with it taking priority. Past performance success can also be used as part of the decision process as the system can remember which technique worked best in the past. Well known search strategies such as branch and bound can also be employed in order to find applicable knowledge. Which search strategy to use is yet another case for the use of meta-knowledge. Of course, any efficiency gained by the use of meta-procedures must be evaluated against the extra overhead involved in considering a large number of meta-knowledge possibilities(Barnett, 1984). For large amounts of meta-knowledge, there may well have to be an internal structure in the meta-knowledge data base as in the domain knowledge base.

The other primary use of meta-knowledge is in the area of uncertainty; of dealing with knowledge that is missing, garbled, vague, or in conflict. It seems that a great deal of the skill that human beings have is related to our ability to make usually valid judgements in such cases. In the attempt, then, to mimic human skill it is necessary for expert systems to deal with ambiguities. Confidence factors, fuzzy reasoning, weighting schemes, and consistency checking (e.g. Reboh, 1981; Thompson, 1984; Shenoi, 1984) have all been used as meta-knowledge techniques to obtain solutions when exact procedural approaches are inadequate. The nature of the knowledge base and of the domain task are the primary determining factors in the selection of an appropriate method. The inference engine ideally would be able to handle more than one approach; again, at the cost of more overhead.

A typical control meta-rule would be as follows; here used to specify sub-goals (Clancey, 1983).

If the hypothesis being focused upon has a child that has not been pursued then pursue that child.

3.5 LEARNING FACILITY

The learning facility is not ordinarily considered a separate part of an expert system but its inclusion as such seems inevitable. No production system can remain static for very long as new knowledge is steadily discovered and old lapses in design are uncovered. It is best to have a formally declared learning system to both learn new knowledge from the users and to uncover knowledge already hidden within the system. Thus, the knowledge acquisition facility is contained within the learning module and as such is responsible for formatting incoming information either into an appropriate internal form or into a new record structure tailored to the new data.

The learning facility should also be capable of introspection and by this technique be able to discover new truisms as yet unspecified. Rules leading to a greater probability of success could be identified, for example, and a meta-rule created to try these rules ahead of others. On the other hand, rules that never led to a successful conclusion could be found and deleted. If time constraints forbid the constant application of the learning system for introspection, the system controller could put such activities at a lower priority level - to be done whenever time allowed. Various pattern matching algorithms could also be included here to look for, say, similarities in rule structure. It would be wise, of course, to mark in some way any changes in the program by the learning module so that the programmer or expert could review

65

them later.

The learning facility would provide both the mechanism for the initial input of data into the knowledge base and also the means for subsequent improvement. As the learning system continues to operate, the performance should get incrementally better. In this way the overall system would gradually gain human like degrees of expertise and autonomy.

An example of a meta-learning rule would be as follows (Fu, 1980).

    If a part of the differential list
    is: "((M1 M ...)"
    then a potential reordering meta-rule
    can be formed
        MR3: A1 ->
        Rules mentioning M3 DOBEFORE
        Rules mentioning M1.
    Since M1 is denied by A1.

4. CONCLUSION

There are still unresolved issues in the design of large expert systems. The exact nature of the best knowledge representation, or representations, to be employed is an open issue - as is the nature or how to handle uncertainty and degrees of confidence. There is a trade off, too, between efficiencies gained by application of meta-knowledge and the extra overhead involved in such approaches. The optimal fashion for dividing up the knowledge base is also uncertain; perhaps no divisions ought to be made at all or perhaps the granularity should be quite small. The role of multi-processors is certain to be a major factor in the relatively near future but exactly how expertise should be divided and recoupled is unclear.

Continued work in the area of meta-knowledge and expert systems should show steady progress, however. The more self knowledge any agency has the more capable it is of handling ordinary and extraordinary stress. As the expectations for expert systems get higher, the more self reliant they must become. This indicates increased usage of meta-knowledge and meta-programming. The inclusion of meta-constructs should, therefore, be assumed at the initial design phase and the interfaces between meta and domain modules be made explicit.

REFERENCE

The following references include works that are important but not referenced.

Aikins, J. Representation of control knowledge in expert systems, National Conference on A.I., 1980, 121-123.

Balzer, R., Goldman, N., and Wile, D. Meta-evaluation as a tool for program understanding, International Joint conference on A.I., 1977, 398-403.

Barnett, J. How much is control knowledge worth? A primitive example. Artificial Intelligence, 1984, 22, 77-89.

Barr, A. Meta-knowledge and cognition, Sixth International Conference on A.I., Tokyo, (vol. 2), 1979, 31-33.

Buchanan, B. G., and Feigenbaum, E.A. DENDRAL and META-DENDRAL: their application dimension, Artificial Intelligence, Vol. 11, 1978.

Clancey, W.J., and Letsinger, R. Neomycin reconfiguring a rule-based expert system for application to teaching, International Joint Conference on A.I., (Vol. 2), 1981, 829-835.

Clancey, W. The advantage of abstract control knowledge in expert system design, AAA1-83, 1983, 74-78.

Davis, R., and Buchanan, B. Meta-level knowledge: Overview and applications, International Joint Conference on A.I., 1977, 920-927.

Davis, R. Meta-rules: reasoning about control Artificial Intelligence, 1980a, 15, 179-222.

Davis, R. Content reference: reasoning about rules, Artificial Intelligence, 1980b, 15, 223-239.

Davis, R., and Lenat, D. Knowledge-based in artificial intelligence, McGraw Hill, 1982.

Davis, R., and Buchanan, B. Meta-level knowledge. In Buchanan, B., and Shortcliffe, E. (Eds.) Rule based expert systems, Reading, mass., Addison Wesley, 1984, 507-558.

Doyle, J. A truth maintenance system, Artificial Intelligence, 1979, 12, 231-272.

Filman, R.E., Lamping, J. and Montalvo, F.S. Meta-language and meta-reasoning, International Joint Conference on A.I., (vol. 1), 1981, 365-369.

Friedman, L. Controlling production firing: the FCL language. International Joint Conference on A.I., 1985, 359-366.

Fu, L., and Buchanan, B. Enhancing performance of expert systems by automated discovery of meta-rules, IEEE, 1980, 107-115.

Gallanti, M., Guida, G., Spampinato, L., and Stefanini, A. Representing procedural knowledge in expert systems: an application to process control, International Joint Conference on A.I., (Vol. 1), 1985, 345-352.

Genesereth, M. Metaphors and models, AAAl-80, 206-211.

Genesereth, M. An overview of meta-level architecture, National conference on A.I., 10983, 119-124.

Ginsberg, M. Non-monotonic reasoning using Dempster's rule, National Conference on A.I., 1984, 126-129.

Goosens, D. Meta-interpretation of recursive list-processing programs, Sixth international Joint Conference on A.I., Tokyo, (vol. 2), 1979, 7-12.

Hudlicka, E., and Lesser, V. Meta-level control through fault detection and diagnosis, National Conference on A.I., 1984, 153-161.

Konolige, K. and Nilsson, N. Multiple-agent planning systems, AAAl-80, 1980, 138-142.

Reboh, R Extracting useful advice from conflicting expertise, International Joint Conference on A.I., (Vol. 1), 1981, 145-150.

Roach, J., Lee, S. Wilcke, J., and Ehrich, M. An expert system that criticizes decisions in combination drug therapy, IEEE, 1984, 344-349.

Shenoi, S., Fan, L., Toguchi, K., and Lai, F. Meta-knowledge in balanced premise evaluations, IEEE, 1984, 518-523.

Slagle, J., and Gaynor, M. Expert system consultation control strategy, AAAI-83, 1983, 369-372.

Thompson, T.F., and Wojcik, R.M. Meld: An implementation of a meta-level architecture for process diagnosis, IEEE, 1984, 321-330.

A Knowledge-based Decision Support System for Payload Scheduling

Stephen Floyd*
and
Donnie Ford**

*Department of Management Information Systems and Management Science
School of Administrative Sciences, University of Alabama in Huntsville

**Cognitive Systems Laboratory - Johnson Research Center
University of Alabama in Huntsville

ABSTRACT

The purpose of this paper is to illustrate the role that artificial intelligence/expert systems technologies can play in the development and implementation of effective decision support systems. A recently developed prototype system for supporting the scheduling of subsystems and payloads/experiments for NASA´s space station program is presented and serves to highlight various concepts. The potential integration of knowledge based systems and decision support systems which has been proposed in several recent articles and presentations is illustrated.

1. INTRODUCTION

At the Sixth International DSS Conference (DSS-86) Peter Keen in the closing plenary address entitled "DSS: The Next Decade" discussed what he perceived as the important roles of current and future AI/ES technology in extending the field of decision support systems. Among his perceptions was the fact that the field of AI could play a major role in the development of systems to support the tougher, ill-structured types of problems. He also viewed current AI/ES hardware and software technology as "power tools" for DSS development. A few months earlier John Little in an article entitled "Research Opportunities in the Decision and Management Sciences" promoted similar observations while discussing research priorities' of NSF´s Decision and Management Science program [9]. Major among these priorities was the role that expert systems technology could play in advancing the Decision Sciences. Similar ideas have been expressed over the past few year by other reseachers in articles and at major conferences such as ORSA/TIMS, DSS-86, IDS and AAAI [8], [13], [16], [17]. This paper supports these observations by describing knowledge-based DSS for scheduling payloads for NASA´s space station program. The payload scheduling system serves to illustrate the potential integration of DSS and ES as it involves the addition of a knowledge based component to a system which currently provides decision support via extensive interaction between scheduling personnel and more traditional scheduling techniques. It is the authors´ hope that the following discussion of the scheduling system will help other researchers in establishing the applicability of the new "power tools" in DSS development.

This paper concerns the development of a solution procedure and interactive system for scheduling subsystems and payloads/experiments for the National Aeronautics and Space Administration space station program. Traditionally, scheduling problems have been viewed as static in nature (i.e., a schedule is developed for a particular planning horizon and adhered to) and were cast as having one or more clearly defined objectives (e.g., minimize overall completion time,

maximize resource utilization, etc.). As such, these problems were most commonly solved via application of optimal seeking algorithms, heuristics or simulation analysis [1] [4] [6] [7] [15]. The payload scheduling problem, in contrast, is representative of a class of scheduling problems which are highly dynamic in nature. Not only may the various parameters change at any time, but the objectives themselves may change also. As will be illustrated in this paper, the nature of this class of problems is such that they can be most effectively solved by knowledge based expert systems [2] [3] [5] [11] [18] [19].

Provided in the first section of the paper is a detailed description of the class of problems under investigation. After an overview of the problem domain, the specifics are provided for the NASA problem which lead to the development of the system. The third section discusses the initial dynamic scheduler solution strategy that was developed for the prototype system. The details of this prototype expert system and its development are provided in the fourth section. The fifth section discusses future enhancements that have been identified for the system. The final section of the paper provides some concluding remarks on the research to date, and some suggestions for future research in the area of dynamic scheduling.

2. PROBLEM DESCRIPTION

The application addressed in this paper concerns development of a system for the scheduling of subsystems and payloads aboard the space station. Subsystems are systems which function to support space station on an ongoing basis. These include such subsystems as life support systems, communications systems, and various "housekeeping" systems. Aboard space station will also be various payloads and experiments. These will be sponsored not only by NASA but also by other U.S. and foreign government agencies, universities and private industries.

Each of the subsystems or payloads has a certain set of characteristics and requirements which must be considered in determining when during the mission it should be scheduled. For example, each subsystem and most of the payload/experiments will draw operating power from Space Station's limited power supply. Additionally, certain of them will require astronaut intervention either on a continuous basis for the duration of the experiment or for specified subintervals of time. Some subsystems and experiments are continuous in nature and run uninterupted throughout the entire mission. Still others operate either continuously or intermittently for only a specified subinterval of the mission time window. The nature of some experiments will require that they be conducted only during certain phases of the mission (e.g., during day orbit, during night orbit, during certain orientations of space station, etc.). These example characteristics, as well as others which will not be detailed here, coupled with the fact that payload/experiments are placed in priority classes which must be reflected in the schedule, form the basic criteria for establishing feasible schedules.

The complexity of the scheduling problem is compounded further by the fact that events which will be occurring during the mission will serve either directly or indirectly to upset current schedules and/or influence future ones. For example, at any time during the mission an ongoing experiment may fail or be aborted for some reason, a scheduled experiment may be withdrawn from the schedule, an experiment or entire class of experiments may be added and/or experiment priorities changed. The scheduler must be designed to handle such dynamic changes via interaction with various mission personnel, including astronauts, mission planning specialists and principal investigators of affected experiments.

70

As mentioned previously, each subsystem and payload/experiment will consume various resources. Major among these will be energy from the Space Station's power supply and manpower provided by the astronauts on board. Such limited resources place constraints on what systems and experiments can be concurrently ongoing. Additionally, and this is another of the dynamic aspects of the problem, the power and manpower allotments themselves may change at various times throughout the mission. In some instances the change notification will provide lead time for scheduling adjustments, whereas in others no lead time will be provided. Changes will occur, for example, when vehicles dock with Space Station. Such changes result from the fact that the docking will usually draw on such resources as the power and manpower supply. In light of the above mentioned characteristics, the scheduler must have capabilities beyond the generation of traditional static feasible schedules. The dynamic scheduler must have the capacity to respond interactively to such changes and, when required, maintain feasibility via a rescheduling procedure.

A final characteristic of the payload scheduling problem is that the scheduling objectives are variable. During the course of a Space Station mission, mission specialists may re-structure the scheduling objectives. For example, it might be that early in a mission a resource leveling strategy is adopted which will maintain a fairly constant and conservative power consumption rate. Such an objective would naturally "stretch out" the scheduling of experiments over some designated planning horizon. Later in the mission cycle, however, factors may change this objective to one of scheduling as many payloads/experiments as possible (subject to the maximum power availability and other constraints) in a given time frame. These characteristics then establish the need to develop a system which is capable of not only establishing static schedules but also of dynamically maintaining feasible payload/experiment schedules which reflect the varying parameters of the problem.

3. SOLUTION STRATEGY

Sample data around which the prototype system could be constructed was provided by NASA's Power Branch. The data as considered by NASA to be representative of actual scheduling data. As seen from Table 1, four subsystems and forty-five (45) payloads/experiments were included. Provided in the table are the experiment name, the associated power consumption requirements in kilowatts, the sponsoring agency, the time duration (including other specifications such as continuous/intermittent, day orbit/night orbit, etc.) and crew involvement required. In addition to the data in the table, other problem specifications were also provided. Most pertinent among these were (1) the specification of a normal lab module power level of 25 kilowatts, (2) a priority structure based on the sponsoring agency and the nature of the payloads/experiments, and (3) a two-week scheduling horizon. Additionally, several system requirements pertaining to the actual operation of the scheduler were specified. These provided a framework for the user interface and system output as detailed later in the system description section of the paper.

To prototype an initial system for user evaluation and feedback, a means of generating feasible schedules in the absence of a complete corporate knowledge base had to be developed. This was accomplished via the modification of a scheduling strategy presently used by NASA scheduling personnel which involves conceptualizing schedules using a Gantt chart type format. This heuristic procedure is representative of those that when augumented by various scheduling rules will comprise the scheduling knowledge base of the final system. An example schedule for a simple four experiment problem is given in figure 1. As can be seen, experiments one, two and three are continuous, and experiment four is intermittent.

71

Given inside the bars, which represent the experiment durations, is the power requirement of the particular experiment. For simplicity these power requirements are assumed constant as long as the experiment is "on." Through the use of this four experiment example, the heuristic will now be described.

As an experiment is placed on the chart, its beginning and ending point(s) serve to divide the overall time window, the x-axis, into intervals as illustrated by the dotted lines in figure 1. By updating as each experiment is scheduled, one can maintain for each subinterval of time the information necessary in determining the time slot for the next experiment to be scheduled. The determination of which experiment is to be scheduled next is based on the user predefined priority structure in effect at the time of the scheduling or rescheduling procedure. For the sake of illustration we will simplify the four experiment example further by assuming a single scheduling objective of maximizing power utilization. Each experiment is scheduled by searching through time on the x-axis in figure 1 from left to right until a subinterval or group of successive subintervals is found which has sufficient duration and power availability to support the given experiment. The experiment is then scheduled and added to the chart in correspondence with this subinterval. Subinterval information is updated to reflect resource availability to reflect resource availability (i.e., power and manpower) and the scheduling procedure continues.

Applying the scheduling heuristic to the representative problem provided by NASA is obviously much more involved than the example provided above as the various experiment characteristics and requirements must be matched to appropriate intervals. As the number of requirements increases for experiments, so too does the amount of infor mation being kept on each subinterval. Additionally, as the number of experiments already scheduled increases, the number of subintervals to be examined during each individual scheduling process also increases. This increase in the number of subintervals is compounded even further when the experiment scheduled is of an intermittent nature. These facts, coupled with the previously mentioned dynamic aspects of the problem, necessitate an automated procedure for generating schedules. The next section of the paper will describe the prototype system developed to accomplish this.

4. SYSTEM DESCRIPTION

The prototype system follows the basic production system structure of a knowledge base, inference engine and working memory or global data base. The knowledge base consists of a reduced set of scheduling rules and knowledge pertinent to the example problem. The system utilizes a frame representation scheme which allows for utilization and exploitation of knowledge other than rules. This feature increases the speed and efficiency of the system; in particular, the inferencing process.

The inference engine performs only forward chaining. This was determined from the structure of the problem. There is an abundance of related facts and information at the beginning of the problem solving process which in turn accomodates the forward chaining process. The conflict resolution problem is solved by allowing the first rule that is satisfied to be implemented. This necessitates an ordering of the rules. This resolution method was chosen because of the short time frame for delivering a "demo" system. This also facilitates the search through the working memory.

The working memory consists of a list of experiment names. Associated with

72

these names are certain facts that are placed into the knowledge base. These include the power requirements, identification number, priority class, sponsoring agency, duration of the experiment, and the required crew involvement. Using this information, a prioritized list of experiments is generated for utilization during the scheduling phase.

The system has been designed and developed in an open-ended fashion to allow system to be extended with only minor adjustments. It contains, in addition to the knowledge-base system structure, an output interface which is at present for demonstration purposes only. This interface will be detailed later. he system itself is dynamic in that it moves through or between different phases of the problem solution. The phases include preparation, scheduling, operation, and rescheduling.

During the preparation phase the individual experiment information is provided to the system from an external data base source and appropriately stored, also the working memory is organized and then prioritized for the scheduling phase. This is accomplished using a priority scheme developed from user input. In the scheduling phase, the experiments are scheduled under the previously explained heuristic procedure and the schedule is created. The schedule itself is part of the knowledge base and is represented as frames. As experiments are scheduled, the subintervals required by the heuristic procedure are defined by start and stop times of the experiments. For each interval the power available, crew available, and the experiments that are currently on-going are determined and stored. This information is required for the remaining two phases, namely operation and rescheduling. The initial schedule is provided to the user for evaluation in a Gantt chart format with appropriate labels (i.e., experiment identification, start and end times, resource loadings, etc.). The user is then afforded an apportunity to make several types of scheduling changes including changing the planning horizon, manually scheduling experiments, reprioritizing experiments, changing rfesource parameters, etc. Based on nthe changes specified, the system determines whether any rule/constraint conflicts exist and if so performs a rescheduling operation as descr5ibed below to resolve the conflicts. In cases where the specified changes do not allow for conflict resolution, the user is so informed.

The output interface during the operation and rescheduling phases is graphical in nature and menu driven. During the operation and rescheduling phases, the system simulates control of the power source for the experiments, i.e., it turns them off or on at the appropriate times indicated by the schedule and updates all the necessary interface information accordingly. The operation phase has two modes: (1) static and (2) dynamic. In the static mode, the system is capable of displaying a power utilization graph for a two week, one week, one day or six hour period of time. Also, the vital information for each experiment (start time, end time, etc.) can be requested by the user simply by using the mouse and a selection menu. In the dynamic mode, the system uses the output interface (see figure 2) to interact with the user through four basic windows - a current status window, a schedule window, a power curve window and a message window. The current status window shows the current status of all the experiments of a payload at a particular point in time. This is accomplished, as illustrated in the window at the top of the screen in figure 2, by representing each experiment as a numbered box. Reverse video is then used to differentiate the on state. Labels placed within the boxes indicate such statuses as aborted, removed, completed, etc. The schedule window (bottom left of screen in figure 2) displays the names of the experiments on separate lines and uses a Gantt chart format similar to that shown in figure 1 to display the scheduling of each experiment. This window simulates movement through time, i.e. as

73

time passes the bars that represent the experiment move to the left and disappear as the experiment is completed. When the experiment is completed the word "completed" appears next to the experiment name. In the schedule window the experiments are also numbered to provide a cross-reference to the numbered experiment boxes in the current status window. The power curve window (middle right of display screen) plots percent power utilization as it scrolls through time. The remaining window is the message window. This is used for interaction and control purposes.

One of the important capabilities built into this system is its ability to reschedule the experiments when deemed necessary. This is one of the main differentiators of this system when compared to others developed for such scheduling applications. The system is capable of determining when it is necessary to reschedule. When such a determination is made, the experiments affected are identified and removed from the active schedule. A rescheduling is conducted and the new schedule is implemented (i.e., made active). There are, based on the initial problem description, a limited number of occurences that would warrant a reschedule. These include an experiment failure, an experiment abort, a power allotment increase or decrease, or the announced arrival of orbital docking and/or servicing vehicles. The first two occurrences require an automatic rescheduling while the others require the system to check working memory and the knowledge base to determine if rescheduling is in fact necessary. Thus we see the system is capable of moving between the different phases, capable of recognizing where it is and what knowledge is applicable, and dynamic in its ability to genenrate and maintain a schedule that will accomplish the objective or objectives of the mission as specified by mission
planning specialists.

5. FUTURE ENHANCEMENTS

While the system demonstrates the potential of using a knowledge base system approach in the area of scheduling, there are several enhancements that have been identified and are currently being implemented to improve the performance and capabilities of the system. First and foremost is that more experiential knowledge needs to be added to the knowledge base. Sessions have been scheduled with the appropriate NASA personnel to begin the task of knowledge engineering [2] [3] [11] [18]. Also, knowledge concerning the determination of alternatives to the schedule instead of just developing a single initial schedule will be added. This will provide the system with the capability of helping NASA personnel in satisfying the dynamic objectives experienced during a mission and will also facilitate the rescheduling process. An example of such an objective is when power allotment reduction forces the schedule to run past the end-of-mission time. Having "knowledge" of alternatives, the system will have a better understanding of which experiments to schedule. Should it continue with the normal rescheduling rules or does some special set of priorities apply? Not only will more objectives be handled in the enhanced system, but the system will be able to handle more constraints, (e.g., fluctuating power requirements of experiments, orientation of the experiments, etc.). Another area of knowledge enhancement concerns the rescheduling function. It has been determined that the system should be capable of performing a quick-fix reschedule when necessary. This will provide the necessary time to perform a more detailed and thorough reschedule in the event of an emergency situation where a temporary "quick fix" is necessary.

The second area of improvement and enhancement to the system is efficiency. Not only is the efficiency of the code being considered, but a more efficient and effective method for searching the schedule and determining experiment slots is

74

under development. This search process, as was mentioned previously, is complicated by the scheduling of intermittant experiments early in the scheduling process. In particular, one experiment in the sample data is required to be scheduled ten minutes out of every hour that the mission is operating. Under the present system this creates 336 additional time intervals that might have to be checked for power and crew availability in determining a feasible interval for a later experiment. The present system takes 15 minutes to schedule the 47 test experiments. The majority of this time is due to the early scheduling of intermittent experiments. Another efficiency enhancement is for the system itself to determine the mission time horizon. This will reduce any excess time that is added in order to accomodate all the experiments. At present, the time horizon is driven by the number of experiments requireing crew involvement and the number of crew available to work with the experiments. A critical path algorithm is being investigated for application in this area.

The final area of enhancements is in the user interface, both input and output interfaces. On the input side, a query/answer system will be added to allow for easy input of experiment data and knowledge base maintenance. This interface will have a limited, natural language parser [10] [12] and will exploit the use of graphics. On the output side several enhancements will be made. First, the system will have an explanation capability for how and why it chose the schedule it is recommending. This capability will soon be provided since the system is currently being redone using an expert system shell which provides how and why facilities. Also, a hardcopy capability for printing out the schedule in "readable" form will be added. Currently the schedule is only stored in symbiolic form. These output enhancements will facilitate the evaluation of system performance. This should allow the system in turn to gain user acceptance more quickly and will also help facilitate the implementation phase.

6. CONCLUSIONS

This paper has detailed a knowledge-based system for solving the NASA space station payload/experiment scheduling problem. The problem is representative of a larger class of dynamic scheduling problems which, for the most part, have been ineffectively handled using more traditional numeric techniques. An expert systems approach allows one to effectively deal with the dynamics and incomplete information which characterize this class of problem. Still in prototype form the system is meeting with wide acceptance and interest not only from the sponsoring agency, but also from other independent sources.

The interest this project has received indicates that there is potential for further research in this area. The wide problem domain encompassed by dynamic scheduling provides many areas for future applications (e.g., project scheduling, production scheduling, manpower scheduling, etc.). Additionally, as systems are implemented and knowledge engineering continues, there is a good likelihood that commonalities will be established across various scheduling applications. This would allow devlopment of an expert system shell for such problems. Such a shell would allow scheduling systems to be readily developed and implemented.

REFERENCES

1. Baker, K. R., Introduction to Sequencing and Scheduling, New York: John Wiley and Sons, Inc. 1974.
2. Barr, A. and Fiegenbaum, E., The Handbook of Artificial Intelligence, Volumes I and II, William Kaufmannm Inc., 1982.

3. Davis, Randall, and Douglas B. Lenat, Knowledge - Based Systems in Artificial Intelligence, New York: McGraw - Hill, Inc., 1982.

4. French, S., Sequencing and Scheduling: An Introduction to the Mathematics of the Job Shop, New York: John Wiley and Sons, Inc., 1982.

5. Harmon, Paul, and David King, Expert Systems, New York: John Wiley and Sons, Inc., 1985.

6. Johnson, L. A. and D. C. Montgomery, Operations Research in Production Planning, Scheduling and Inventory Control, New York: John Wiley and Sons, Inc., 1974.

7. Lawler, E. L., J. K. Lenstra and A. H. G. Rinnooy Kan, "Recent Developments in Deterministic Sequencing and Scheduling: A Survey," in M. A. H. Dempster et. al. (Eds.), Deterministic and Stochastic Scheduling, Reidel, Dordrecht, 1982, 35-73.

8. Lehner, P. E. and Donnel, M. L. "Building Decision Aids: Exploiting the Synergy Between Decision Analysis and Artificial Intelligence," Paper at ORSA/TIMS, San Francisco, May 1984.

9. Little, John D. C., "Research Opportunities in the Decision and Management Sciences", Management Science, Vol. 32, No. 1, January 1986.

10. Rauch-Hindin, Wendy, "Natural Language: An Easy Way to Talk to Computers," Systems & Software, January, 1984, pp. 187-230.

11. Riesbeck, C. K. and Roger Schank, "Comprehension by Computer: Expectation-based Analysis of Sentences in Context," in W. J. M. Levelt and G. B. Hores d'Arcais (Eds.),Studies in the Perception of Language. Chichester, England: John Wiley and Sons, 1976, pp. 247-294.

12. Rich, E., Artificial Intelligence, New York: McGraw-Hill, Inc., 1983.

13. Sprague, R., "The Role of Expert Systems in DSS," Paper at ORSA/TIMS, Dallas, Nov. 1984.

14. Symbolics software. Report, Symbolics, Inc., 21150 Califa Street, Woodland Hills, California, 1981.

15. Tersine, Richard J., Production/Operations Management: Concepts, Structure, and Analysis, (2nd ed.), New York: North-Holland Press, 1985.

16. Turban, E., and King, David, "Building Expert Systems For Decision Support," DSS-86 Transactions, Jane Fedorowicz, Editor, 1986.

17. Turban, E., and Watkins, P., "Integrating Expert Systems and Decision Support Systems," MIS Quarterly, June 1986.

18. Waterman, Donald A., A Guide to Expert Systems, Reading, Massachusetts: Addison-Wesley Publishing Company.

19. Winston, Patrick H., Articial Intelligence, (2nd ed.), Reading, Massachusetts: Addison-Wesley Publishing Company, 1984.

20. Winston, P. H., and Horn, B. K. P. LISP, Reading, Massachusetts: Addison-Wesley Publishing Company.

LABORATORY MODULE - SPACE STATION

DYNAMIC PAYLOAD SCHEDULER

SUBSYSTEMS:

NAME	POWER WATTS	PRIORITY CLASS	AGENCY	DURATION	CREW
ECLSS	6200	I	NASA	CONTINUOUS W/20 MIN. LAPSES OKAY EVERY 4 HRS.	0
COMMUNICATIONS	1480	I	NASA	CONTINUOUS	0
THERMAL CONTROL	600	I	NASA	CONTINUOUS-REDUCES LINEARLY TO 400W FOR 10 KW POWER LEVELS	0
HOUSEKEEPING(MISC)	6000	I	NASA	CONTINUOUS	1

PAYLOAD/EXPERIMENTS:

NAME	POWER WATTS	PRIORITY CLASS	AGENCY	DURATION	CREW
DOD/PAYLOAD 1	890	II	DOD	48 HRS	1
ESA PAYLOAD 1	1845	II	ESA	214 HRS	1
IPS	165	II	ESA	240 HRS	1
ELECT DIAG STA	435	II	NASA	10 MIN OF EVERY HR	1
IECM	480	III	NASA	200 HRS	0
CRNE	930	IV	U.K.	240 HRS	0
GEN PURPOSE COMP	383	III	NASA	CONTINUOUS	1
SOLID POLYMER ELECT	415	IV	3M	36 HRS (5 MIN/HR) 1-5 MIN. EVERY 3 HRS	
IEF	125	IV	NASA	6 HRS	0
MLR	350	IV	U/IOWQ	20 HRS	0
FES-VCGS	600	III	NASA	15 HRS	0
ROTI	36	IV	UAH	43 HRS	0
SEM	2648	IV	NASA	6 HRS	0
RTG	94	IV	NASA	12 HRS	0
TAPE RECORDER 1	85	II	NASA	CONTINUOUS	0
TIME CODE GEN	32	II	NASA	CONTINUOUS	0
MASS SPECTROMETER	215	IV	JAPAN	2 HRS	0
TOOL CHARGER	50	II	NASA	CONTINUOUS	0
FILM PROCESSOR	163	II	NASA	1 HR/DAY	1
SUPER FURNANCE	7840	II	G.E.	32 HRS	0
SILICON WAFER PROD	4760	III	INTEL	14 HRS	0
TAPE RECORDER 2	85	III	NASA	CONTINUOUS	0
TGA	612	IV	ESA	8 HRS	0
MEA	1800	IV	NASA	14 HRS	0
WELDING EXP	1610	IV	NASA	4 HRS	0
CFES	890	III	NASA	36 HRS	0
3-AAL	500	IV	NASA	10 HRS	0
EML	420	IV	NASA	2 HRS	0
GFFC	375	IV	NASA	6 HRS	0
ADSF	480	IV	NASA	48 HRS	1
ARC	215	IV	NASA	8 HRS	0
SAFE	400	III	NASA	15 HRS	1
SOLAR OBS	375	II	NASA	ORBIT/DAYTIME ONLY FOR 36 ORBITS	0
LIGHTNING DET	125	IV	NASA	ORBIT NIGHTTIME ONLY FOR 12 ORBITS	0
CRYSTAL GROWTH	1200	II	NASA	1 HR	0
COMET SEARCH	650	III	JAPAN	ORBIT NIGHTTIME ONLY FOR 40 ORBITS	0
LIFE SCI 1	135	III	A&M	36 HRS	0
LIFE SCI 2	1145	III	UAB	22 HRS	0
LIFE SCI 3	842	III	UAB	66 HRS	0
CLASSIFIED 1	1300	II	DOD	8 HRS	0
CLASSIFIED 2	645	II	DOD	18 HRS	0
MAPPING (WEATHER)	300	III	USWS	CONTINUOUS (CAN BE INTERRUPTED ANYTIME)	0
MAPPING (GEO)	690	III	USGS	60 HRS	0
ORBITER DOCKING	6500	**	NASA	24-72 HRS, WILL BE GIVEN 6 HRS NOTICE	0
ORBITER SERVICER	2400	**	NASA	4-10 HRS, WILL BE GIVEN 2 HRS NOTICE	0

** WILL BE GIVEN TOP PAYLOAD PRIORITY WHEN NEEDED
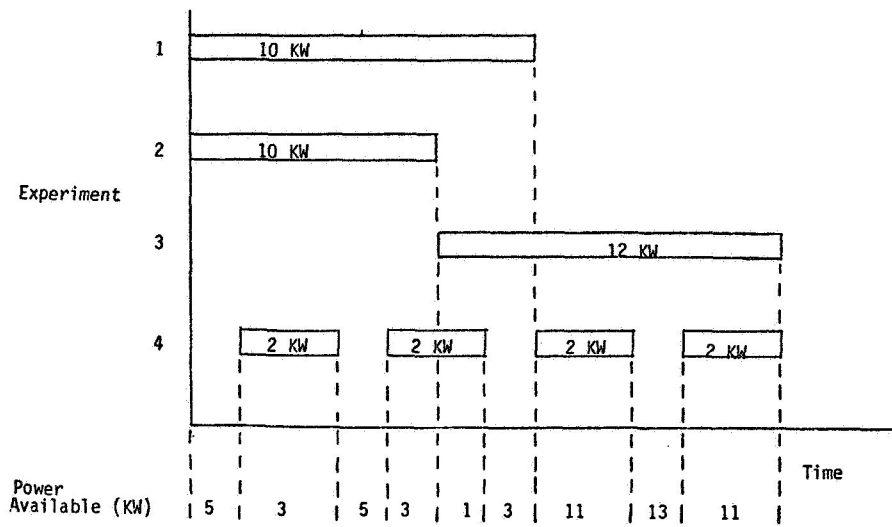
Table 1. Experiment Data
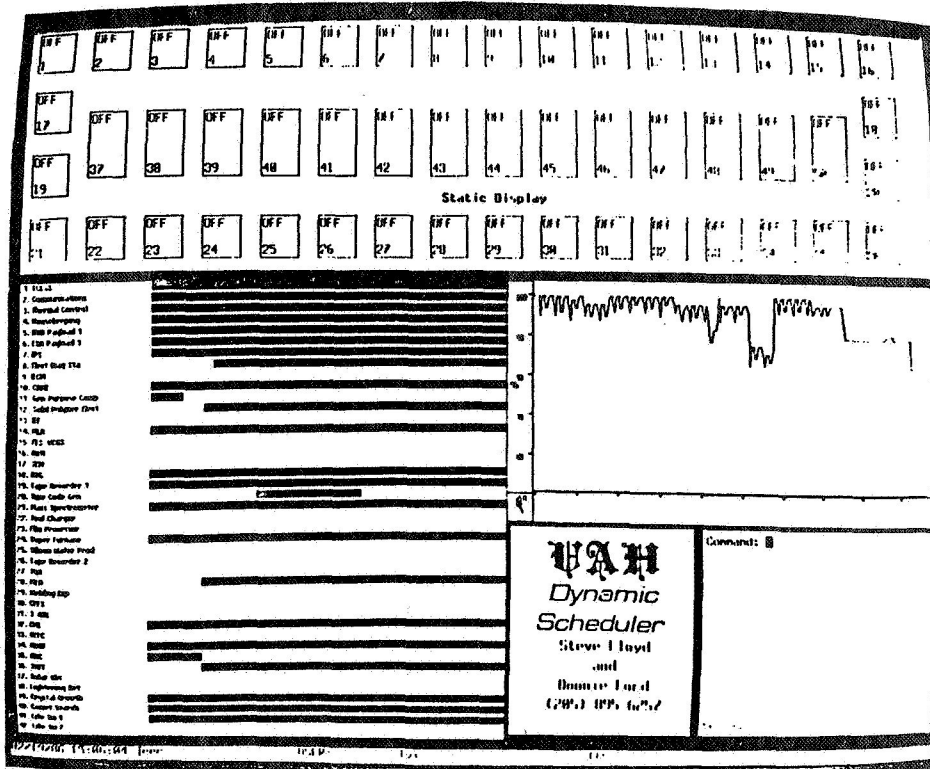
Figure 1. Example Schedule

Figure 2. Scheduler Display Screen

ARTIFICIAL INTELLIGENCE APPLIED TO PROCESS SIGNAL ANALYSIS

Dan Corsberg
Idaho National Engineering Laboratory
EG&G Idaho, Inc.

Many space station processes are highly complex systems subject to sudden, major transients. In any complex process control system, a critical aspect of the human/machine interface is the analysis and display of process information. Human operators can be overwhelmed by large clusters of alarms that inhibit their ability to diagnose and respond to a disturbance. Using artificial intelligence techniques and a knowledge-based approach to this problem, the power of the computer can be used to filter and analyze plant sensor data. This will provide operators with a better description of the process state. Once a process state is recognized, automatic action could be initiated and proper system response monitored.

INTRODUCTION

This paper begins by describing a knowledge-based signal analysis system developed for a nuclear power application. The description includes a discussion of how the approach used is generally applicable to any type of process or dynamic system. The next section explains how this approach could be incorporated into a 'shell'. This 'shell' would minimize the need for knowledge engineers by allowing process and instrumentation engineers to directly incorporate their expertise and knowledge about a process into the signal analysis system. The final section of this paper provides an in-depth look at how to extend this knowledge-based approach into a more complete diagnosis and control system. The promise of this approach is being realized today in both nuclear power and in chemical processing. Full development and extension of the approach could generate expert control systems capable of autonomous operation.

THE ALARM FILTERING SYSTEM

In recent years considerable effort has been made in developing operator decision aids for the nuclear power industry. Several tools have been proposed or implemented in such systems as DMA (Diagnosis of Multiple Alarms),[1] STAR,[2] and DASS (Disturbance Analysis and Surveillance System).[3] These systems use logic trees or cause-consequence trees to identify and emphasize information. These trees, in turn, are difficult and expensive to build, tend to be very inflexible to changes, and are not easily maintained over the life of a plant.

A new system, called the Alarm Filtering System (AFS),[4,5] utilizes artificial intelligence techniques and knowledge-based heuristics to analyze alarm data from process instrumentation and respond to that data according to knowledge encapsulated in objects and rules. AFS was originally developed for the Advanced Test Reactor at the Idaho National Engineering Laboratory. It is currently being installed into a chemical processing plant and into the design specification for the Advanced Test

Reactor control room update. The system filters alarm data, and the most important alarms and information are emphasized to operators during major transients. Alarms not applicable to current process modes are eliminated, while standing alarms resulting from maintenance or unusual operating conditions are inhibited and de-emphasized. Using functional relationships in hierarchical rulesets, AFS performs the following:

o   Generates a description of a situation implied by combinations or sequences of alarms,

o   Suppresses display of information that confirms or is a direct consequence of a previously described situation,

o   Emphasizes alarms that do not fit previous conclusions or alarms that are expected (due to previous alarms or conditions) but are not received within specified time limits. These expected alarms are typically the results of automatic system response to a process state or operator action.

The analysis done by AFS is based on an understanding of functional relationships between alarms and states. Currently defined functional relationships are level precursors, direct precursors, and required actions. Each type of relationship has a set of possible responses and decisions that can be made. As an example of this type of knowledge, consider a High Pressure alarm (at 50 psi) and a High-High Pressure alarm (at 100 psi). The High Pressure alarm is a level precursor to the High-High pressure alarm and, because of that, certain things can be inferred about the two alarms. High Pressure should always occur before High-High Pressure. High Pressure's occurrence doesn't necessarily mean that High-High Pressure will occur. This decision-making knowledge is embodied in rules that are generic (in the sense that they don't address specific alarms or process states). Rules on responding to a level precursor relationship are applicable to any two alarms that are so related. Thus, the example given above would support the more general rule that if A is a level precursor to B, then A should always occur before B. These rules (and their knowledge content) remain unchanged during the development of a specific alarm filtering system. Rules in a filtering system for a reactor would be identical to rules in a filtering system for a chemical processing plant.

The portions of AFS that are unique to the process being monitored are the objects representing the alarms and possible process states. Each object contains data about the specific entity that it represents. This separation of knowledge makes the AFS technology very versatile since alarms (or states) can be changed or added and not affect the structure of the decision-making mechanism; only the knowledge that the mechanism uses is affected.

Procedural, object-oriented, and access-oriented, and rule-based programming paradigms are utilized in AFS. The integration of these paradigms provides AFS with a high degree of modularity and adaptability.

80

Rules allow the capture and maintenance of heuristic knowledge about alarm relationships, while the object- and access-oriented programming allows each alarm's representation to act as an independent entity. An object can perform processing on its own, create new processes to analyze other portions of the system, and cause delayed processing to occur based on a temporal or event basis.

While AFS was originally developed for a nuclear power plant, it is generally applicable to any type of process. In the nuclear power application, AFS was used in conjunction with a wide range of system types. These included pressurized liquid and gas, electrical distribution, power generation, heat transfer, instrumentation, and hydraulic systems. AFS is currently being integrated into the display system for a plant that consists of three independent, complex, chemical processes operating in parallel. In this production environment, AFS will be involved with an equally diverse set of system types. As mentioned previously, the analysis done by AFS is based on the functional relationships between alarms. These relationships are what give AFS its generality since they are not dependent in any way on the type of system being monitored.

EVOLUTION INTO A COMPLETE TOOL

A major bottleneck in the implementation of knowledge-based systems has been the acquisition of expert knowledge.[6,7] In the case of AFS, that expert knowledge is held by process and instrumentation designers and engineers. They understand what the relationships are between the alarms and states of a process. Currently, that understanding is drawn out by a knowledge engineer, inserted into AFS, and then checked for validity. This knowledge acquisition methodology is both time-consuming and inefficient. A solution to this problem would be to eliminate that extra step in the acquisition process by allowing the designers and engineers to directly enter their knowledge.

The desire to improve the efficiency of filtering system implementation has been the primary motivation for plans to build a more complete alarm filtering tool. This tool will allow process and instrumentation experts to construct working alarm filtering systems based on their understanding of specific alarm and control configurations. The tool can be thought of as a shell, or environment, into which information about a specific alarm system configuration is entered. The shell then uses that information to construct and operate an alarm filtering system. This is possible because of the previously mentioned separation of knowledge utilized in AFS. The knowledge about how to respond to a type of relationship is not changed from one system to another. The only knowledge that changes is the specific information about the alarm configuration and what types of relationships the alarms have with each other and with process states.

It is envisioned that an end user of this shell would be familiar with some specific process control and alarm system. He would then determine how the alarms are related and what states need to be defined. Once the relationships are set, the expert would use the shell to build an alarm

81

filtering system for the specific process. The building and modifying of the alarm filtering system would be relatively simple since the alarm data being entered or changed do not affect the rules which determine filtering system response. The shell will help the user develop the representation of the alarms (and states) and assist him in establishing needed functional relationships between the newly defined alarm/state and the already established alarm system configuration. It will automatically establish relationships where applicable and will perform error-checking to ensure that valid expressions are entered. The capability for graphically displaying alarm (and state) relationships will be included to aid the users in both defining and understanding the alarm system they are constructing. In the chemical plant application of AFS, the alarm space contains over 400 digital and analog points, and the graphical overview is critical to the implementation and maintenance of the filtering system.

EXTENDING AFS

The knowledge contained in AFS is based on a small portion of designer and operator experience and does not include understanding of the underlying principles of the systems being monitored. This shallow knowledge is actually a distillation of those underlying principles based on the statistical occurrence of what has happened in the past or is expected to happen based on design criteria. This distilled knowledge is a highly effective way of rapidly identifying a situation and, in the case of AFS, can be envisioned as handling a high percentage of the alarm filtering problems encountered. However, when AFS is presented with contradictory data or a situation that has not been anticipated, AFS defaults to 'normal' alarm display. Additionally, if an operator wanted an explanation as to why AFS reached a conclusion, the only reasoning that AFS could provide would be based on shallow knowledge about functional relationships between alarms rather than on the physical principles that the operator is probably interested in.

To handle these more difficult problems, provide better explanation capabilities, and allow for the verification of conclusions, AFS would need to be incorporated into a much larger environment of cooperating knowledge-based systems. AFS, in its present form, would act as a base for the environment, handling most of the alarm situations and passing conflicts, explanation requests, and verification data up to other knowledge-based systems that would model and understand the principles of the processes being monitored by the alarm system. As an example, there would be a system that understood general principles about heat transfer processes. It would be applied to the specific model of a component cooling process. This knowledge about general principles could be termed 'deep' (at least in comparison to the knowledge currently used by AFS) and would be useful for handling the more difficult tasks. The reason that these types of systems are not used to handle all tasks is because of the need to keep the AFS response time to near real time. Processing this deeper knowledge will take more computational effort and more time to arrive at a useful conclusion.

A key to the successful implementation of a system such as this would be maintaining an object-oriented approach. As mentioned above, the

object-oriented approach provides a high level of independence between the processing associated with each alarm's activation. This independence would be important for maintaining the cooperation between the various subsystems, but would also lend itself to using multiple processors for the environment as a whole and even for the specific subsystems themselves. A logical next step would be to distribute the processing over a network of cooperating processors and implement a blackboard system for overall control and analysis.

As the knowledge contained in AFS increases, AFS will gradually lose its generality of application. The current version of AFS can be applied to a wide range of system types because the filtering system bases its analysis on very shallow knowledge about the relationships of alarms and process states. This knowledge about relationships is not process specific. Once the depth of knowledge is increased, new knowledge is added that is specific to the physical principles of the systems being monitored. While some of this knowledge would be general enough to be used in both a nuclear plant's heat transfer system and a space station's heat transfer system, there would be much information that would be particular to the application (although the knowledge about the space station heat transfer system could easily be usable for a satellite's or a shuttle's heat transfer system).

CONCLUSION

AFS has simplified the interface between plant and operator while simultaneously attacking the alarm-filtering problem with new techniques and ideas. As a system, AFS integrates rule-oriented programming into an object-oriented environment to avoid exhaustive searches of extensive databases or structures. Since the only rules checked and only objects referenced are those directly related to the event being processed, AFS's computational requirements per event are not dependent upon the alarm space size, but rather upon the relational complexity of those alarms. The use of an object-oriented alarm model has also resulted in AFS having a high degree of flexibility. Once the functional relationships of the alarms and states have been defined, a working system can easily be built and maintained.

A major weakness of AFS is that its analysis is presently limited to the localized relationships between alarms. Some abstraction has been introduced through the use of states, but higher levels are needed to assist in identifying plant conditions. It is felt that the technology used in AFS provides a good foundation for taking a layered approach to the task of analyzing alarms (for a description of this type of an approach, see Reference 8). The use of alarms and states is envisioned as an effective way of handling a high percentage of expected alarm sequences. AFS already recognizes the points at which further processing would be needed and those points would provide "hooks" for deeper knowledge processing. The next step in the evolution of AFS would be the development of a blackboard type of control structure to handle several cooperating systems that would be processing at various levels of abstraction.

REFERENCES

1. M. M. Danchak, <u>Alarms Within Advanced Display Streams: Alternatives and Performance Measures</u>, NUREG/CR-2276, EGG-2202, September 1982.

2. L. Felkel, "The STAR Concept, Systems to Assist the Operator during Abnormal Events," <u>Atomkernenergie, Kertechnik</u>, Vol. 45, No. 4, 1984, pp. 252-262.

3. A. B. Long, R. M. Kanazava et al., <u>Summary and Evaluation of Scoping and Feasibility Studies for Disturbance Analysis and Surveillance Systems (DASS)</u>, Topical Report EPRI NP-1684, December 1980.

4. D. R. Corsberg and D. Wilkie, "An Object-Oriented Alarm-Filtering System," <u>Power Plant Dynamics, Control, and Testing Symposium, Knoxville, Tennessee, April 14-16, 1986</u>.

5. D. R. Corsberg, "Extending An Object-Oriented Alarm-Filtering System," <u>Expert Systems In Government Conference, Washington, D.C., October 20-24, 1986</u>.

6. F. Hayes-Roth, "The Knowledge-Based Expert System: A Tutorial," COMPUTER, September, 1984, pp 11-28.

7. C. J. Culbert, "Expert System Building Tools," NASA Memorandum FM7(86-19), February 11, 1986.

8. D. R. Corsberg and D. Sebo, "A Functional Relationship Based Alarm Processing System For Nuclear Power," <u>International ANS/ENS Topical Meeting on the Operability of Nuclear Power Systems in Normal and Adverse Environments, Albuquerque, New Mexico, September 29 - October 3, 1986</u>.

C-2

COUPLING EXPERT SYSTEMS AND SIMULATION

K. Kawamura, G. Beale, S. Padalkar, J. Rodriguez-Moscoso, and B.J. Hsieh
Center for Intelligent Systems
Vanderbilt University, Nashville, TN 37235

F. Vinz and K.R. Fernandez
NASA George C. Marshall Space Flight Center

ABSTRACT. This paper describes a prototype coupled system called NESS (NASA Expert Simulation System) [1]. NESS assists the user in running digital simulations of dynamic systems, interprets the output data to determine system characteristics and if the output does not meet the performance specifications, recommends a suitable series compensator to be added to the simulation model.

INTRODUCTION

Recently expert systems technology is gaining popularity in the engineering community [2]. Application areas range from diagnostic and repair to design assistance. Common rule-based expert systems, however, due to their focus in symbolic reasoning, are often not well suited to deal with numerical processing. In contrast, numerical techniques can process a great amount of number crunching, but cannot provide insights into the problem solving process and interpretation of the results. Therefore, there is a need to couple symbolic and numerical techniques [3]. Such coupled systems promise to integrate the explanation and problem solving capabilities of expert systems with the precision of traditional numerical computing.

One application area where this coupling of symbolic and numerical computing will have great benefit is the simulation of aerospace vehicles. The simulation programs are typically large, run with different parameter values, and run by different users. The programs are used to evaluate the performance of the vehicle under various environmental conditions, with different values for parameters, and with different control algorithms. An expert system can serve as an intelligent user interface and can interpret the simulation results with respect to users' goals.

DESIGN PRINCIPLE

The rationale for using an expert system in conjunction with the digital simulation of a dynamic system is that it takes specialized knowledge to obtain an accurate simulation of a complex system. Once an acceptable simulation of the given system is achieved, the performance of the system must be evaluated relative to its goals. If the performance is not satisfactory, then modification to the system through the addition of compensating elements might be required. The knowledge necessary to simulate the system accurately, interpret the performance in terms of goals, and recommend compensators might not be directly available to the user. An expert in the domain of the system to be simulated may not have the

detailed knowledge in numerical methods and control system synthesis.

One feature that makes this application different from the usual simulation problem is that the program which provides the numerical integration algorithms and performs the. system analysis and compensator design has no information on the dynamics of the system being simulated. The only knowledge that can be used during execution of the simulation and design of the compensator comes from input/output measurements made during the simulation. The expert system is therefore dealing with a "black box" during the simulation, and the results of analyses and compensator designs will reflect this.

The design principle for NESS is that software code representing the system to be simulated, along with a glossary of state variables, parameter values, and time constraints, can be generated under the user's direction. NESS would then: (1) determine the methods of numerical integration to be used; (2) control the execution of the simulation; (3) present and interpret the results; and (4) recommend any compensators required to achieve the desired performance. The user can then incorporate the modifications into the code for the system model and evaluate the improved performance through another simulation session. Therefore, the time required for the iterative design and verification process can be shortened through the use of intelligent simulation since the user is assured of as accurate simulation of the given mathematical model of the system and is provided with recommendations on how to improve the system performance.

Figure 1 is a block diagram which illustrates this design principle. The simulation model is the software code implementing the equations which mathematically model the actual system. This code is generated from the user's description of the system by a simulation designer or programmer.
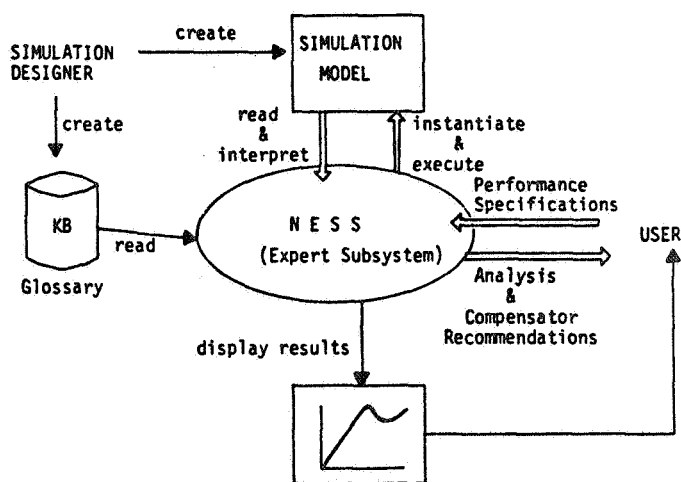

Figure 1. Design Principle

Also created at this time is the glossary. This is a knowledge base about the system which specifies the state variables in the system, initial values for the state variables, values for system parameters, and con-

straints on the size of any of the time steps used in the simulation. NESS reads this knowledge base, creates an interface with the simulation model, and instantiates the model with the values from the knowledge base. In general, this interface will include the numerical integration algorithms for all of the state variables. The user can input the system performance specifications directly into NESS in an interactive manner. Once the model is instantiated, the simulation may be performed under the direction of NESS. NESS can proceed in either an automatic or a manual mode. In the automatic mode, NESS performs as many actions as possible without further user interaction, including compensator design. In the manual mode, the user is given the option at each step either of agreeing with a suggestion by NESS or of providing direction to NESS.

NESS is able to simulate the system in either the time domain or the frequency domain. The results of the simulation are interpreted by NESS relative to the performance specifications supplied by the user. For example, the transient performance can be described by the percent overshoot in the time domain or the phase margin in the frequency domain. Steady-state accuracy to step or ramp inputs can be obtained directly in the time domain or inferred from frequency domain data. Based on the performance analysis, the user would be told that all specifications are met or that some specifications are not satisfied. In the latter case a lead, lag, or lag-lead compensator would be recommended, depending on the type of performance error.

SYSTEM ARCHITECTURE

The system architecture is composed of two subsystems: one performing symbolic manipulations, called the expert subsystem, and the other performing numeric computations, called the numeric subsystem (Figure 2). The
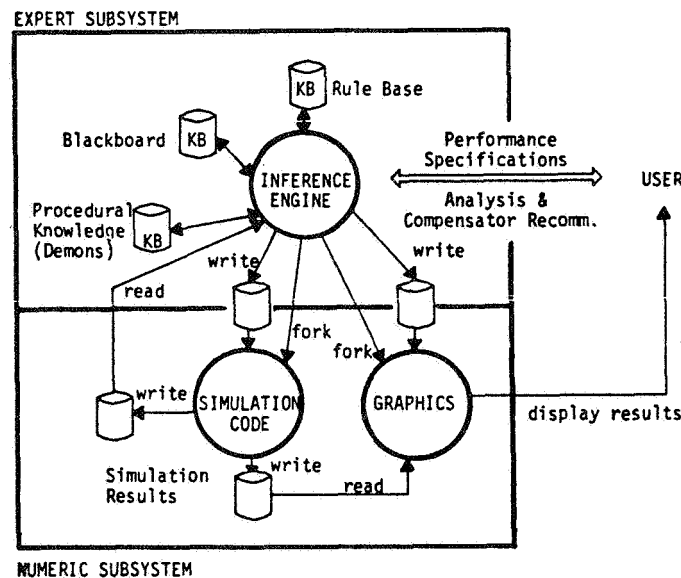

Figure 2. NESS Architecture

expert subsystem is written in FRANZ LISP [4] and uses an inference engine named GENIE (GENeric Inference Engine), developed at Vanderbilt University [5].

The numeric subsystem is composed of a set of Fortran-77 subroutines and handles any application code supplied by simulation designers. The package provided by NESS includes an integrator module, an analysis module, and a graphics module.

Expert subsystem. The expert subsystem contains knowledge and intelligence required to assist the user in instantiating and running a simulation model. It also has the capability of displaying the simulation results and interpreting their results. On discovering some unwanted results, it can design compensators that can be added to the model to achieve the desired results.

The expert subsystem is composed of seven modules, each containing knowledge in the form of rules and demons, required to carry out a specific function. A blackboard is used to communicate knowledge and data among them.

Numeric subsystem. The numeric subsystem contains four modules to generate the executable simulation code from the user-supplied model and to display time- and frequency-domain plots generated from the simulation output files.

Coupling expert and numeric subsystems. The expert subsystem uses the FRANZ LISP "*process" function to fork a descendent process which runs the executable simulation program in the numeric subsystem. Before forking the descendent process, the expert subsystem creates two files containing the simulation model parameter values. When the simulation run is completed, the numeric subsystem writes the results to a file which in turn is read by the expert subsystem.

SIMULATION RESULTS INTERPRETATION

The function of the Simulation Interpreter is to interpret results of a simulation. If the results do not indicate a stable simulation, an iterative procedure is followed to try and obtain stability. If the simulation is stable, the performance is compared with user-supplied or default specifications. If the specifications are satisfied, results are presented to the user; otherwise a compensator is designed. At the outset the user is asked whether he would like NESS to automatically execute the simulation and interpret the results. If the user responds yes, then NESS will make decisions and perform actions without any further user input and present the user with a simulation that meets specifications or a simulation along with a suggested compensator that will meet specifications. If the user does not feel comfortable with the intelligence residing within NESS, NESS will inform him of all future decisions and seek his approval of all future actions.

System stability. A simulation run is classified to be one of three types: error-prone, not-yet-reached steady state, and stable. An error-prone simulation run is one in which a Fortran run-time error occurred. The other two types are self explanatory.

A larger-than-desired value for step size sometimes results in an overflow or an underflow condition leading to a run-time error during a

88

simulation. To try and achieve a stable simulation, NESS reduces the step size and reruns the simulation.

A larger-than-desired value for step size and/or a smaller-than-desired value for the simulation final time leads to a simulation that has not yet reached a steady-state condition. NESS reduces the step size and increases the simulation final time and reruns the simulation.

When a simulation run is found to be stable, NESS tries to ensure an accurate simulation by reducing step size and rerunning the simulation until the results from two such runs are close to each other. Then NESS looks at the generated results and if they match the user's specifications, it displays them. If the generated results do not match the user's specifications, a compensator is suggested.

Compensator design. For a stable simulation when the system performance does not yet meet the NESS default performance or the user desired system specification, NESS can suggest a compensator. Classical textbook design procedures are used for phase lead or phase lag compensators with either frequency domain or root locus techniques. The assumption that the system response is dominated by a pair of closed-loop poles is made. Typical rules for determining the type of compensator needed are:

a. If the refinement of transient response is required, a phase-lead compensator will be suggested.

b. If the steady-state accuracy needs improvement, a phase-lag compensator will be recommended.

TEST MODEL

In order to test NESS' simulation and analysis capabilities, a spacecraft attitude control system was run by NESS. The generic simulation program [6], the software for encoding the spacecraft attitude control problem, offers the means for simulating the attitude control problem of a spacecraft vehicle. One interesting feature about the control of a space vehicle is that several time-varying coordinate frames are required when maneuvering the vehicle into a certain orientation, and maintain it over an extended period of time. Figure 3 depicts a block diagram of this model. This model is composed of six basic modules: a generic command generator, a proportional-integral-derivative (PID) controller, actuators, body dynamics, sensors, and quaternion modules.

The command generator module injects to the system, not only step, ramp, or sinusoidal type of inputs, but also is able to inject those signals described as continuous scans, peak-up maneuvers, in-track or cross-track offsets, and dwell scans [6]. In our discussion, only the ramp and sinusoidal types are considered. The PID controller generate the three axis torque command signals required to drive the vehicle according to the generic command generator outputs. A digital PID controller is implemented based on the Tustin transformation operator [7]. The actuator module is included during the simulation when the effect of the small signal nonlinearities due to reaction wheel assemblies, magnetic torquers, or steering law for control moment gyros need to be considered. The body dynamics
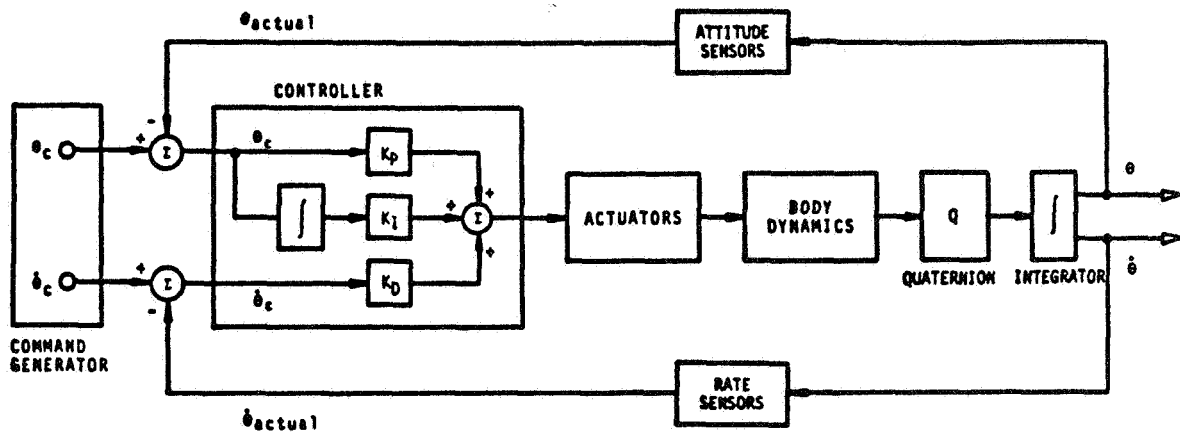
Figure 3. The Generic Simulation Block Diagram

module includes rigid and flexible body rotational effects. The rigid body rotational equations are solved in vehicle coordinates in terms of the rigid body rate, as a function of the reaction wheel rate, gravity gradient torque, reaction wheel output torque, vehicle and reaction wheel inertia matrices. The flexible body rotational equations are derived from a single bending mode from actuator torque to bending rate. The sensor module provides a model of the measurements for both the angular position and angular rate (star trackers and rate gyros) which are available to the control system. Finally, the quaternion block is included to solve the set of differential equations that represent the rate of change of the vectors that specify the vehicle's attitude.

CONCLUSIONS

This paper has illustrated the capabilities of a prototype coupled expert system called NESS. One of the major goals of this project was to demonstrate the feasibility of applying expert systems technology to the area of digital simulation of dynamic systems. It has been demonstrated that an expert system can interact with the user to determine the simulation goals, instantiate parameters with the proper values, execute the simulation, and interpret the results relative to performance specifications. It has also been shown that it is feasible for an expert system to intelligently select integration methods and time steps for the simulation, even when little is known about the system to be simulated. This is a difficult task which requires trial and error even for a human expert. More work needs to be done in this area to improve the robustness of the knowledge. NESS has also shown itself to be capable of designing simple compensators to improve the performance of the system. The ability to design appropriate compensators would be greatly improved with more information available about the system being simulated. Additional knowledge needs to be added to expand the capabilities of the compensator design procedures and to allow NESS to handle a wide variety of simulation programs. Future work is considered to allow NESS operation in a network

90

environment controlling several simulations.

REFERENCES

[1] Kawamura, K., et al. Research on an Expert System for Database Operation of Simulation/Emulation Math Models, Phase II Results, NASA Contract #NAS8-36285, Center for Intelligent Systems, Vanderbilt University, August, 1986.

[2] For example, see IEEE Computer Society, Proceedings of the Second Conference on Artificial Intelligence Applications, "The Engineering of Knowledge-Based Systems," Miami Beach, Florida, December, 1985.

[3] Kowalik, J.S. (Ed.) Coupling Symbolic and Numerical Computing in Expert Systems, North-Holland, Amsterdam, 1986.

[4] Foderaro, J.K., and Sklower, K.L. The FRANZ LISP Manual, The Regents of the University of California, June, 1983.

[5] Sandell, H.S.H., "GENIE User's Guide and Reference Manual," Department of Electrical and Biomedical Engineering, Vanderbilt University, Technical Report #84-003, Nashville, TN, July, 1984.

[6] Rodriguez-Moscoso, J. J., "A Prolog-Based Coupled Expert System for the Simulation of Spacecraft Attitude Control Problem," Masters Thesis, Vanderbilt University, August 1986.

[7] Rosko, J. S., "Digital Simulation of Physical Systems," Addison-Wesley Publishing Co., Reading, Massachusetts, 1972.

Left column:

```
s---s---s  W E L C O M E   T O   N E S S  s---s---s

       NASA EXPERT SIMULATION SYSTEM

Please give me the name of your MAIN file
Only name no extension
>> gsim

Enter name of GLOSSARY file in format 'name.extension'
>> gsim.glo

   Please choose a desired type of response:

      1) step
      2) ramp

   Please enter choice(s):
      # 1

   Please enter the desired system performance for the STEP input:

      1) Delay time  ( 0%-50% of the final output)
      2) Rising time (10%-90% of the final output)
      3) Steady-state-error
      4) Percentage of maximum overshoot
      5) Peak time
      6) None of the above

   Please enter choice(s):
      # 3

      >> Steady-state-error (%) = 0.01

         ---s--- End of GOAL SETTING ---s---

   Please enter desired default state variable value(s):

      1) all
      2) some
      3) none

   Please enter choice(s):
      # 1
   All the initial values for the state variables have been found.

   Please enter desired default values for model coefficients:

      1) all
      2) some
      3) none

   Please enter choice(s):
      # 1
   All the initial values for the model coefficients have been found.

   Please enter the following time parameter(s), or
   hit the 'return' key to set to default value:

   1) Initial time (T0) [Default value = 0.0 (sec)]:
         >>
   2) Final time (TF) [Default value = 50.0 (sec)]:
         >> 20.0

   Do you want NESS to execute and interpret the simulation in an
   AUTOMATIC mode ? (If "NO" then you will be prompted at each major
   decision.)

      1) yes
      2) no

   Please enter choice(s):
      # 1

   The simulation program is now being run

   The Simulation run is successfull

   The simulation steady-state error is 0.011%

   To ensure accuracy of the simulation results the integration rate
   is being divided by 4.0 and the simulation is being rerun.

   The simulation program is now being run

   The Simulation run is successfull

   The simulation steady-state error is 0.009%
```


PLOT INFORMATION

GENSIM-0 T S

X-AXIS: SEC
XMIN = 0.00E+00
XMAX = 20.
XINC = 1.0
SIZE = 0.0625 s

Y-AXIS: RAD
YMIN = 0.00E+00
YMAX = 1.4
YINC = 0.10

HIT THE GR KEY
TO RESUME PLOT.

Right column:

```
Please choose a desired type of analysis

      1) time
      2) frequency

Please enter choice(s):
      # 2

Please enter the desired system performance for FREQUENCY response:

      1) Gain Margin
      2) Phase margin
      3) none of the above

Please enter choice(s):
      # 2

      >> Phase margin (degrees) = 45

         ---s--- End of GOAL SETTING ---s---

Please enter desired default state variable value(s):

      1) all
      2) some
      3) none

Please enter choice(s):
      # 1
All the initial values for the state variables have been found.

Following is a menu of the the possible access point(s) for
freq-domain analysis. Option 1 followed by the state variable
number(s) of your choice allows you to look at the attributes
of the variable(s) chosen.

      1) Description
      2) THETAX
      3) THETAY
      4) THETAZ
      5) BENDIX
      6) BENDIY
      7) BENDIZ
      8) ACTX
      9) ACTY
     10) ACTZ

Please enter choice(s):
      >> 2

Following is a menu of possible frequency domain output points.
Option 1 followed by the state variable number(s) allows you to
look at the the attributes for the state variable(s) chosen.
Or just enter the variable number for output.

      1) Description
      2) THETAX
      3) THETAY
      4) THETAZ
      5) BENDIX
      6) BENDIY
      7) BENDIZ
      8) ACTX
      9) ACTY
     10) ACTZ

Please enter choice(s):
      >> 2

Please enter desired default values for model coefficients:

      1) all
      2) some
      3) none

Please enter choice(s):
      # 1
All the initial values for the model coefficients have been found.

Please enter the following frequency parameter(s),
or hit the 'return' key to set to default value:

1) Lowest frequency (W0) [Default value = 0.1 (rad/sec)]:
      >> 0.1
2) Number of decades [Default value = 3 (decades)]:
      >> 3
3) Number of sampling freqs/decade [Default value = 3 (freq/dec)]:
      >>
4) Number of sampling points [Default value = 512 (points)]:
      >> 1024
5) Final time (TF) [Default value = 50.0 (sec)]:
      >> 30.0

Do you want NESS to execute and interpret the simulation in an
AUTOMATIC mode ? (If "NO" then you will be prompted at each major
decision.)

      1) yes
      2) no

Please enter choice(s):
      # 1

The simulation program is now being run

The Simulation run is successfull

The simulation phase margin is 49.9 degrees.

To ensure accuracy of the simulation results the integration rate
is being divided by 4.0 and the simulation is being rerun.

The simulation program is now being run

The Simulation run is successfull

The simulation phase margin is 51.9 degrees.

The results of the two runs are within 10 percent of each other
and hence the simulation results are accurate.

The simulation phase margin 51.9 degrees is larger than user specified
goal of 45 degrees, therefore no compensator is recommended.

Goodbye. Hope you enjoyed the session.
```
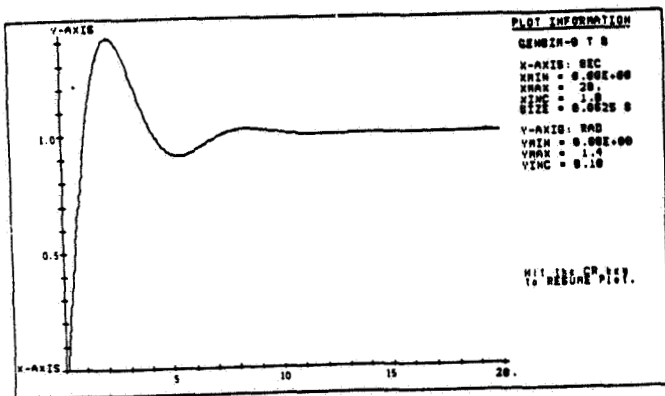

PLOT INFORMATION

GENSIM-0 F U

W-AXIS: RAD/SEC
WMIN = 0.10E-02
WMAX = 0.10E+03
WINC = 0.20
SIZE = 1/1024 s

MAGNITUDE DB
MIN = -20.
MAX = 80.
INC = 20.

PHASE DEGREES
MIN = -.22E+03
MAX = 0.00E+00
INC = 45.

HIT THE GR KEY
TO RESUME PLOT.
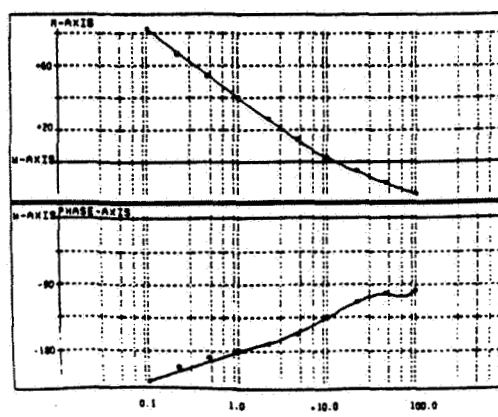
OPTIMIZATION OF LOW GRAVITY MATERIALS PROCESSING EXPERIMENTS USING EXPERT SYSTEMS

Gary L. Workman and Amar Choudry
The University of Alabama in Huntsville

Abstract

The use of an expert system for the control of materials processing experiments in a facility such as the space station provides a number of attractive features for insuring that certain critical process parameters can be used to optimize the productivity of the materials processing experiments. The successful implementation of such a system will require sensory fusion of sufficient depth to enable determination of the status of the critical process parameters, as well as determination of the trends of the parameters, in order that appropriate process alternatives available for optimal control of the experiment can be implemented by the expert system.

The proposed approach to implementation of an expert system utilizes a knowledge base of desired process characteristics which will provide the desired result. The knowledge base for each experiment will be created in conjunction with the scientific investigator in charge of the experiment. Improvements to the knowledge base will be expanded to include self learning sessions in ground based experiments in order to teach the expert system how to respond to perturbations in the process and update the knowledge base on what process change should be implemented in order to reach the desired end product.

This paper deals with design considerations using an on-line real-time expert system such as PICON. The types of experiments evaluated within this paper will include several types of solidification experiments which will be performed on the Space Station.

Introduction

There are a number of valid reasons for studying materials processing phenomena in low gravity. We often hear about the lack of convection due to thermal gradients, the lack of buoyancy forces when working with materials of different densities, and in general that a spacelab facility will provide an environment in which optimal control over experimental parameters is possible. Such an environment can then enable scientific investigators the maximum opportunity to grasp a better understanding of some fundamental processes. Unfortunately there are some exceptions to this concept; for instance, in a manned space station, there will be numerous unintended accelerations caused by people movements and other possible events caused by various motions and impacts within the station. It is these unintended acceleration type phenomena which can impact the beneficial aspects of scientific experimentation in low graavity. Therefore in order to optimize upon the potential results of materials processing experiments aboard the space station facilities, it is important to implement controls to compensate for any undesired events.

Many analog and digital control schemes are available for controlling furnace experiments which basically provide a suitable thermal profile in order to accomplish a desired solidification product from a melted sample. Currently NASA does fly several types in the shuttle laboratory facilities and on the NASA KC-135 airplane. In order to demonstrate the potential for expert systems approach to studying solidification phenomena in low gravity, we have focussed on the directional solidification experimental technique for a number of reasons. Most important is the fact that a large number of scientific experimenters utilize this technique for materials science research and

94

UAH has experience in designing, fabricating, and working with directional solidification furnaces for low gravity research flights aboard the KC-135.

Directional Solidification Experimentation

Most experimentation studying solidification phenomena in the directional solidification technique utilize the technique in order to minimize the number of parameters which vary during the experiment. Casting experiments (bulk solidification) do not provide an as easily controlled thermal environment for the scientist. For this reason, directional solidification has become an important tool in investigating solidification in low gravity. The experimental arrangement shown in Figure 1 also illustrates the fundamental technique. By moving the sample through a hot zone and quenching the melt on the other side, one has an easily controlled thermal environment. The parameters of interest have been identified by many authors with excellent treatments given by Flemings and McLean. The two parameters (besides compositional characteristics) which most affect the resulting solid are the temperature gradient across the solidification zone and the translation rate. Assuming a planar solidification front (which is most easily provided in directional solidification) one can obtain different morphologies by varying the translation rate (V) or the temperature gradient (G) as indicated in Figures 1 and 2. These figures are taken from McLean.

The Automated Directional Solidification Furnace flown by NASA on Shuttle flights is shown in Figure 3. Notice that in practice it is generally preferred to move the furnace (particularly in the case of low gravity research) and not the sample as implied in Figure 1. UAH has also built several furnaces similar to this for flights on the KC-135.

An important characteristic of the furnace is the ability to vary the temperature profile to best fit a particular experiment's needs and thereby have optimal control over the temperature gradient (G) seen by the solidifying sample. Frequently two or three different heating elements are used in order to sharpen up the gradient (high G) versus having one heater providing a lower gradient (G).

Why an Expert System?

In order to visualize the importance of an expert system controller for directional solidification experiments in space we need to consider several attributes of an expert system controller and the needs of the low gravity experiment. An important consequence of low gravity research is that most Space Station experiments will have many different samples from a large number of scientific investigators all using the same furnace. Realizing that each experiment will have different parameters, this means that the furnace controller has to be quite versatile or `robust`. The important characteristics of an expert system controller are the ability to build in "knowledge" for each experiment both in temperature and thermal profiles and in its ability to handle undesired events through inference.

Figure 4 illustrates a conceptual architecture for the anticipated controller which is directly applicable to this effort. This approach is taken from Winter. Notice that solidification begins as a skill with the scientific investigator providing experience in determining how one should proceed with the experiment. As the expert in the process, one can then extract rules for performing the experiment properly and with the knowledge of the furnace parameters (sensory fusion) the optimal controller performs the function accordingly. Notice that artificial intelligence is required (as opposed to simpler digital control

96

algorithms) in order to take the proper action when unintended occurences result. It is interesting to note that an expert system shell already exists which satisfies this architecture and that particular product is PICON.*

Implementation of the Expert System

PICON offers a high level environment for simulating and subsequently implementaing real time process control. The programming in PICON is based on icon-oriented logical objects. Rules and parameters are defined which constitute a property list for the icon. In physical terms these icons represent the various controllers and sensors found in any process control environment. With the help of a graphic software package these icons can be combined, according to a well-defined grammar, to represent the sensor and control schema of a real process. The simulation of the process is done by real-time module called RTIME. It ˋsensesˊ the sensor output in real-time and updates the system ˋstate-vectorˊ at each increment of time. It also refers to the knowledge base and fires various rules which need special attention or cause controller action e.g. "IF TEMP TOO HIGH THEN START FAN".

All the aspects of the simulation environment are screen-editable. Thus one can change the schema or the property list of any controller or sensor including entry points for noise. In this manner a very realistic simulation of the system can be realized. Some of the routines involving the translation portion of the simulation will require LISP programming since PICON currently does not provide that capability. The simulation will provide feasibility of the concept and also allow us to determine how many sensory inputs will be required to accurately determine the temperature gradient of the sample.

*Registered product of LMI.

97

In essence the use of an expert shell like PICON enables us to first simulate graphically the control features of the directional solidification experiment with particular emphasis on translation rates and temperature gradients. The ability to graphically translate the controlled rod is a function not incorporated into PICON. The program PICON runs independent on the LISP processor. When it becomes necessary to simulate in real time, a second processor is used to do the interfacing and updating of a shared memory array which PICON will draw data from. The second processor will provide the data necessary for PICON to graphically display the translating rod. PICON will remain superior to the second process so that the simulation can set or reset the control features of the experiment.

After accumulating sufficient simulation results to allow us to design a proper physical arrangement of the system, we will test the concept on hardware which currently flies on the KC-135. Normally temperature measurements will be obtained with thermocouples, just as we currently do. In this more intelligent approach; however, we will be attempting to better define a true temperature gradient at all times during the experiment by strategically placing more temperature sensors in the sample volume. Additionally, translation and acceleration parameters will be collected for fusion into the data stream. The scientific investigators will provide the information on the desired temperature gradients and then PICON will perform the required thermal and translation control.

Training the Expert Controller

Only part of the knowledge base required for optimization of the directional solidification experiments is defined in terms of set temperature gradients and translation rates. Another part of the

98

knowledge base contains the responses when the system parameters deviate from the intended profile. Through the use of a self-learning mode, the expert system will learn from the scientific investigator what actions to take. In this way the expert system controller can follow the best previously defined procedure to still produce the optimal result in real-time. The self-learning concept will be generated through pre-defined events which might occur during the experiment. The most difficult portion of the concept to implement will be how to handle the unknown events. Current strategy involves use of the PICON inference engine to determine the best procedure to follow. We anticipate that there will have to be some checks in the procedures called from the inference system so that we can make sure that an inferred procedure might not do more harm than good. Experimentation with PICON will enable us to determine the best approach to this problem.

Summary

The materials processing facilities for space facilities can benefit from expert systems control in order to optimize in spite of unexpected events using real-time control schemes such as presented here. Expert system shells such as demonstrated by PICON will play an extensive role in space based control. In addition to directional solidification experiments, many other techniques such as float-zone solidification, phase partitioning, and protein crystal growth can benefit from these approaches if sensory fusion is sufficient to control the experiment dynamically.

References

1.   Solidification Processing, M. C. Flemings, McGraw-Hill, New York, 1974.

2.   Directionally Solidified Materials for High Temperature Service, M. McLean, The Metals Society, London, 1983.

3.   Microstructural Variations Induced by Gravity Level during Directional Solidification of Near-Entectic Iron-Carbon Type Alloys, D. M. Stetanesce, P. A. Curreri, and M. R. Fishe, Metallurgical Transaction A, 17A, 1986.

Acknowledgements

Figure 1.

Schematic illustration of directional solidification of a rod by moving it relative to fixed heating source and cold sink and the steady state temperature profile

(From McLean)
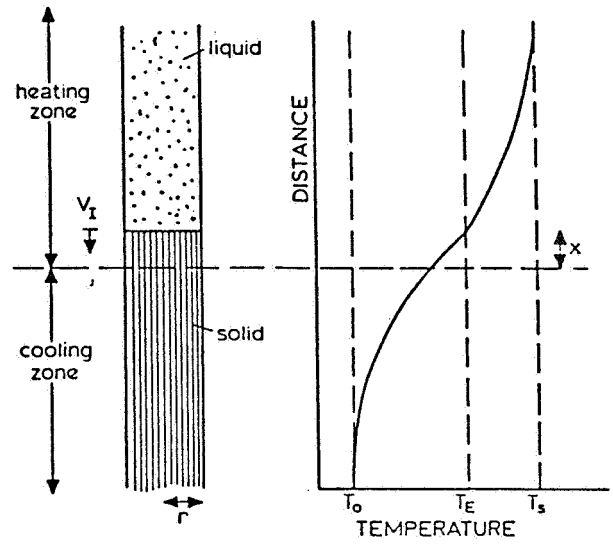
Figure 2.

Schematic illustration of the changing shape of the solidifying front with increasing V_I for a constant G_L as the range of constitutional supercooling extends to the equilibrium melting range of the alloy

(From McLean)
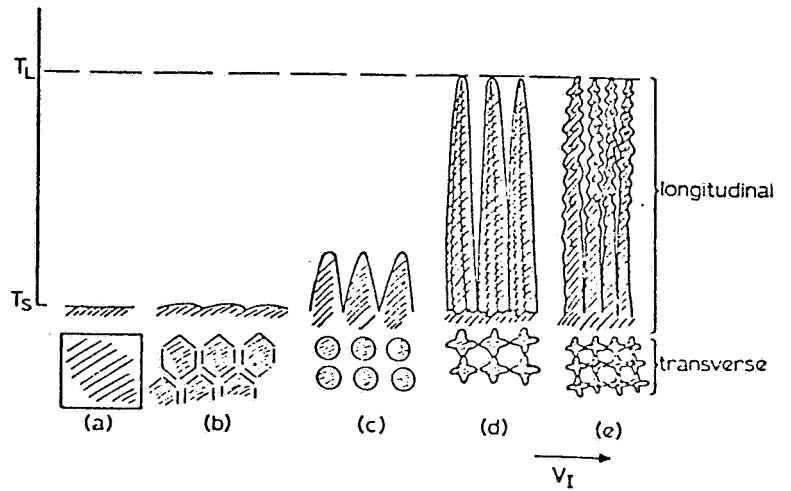
Figure 3. NASA's Automated Directional Solidificational Furnace
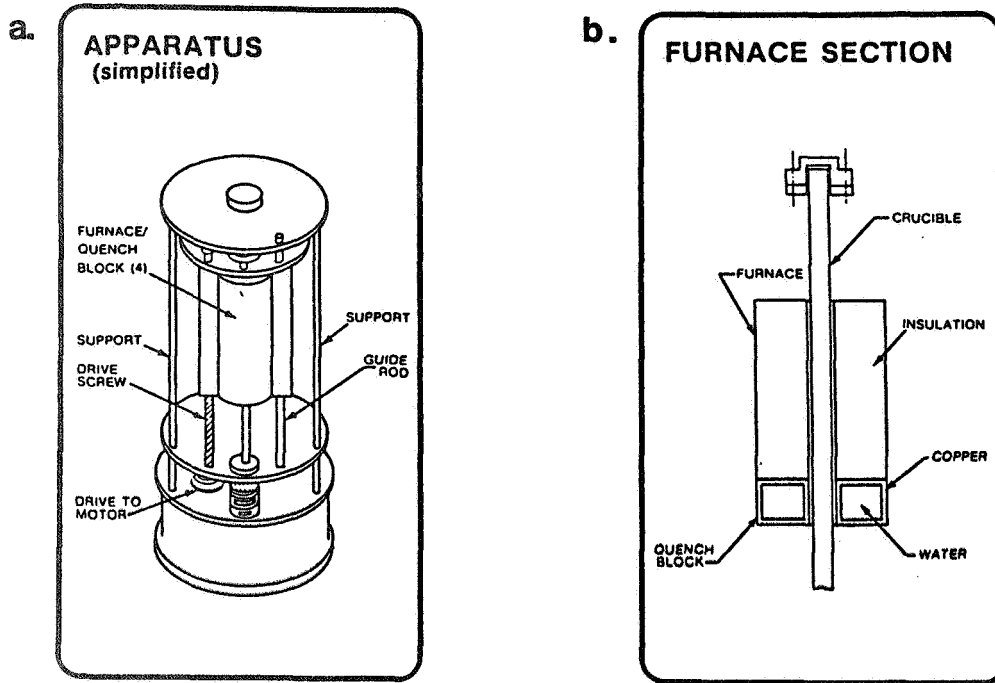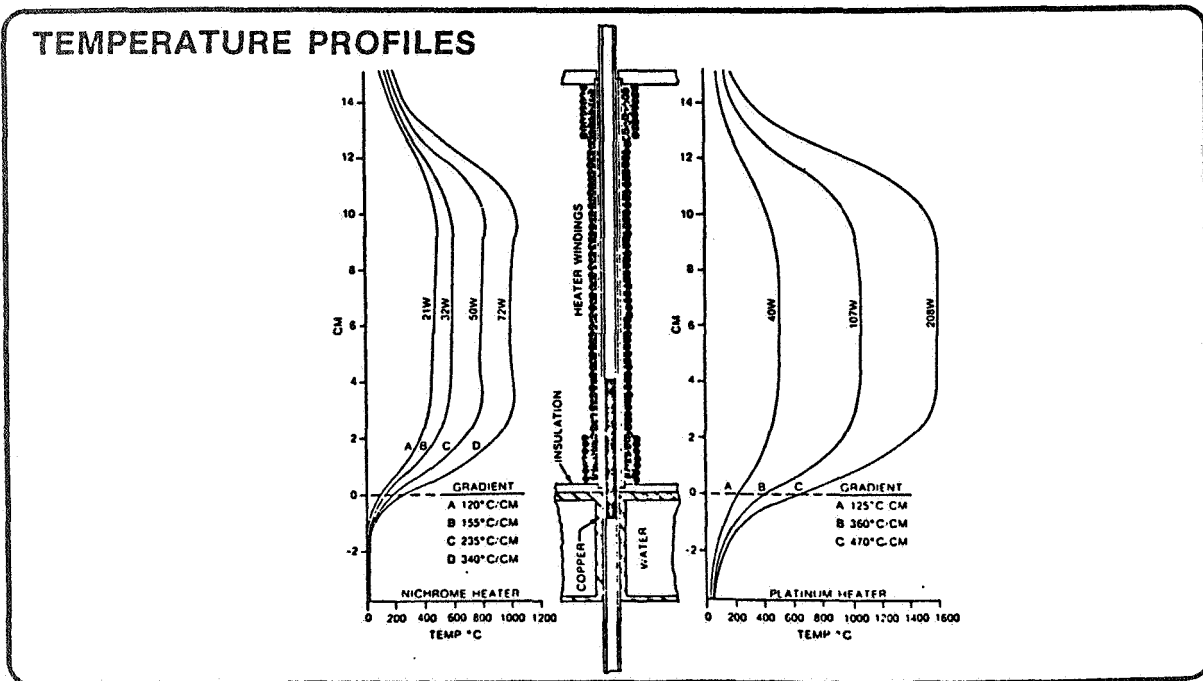
Figure 4. Idealized Expert System Controller Architecture
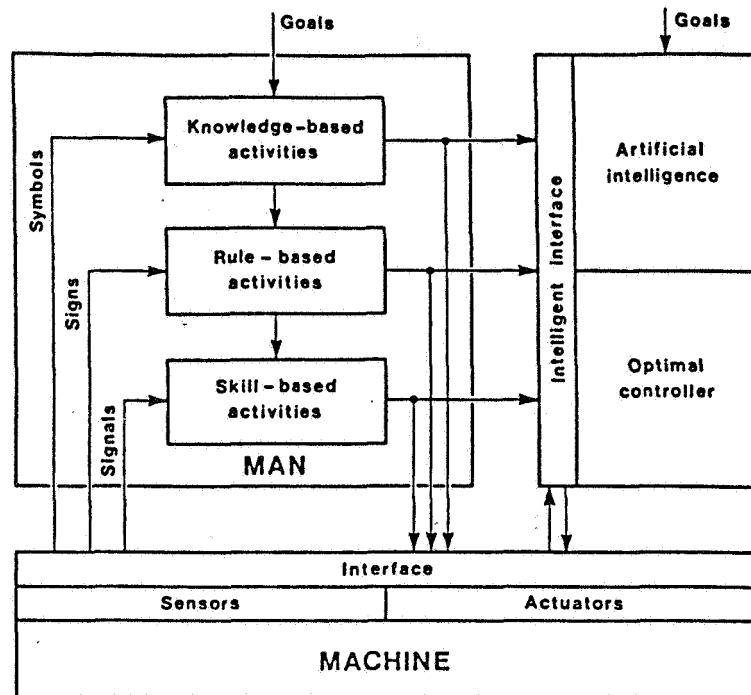
Figure 5. PICON Representation
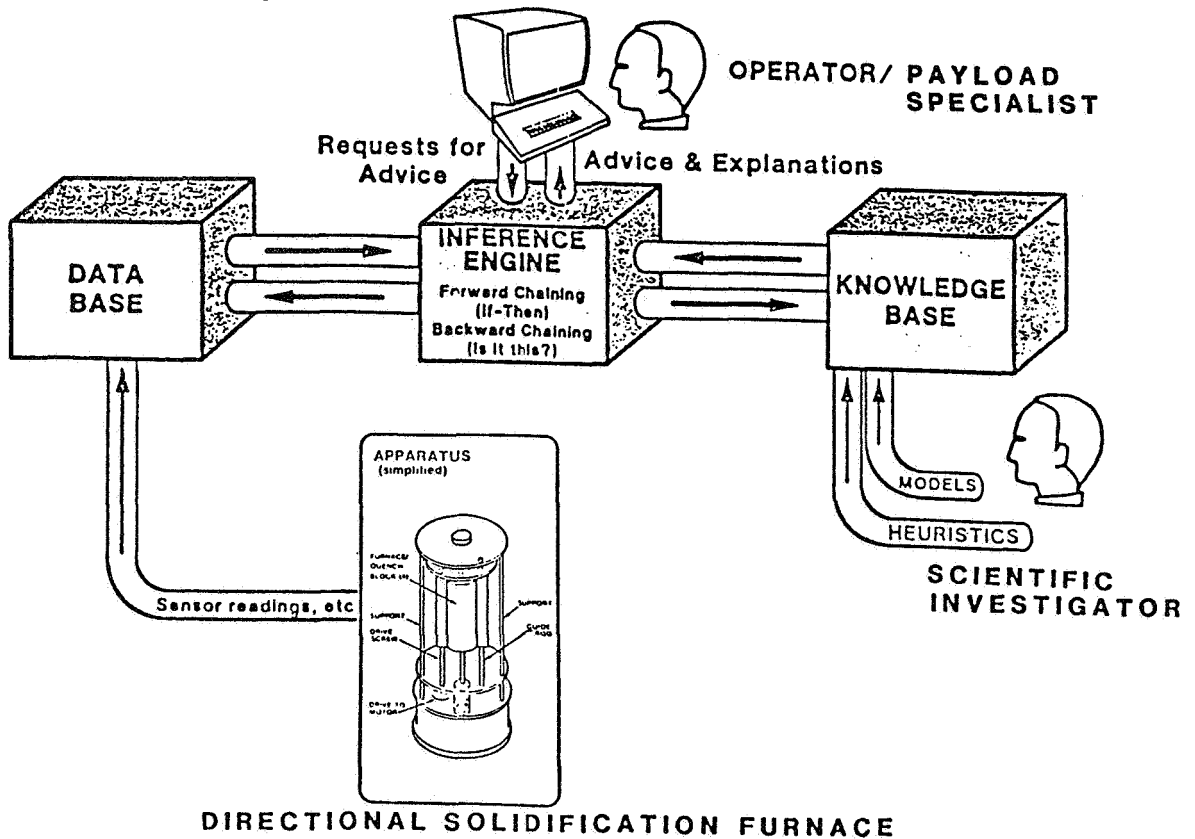
An Approach to Combining Heuristic and Qualitative Reasoning in an Expert System

*Wei-Si Jiang, **Chia Yung Han, *Lian Cheng Tsai, *William G. Wee

*Department of Electrical and Computer Engineering
**Department of Computer Science
University of Cincinnati
Cincinnati, Ohio 45221

Abstract

This paper describes an approach to combining the heuristic reasoning from "shallow" knowledge (or the rules of thumb) and the qualitative reasoning from "deep" knowledge (or the "first principles" [5]). The "shallow" knowledge is represented in production rules and under the direct control of the inference engine. The "deep" knowledge is represented in frames, which may be put in a relational DataBase Management System (DBMS). This approach takes advantages of both reasoning schemes and results in improved efficiency as well as expanded problem-solving ability.

I. Introduction

First generation expert systems, like MYCIN [9], have been built using large collections of rules of thumb, that is, rules based on empirical associations [5]. The knowledge base of these systems usually consists of pattern/decision pairs, with perhaps a simple control structure to navigate through the knowledge base [3]. Although many expert systems have proved to be successful within the domain of the expertise which is based solely on the rules of thumb, it is realized that at the boundary of the field domain these systems have a very fragile behavior [4]. The limitations imposed on expert systems by exclusive reliance on the "shallow" expert knowledge have spurred development of the systems that reason from the "deep" knowledge of the structure, function, and behavior of objects. This kind of reasoning can be called model-based reasoning, which belongs to the category of qualitative reasoning. In the rest of the paper, we will use model-based and qualitative interchangeably.

As shown in [7], the model-based reasoning can result in improved knowledge accessibility and flexibility, and expanded problem-solving ability. But due to the fact that human knowledge gained by experience is very difficult to formalize and transform precisely into a computer program, and the "first principles" [5] are not always available in some domains of which human understanding is not extensive enough to form a deeper model, heuristic approach is still necessary. In addition, heuristic approach is generally more efficient than the qualitative one. Therefore, the better architecture would include both: the "rules of thumb" consisting of pattern/decision pairs for speed and the qualitative causal reasoning for improved problem-solving ability. In this way, an expert system will be able to exploit both kinds of knowledge for problem solving.

In our approach, the "shallow" knowledge is represented in production rules and put under the direct control of the inference engine, and the "deep" knowledge is represented in frames and can be stored in the database under the control of a relational DBMS. The system will use the "shallow" knowledge more frequently than the "deep" knowledge in the ordinary situations. If a satisfactory solution, which can be measured in the certainty factor or determined by the end-user, can be achieved, the system will stop working on the problem under consideration. Otherwise, the system will exploit the "deep" knowledge for a possible better solution. We will explain all of these further and examine an example in detail.

In the rest of the paper, we first describe our system. Then we explain the control strategies for both reasoning schemes and give a simple detailed example for illustration. Finally, we have some concluding remarks.

II. Descriptions of the PAIS-I System

The PAIS-I system is an expert system shell developed by us and it is written in PROLOG and C. The shell supports both the rule-based and the frame-based knowledge representations and was designed mainly to solve the classification problems including selection, diagnosis and consultation. The control strategies directly supported by the system include the goal-directed backward-chaining, which exploits the PROLOG built-in backtracking facility, and the agenda (or dynamic priority queue) control structure. The user can also write programs within the shell to support the forward-chaining control. The domain-dependent expertise can be represented either in production rules or in frames. The frames for representing structured knowledge can be put in a relational database. We have developed a prototype of a relational database management system called VR-I (VAX/VMS Relational). The PAIS-I shell has a direct access to the VR-I DBMS and exploits the relational database as a frame-like knowledge base. This feature distinguishes PAIS-I from other systems and may be very useful for building large knowledge bases, which is necessary to solve some complicated problems. In the rest of this section, we will first explain this feature in some detail, then we will describe the knowledge representation and its implementation.

Knowledge Bases (KB) in Artificial Intelligence and databases of Database Management Systems can contribute techniques and mechanisms to each other. One of the important differences between a KB system and a DBMS is that, in general, databases in a DBMS represent specific knowledge (knowledge that can be expressed as ground Well-Formed-Formulas (Wffs) in Predicate Calculus) and a DBMS typically lacks any form of reasoning component or means of dealing with general knowledge (knowledge that has to be represented as Wffs with universal or existential quantifiers) [1,2]. One approach to dealing with these two kinds of knowledge is to build two separate knowledge bases: one for representing general knowledge and the other for representing specific knowledge [6]. For a very large knowledge-based system, the efficiency becomes crucial. Using this approach, two dedicated machines can be used for these two different kinds of knowledge: a LISP machine for the general knowledge base, and a database machine for the specific knowledge base. Thus by taking the advantage of special machines, high efficiency may be achieved. The KM-I system [6] employed this approach. Expensive hardware cost is an obvious disadvantage of this approach besides the others (for example, it is not easy to get an efficient interface of the two machines. The KM-I system used another machine, VAX-11/780, for communication between the two specialized machines).

Our approach is different from that mentioned previously. In our approach, the DBMS will remain the same as an ordinary one. All necessary control structures and procedural knowledge will be put in the expert system part. These include knowledge rules, attached procedures for the frames, DBMS query language programs, and the meta-knowledge for handling the cooperation among different kinds of knowledge.

The interface part plays an important role in our approach. Besides the interface between the system and knowledge engineer (knowledge acquisition interface) and the interface between the system and end-user (these two interfaces are necessary for every knowledge-based expert system), the system has a direct access to the DBMS (DBMS interface) and a facility to interpret the information fetched from the database. The system can generate DBMS query language programs according to the requirements of the task and is also able to manipulate the frames in the database and to combine them with the attached procedures and the other knowledge for reasoning.

As we have mentioned, the "shallow" knowledge is represented in production rules and put under the direct control of the inference engine. Here we just describe the representation of the "deep" knowledge, which is represented in frames.

A frame is a data structure for representing a stereotypical situation or a class of objects. The frame has slots for the objects that play a role in the stereotypical situation as well as relations between these slots. So it is appropriate to call it a slot-and-filler representation structure. Attached to each frame are different kinds of information, such as how to use it, what to do if something unexpected happens, default values for its slots, etc.

We have implemented a frame as a tuple in the relation. In the VR-I DBMS system, an attribute

value is a character string. In the DBMS interface part of the expert system shell, we have a text processing unit, which plays an important role in our implementation. This unit takes a character string as its input, and transforms it into literal pairs. By using this facility, a character string can be used to represent a structure consisting of attribute-value pairs. That means a nested structure is available. This facility is very useful for representing some complicated frames, which will be shown clearly later on.

The first attribute in the relation is always the frame name, which is the primary key. Thus it is very easy to invoke a frame using retrieval operation of the DBMS. To facilitate inheritance and default reasoning, each frame has an ISA slot whose value is the name of its "parent" frame, and each slot has a VALUE aspect to store the value of that slot, but VALUE aspect can be empty. Each slot also may have a DEFAULT aspect, whose value is the default value of that slot. If the VALUE aspect is not empty, the default value will not be taken into account. But in many cases, it is better to put the default value in the parent frame and to fetch the default value via the ISA link. Each slot may also have IF-NEEDED, IF-ADDED, and IF-REMOVED aspects. Their values are the names of the corresponding procedures, respectively. When a frame is invoked and a slot is inspected, the corresponding procedures will be activated automatically.

In our implementation, the frame are of the form:

Frame_name
Slot1((Aspect1 : Value1), (Aspect2 : Value2)...)
Slot2((Aspect1 : Value1), ...)
...
SlotN((Aspect1 : value1), ...)

For illustration, we will use AM radio troubleshooting as our problem domain, whose block structure is shown in Figure 1.

In our system, we have a primitive UNLESS for default reasoning. The format is of the form: IF A THEN B UNLESS C. Here we have the closed world assumption about predicate C, and exploit negation as failure in PROLOG to implement it. With the UNLESS primitive, *predicate circumscription* [8] can be easily implemented. Because of the space limitation of this paper, we will not describe the details about it.

III. Reasoning Strategies

As we have said, the system employs a goal-directed, backward-chaining mechanism to direct the reasoning process with the "shallow" knowledge represented in production rules. In many problem-solving tasks, there exist valuable "tips" for solving some problems very efficiently. For instance, in AM radio receiver troubleshooting (we will use this as a detailed example in the next section), these tips may solve the problems very quickly:

a) If there is objectionable 120-Hz hum in the sound output from the speaker, the filter capacitors need replacement.

b) Turning the power switch on and off while listening to the speaker, if no click is audible, there is probably an open circuit in the speaker, in the primary or secondary of the output transformer, or in the connections to the earphone jack.

.....

In contrast, the model-based reasoning will usually generate a longer inference chain. This is especially true in solving such problems as those mentioned above. Therefore, it is a good idea to try heuristic approach first.

To direct the reasoning with the frame knowledge base, the system employs a data-driven and "discrepancy detection" [5] control strategy. Here "data-driven" is different from a conventional data-driven forward-chaining in a production system. In our case, input data are used to select a functional unit where the system starts its reasoning process. The system heavily depends on IF-NEEDED slots to get the necessary information from the end-user. For example, if the radio is "dead", the Power

Source frame will be inspected first. If nothing is wrong in the Power Source, the Speaker and Audio Output will be inspected, and so on. If the trouble symptoms show that the audio function block, which includes Audio Output and Audio Driver, is OK, the system will start reasoning from the Detector.

Using "discrepancy detection" approach, the input and output of each function unit will be checked and compared with the permissive range. If the input is correct and the output is wrong, the problem has been isolated. After that, more specific analysis can be carried out in the same manner at a lower level through the HAS-PART link.

It is vary natural for a frame to represent a functional unit. INPUT and OUTPUT slots are used to represent the input and output of the functional unit, whose RANGE aspects specify the permissive ranges of the input and output, respectively. As mentioned above, the "discrepancy detection" reasoning can be carried out effectively with the frame representation. Natural knowledge representation facilitates the knowledge acquisition, which is the major part of the human efforts in building an expert system. Natural knowledge representation also can be understood more easily by the people other than the system developers. This is very important for the system maintenance and modification. In addition, via ISA and HAS-PART links, the hierarchical knowledge structure can be represented conveniently, and default and inheritance reasoning can be carried out easily and efficiently. In our example, the trouble can be first isolated in a high-level functional unit, then via the HAS-PART slot, the troubleshooting can go down to any details. Therefore, the frame knowledge representation is appropriate to the "discrepancy detection" approach. It should work well in the domains where functional units can be classified clearly.

IV. An Example

Using the "discrepancy detection" approach, we are dealing with the electronic troubleshooting. Here an AM radio receiver (Figure 1) troubleshooting example is given in detail.

The necessary knowledge is represented in the following frames:

Audio Output Frame:

$Audio\_Output$
$Isa((VALUE : Pw\_Amplifier))$
$Input((VALUE : x), (DEFAULT : 150), (RANGE : 100, 200), (IF\_NEEDED : ask\_user))$
$Output((VALUE : x)(DEFAULT : 50), (RANGE : 25, 75), (IF\_NEEDED : ask\_user))$
$Has\_Part((VALUE : Pw\_Transistor))$

Audio Driver Frame:

$Audio\_Driver$
$Isa((VALUE : V\_Amplifier))$
$Input((VALUE : x), (DEFAULT : 2.5), (RANGE : 2, 3), (IF\_NEEDED : ask\_user))$
$Output((VALUE : x), (DEFAULT : 150), (RANGE : 100, 200), (IF\_NEEDED : ask\_user))$
$Has\_Part((VALUE : Transistor))$

Detector Frame:

$Detector$
$Isa((VALUE : D\_Diode))$
$Input((VALUE : x), (DEFAULT : 50), (RANGE : 25, 75), (IF\_NEEDED : ask\_user))$
$Output((VALUE : x), (DEFAULT : 2.5), (RANGE : 2, 3), (IF\_NEEDED : ask\_user))$
$Forward\_bias((VALUE : x), (IF\_NEEDED : ask\_user))$

Detector Diode Frame:

D_Diode
$Isa((VALUE : Diode))$
$Forward\_Bias((VALUE : x), (DEFAULT : 0.15), (RANGE : 0.1, 0.2))$

Now the trouble symptom is output distortion, which is a relatively difficult problem in the AM radio troubleshooting. After the trouble symptom is inputted, the system will routinely ask the user to install a new battery. The trouble still exists, so the system starts to locate the trouble point. Assuming there is no such heuristic rules as IF the Detector Diode is not forward-biased THEN there will be an output distortion, the system will fail to get the solution to this problem by only using the production rules. It should be noticed that the expert system can always solve a specific problem by adding some new production rules. The point is that it is almost impossible to list all pattern/decision pairs in a complicated problem domain. This is why the model-based approach can generally result in improved problem-solving ability.

Now the system turns to exploit the frame knowledge base. Audio Output frame will be inspected first. The user will be asked to input the INPUT and OUTPUT values. Then the system compares these values to the RANGE value, and nothing is wrong. Then the system will look at the Audio Driver frame, and the situation is the same as that of Audio Output.

Then the Detector frame is inspected, the INPUT value is all right and the OUTPUT value is a little bit smaller, but still in the permissive range. The FORWARD-BIAS value is then requested, which is almost 0. But no range value is available in the Detector frame. Now the system will use the value of the ISA link to get to D-Diode frame. Under the FORWARD-BIAS slot, DEFAULT value is 0.15v and the permissive RANGE value is 0.1-0.2v. Comparing this to the user-supplied value 0, the trouble point is finally found through the model-based reasoning.

We have tried to solve ten typical problems in the AM radio troubleshooting using only the "shallow" knowledge or the "deep" knowledge, respectively. We found that the average length of the reasoning chains for the qualitative reasoning is about as three times long as that for the heuristic reasoning. But the rules-of-thumb approach failed to get the correct solutions to two out of the ten problems and the model-based approach succeeded in all the situations. When the both reasoning approaches were combined, the average length of the reasoning chains was only a bit longer than the short one, and the problem-solving ability was the same as that of the qualitative reasoning.

V. Concluding Remarks

To handle complicated problem-solving tasks efficiently and appropriately, it is better to combine both heuristic and model-based reasoning in an expert system. Frame-based knowledge representation is suitable to represent "deep" knowledge in some problem domains. The frame knowledge base can be put into a relational database management system, which is useful for building large knowledge bases.

Currently, the heuristic approach is always carried out first. If it fails to get a satisfactory solution, the system control will switch to the frame-based qualitative reasoning, and the information about the failure is not used appropriately. In general, "heuristic approach first" is appropriate, but when the system control should be switched to the other one, it needs to be studied further how the partial result obtained by production rules can be exploited effectively by the model-based approach.
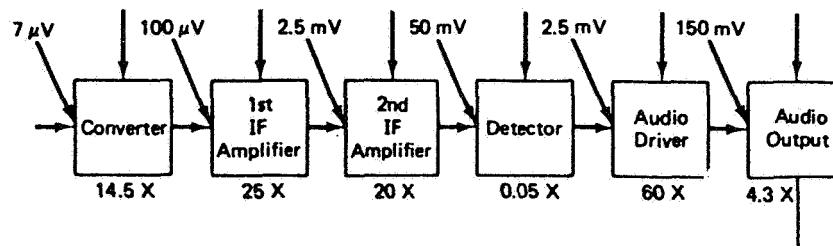
Figure 1   Block Structure of an AM Radio Receiver

References

[1] J. R. Brachman and H. J. Levesque, "What Makes a Knowledge Base Knowledgeable? A View of Database from the Knowledge Level," *Proceedings of the First International Workshop on Expert Database Systems*, Oct. 1984, pp. 30–39.

[2] M. L. Brodie, "On the Development of Data Model," in *On Conceptual Modelling: Perspectives from Artificial Intelligence, Database, and Programming Languages*, Brodie et al (eds.) New York, Springer-Verlog, 1984, pp. 19–47.

[3] B. Chandrasekaran and S. Mittal, "Deep versus Compiled Knowledge Approaches to Diagnostic Problem-Solving," in *Developments in Expert Systems*, M.J. Coombs (ed.), Academic Press, London, 1984, pp. 23–34.

[4] R. Davis, "Expert Systems: Where Are We? Where Do We Go From Here?," *AI Magazine*, Spring 1982, pp. 3–22.

[5] R. Davis, "Reasoning from the First Principles in Electronic Troubleshooting," in *Developments in Expert Systems*, M.J. Coombs (ed.), Academic Press, London, 1984. pp. 23–34.

[6] C. Kellogg, "Knowledge Management: A Practical Amalgam of Knowledge and Data Base Technology," *Proc. AAAI-82*, pp. 306–309.

[7] P. H. Koton, "Empirical and Model-Based Reasoning in Expert Systems," *Proc. IJCAI-9*, 1985, pp. 297–299.

[8] J. McCarthy, "Circumscription – A Form of Non-Monotonic Reasoning," *Artificial Intelligence 13* (1980), pp. 27–39.

[9] E. Shortliffe, *Computer-Based Medical Consultations: MYCIN*, New York, American Elsevier, 1976.

Expertise and Reasoning with Possibility:
An exploration of modal logic and expert systems

Daniel Rochowiak

Department of History and Philosophy

University of Alabama in Huntsville

Abstract

Recently systems of modal reasoning have been brought to the foreground of artificial intelligence studies. The intuitive idea of research efforts in this area is that in addition to the actual world in which sentences have certain truth values there are other worlds in which those sentences have different truth values. Such alternative worlds can be considered as possible worlds, and an agent may or may not have access to some or all of them. This approach to reasoning can be valuable in extending the expert system paradigm. Using the scheme of reasoning proposed by Toulmin, Reike and Janick and the modal system T, I will propose a scheme for expert reasoning that mitigates some of the criticisms raised by Schank and Nickerson.

Man is a rational animal. A simple saying that seems so obviously true that ordinarily we see little reason to defend it. In those cases where we do feel compelled to defend it, it is often because there has been some lapse from the normative criterion of good reasoning. The fact that there has been reasoning is not in doubt; the goodness of the reasoning is.

When the field is narrowed to those humans who are deemed experts, there is a similar shift to good reasoning. Simply knowing things about a domain and reasoning about them does not make one an expert. An expert's reasoning is good reasoning about the domain.

In considering either the distinctive features of humanity or of experts, good reasoning is a central concern. To have this central concern is to presuppose that a clear account of good reasoning can be given. Traditionally this account focuses upon valid deductive arguments. Within the deductive paradigm arguments are collections of sentences or propositions in which it is claimed that one, the conclusion, follows from the others, the premises, and that an argument is valid if whenever all of the premises are true, then the conclusion is true. Deductive arguments are marked by being

demonstrative. For deductive arguments additional information, additional premises, can never render a valid argument invalid. Within the deductive paradigm all arguments are either valid or invalid. Any idea of differing strengths of deductive arguments must be a function of the strength (confidence, probability, truth content) of the premises taken by themselves, and in no case can the strength of the conclusion exceed that of the weakest premise.

The metaphorical structure of the deductive paradigm is captured by the notion of a chain. The logic of inferences forges the form of the links, and the sentences and their strengths are the matter for the links. Groups of such links form the chain of reasoning, and if any link is broken, the chain fails to support its weight, the ultimate conclusion.

The deductive paradigm has, of course, been developed with great care, clarity and subtlety. However, even though there is no reason to think that it is not *a* correct paradigm of good reasoning, there are nagging doubts that it is the *only* form of good reasoning. In particular, it seems reasonable to believe that in many instances good reasoning has a different "texture" and that good reasoning can at times lead to conclusions that are stronger than the weakest premise.

The texture of reasoning can be specified in terms of the functional parts of the reasoning. In a deductive argument, the texture is specified in terms of the premises, conclusion and the method, syntactic or semantic, for moving from the premises to the conclusion. (Diagram 1). The texture of arguments in the deductive paradigm is flat.

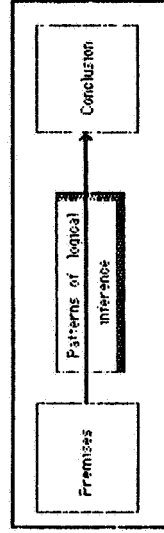Premises — Patterns of logical inference — Conclusion

Diagram 1

Some instances of good reasoning seem to include more functional parts than those allowed by the deductive paradigm. Consider a simple example of scientific reasoning. Potatoes belong to the genus *Solanum* and many members of that genus, for example nightshade, have

poisonous foliage and berries. Now biochemical laboratory studies of closely related species and genera show that closely related plants contain closely related biochemical compounds. Thus, it would seem very possible that potatoes have poisonous foliage unless the biochemistry of the potato more closely resembles that of non-poisonous *Solanum*, for example the eggplant. Such reasoning is not easily captured in the flat texture of the deductive paradigm. Every claim except the claim that potatoes have poisonous foliage would be treated as a premise, and the qualifications — many, closely, very possible — would be treated as determiners of the degree of strength of the claims in the argument. It is, of course, possible to force the reasoning into the deductive paradigm, but there might be a better alternative.

Toulmin, Rieke and Janick have proposed a scheme for representing good reasoning that allows for a greater texture (Diagram 2).[1] Within the TRJ scheme the functional parts are the claims, grounds, warrants, backings, modalities, and the rebuttals.

claim, and the backing indicates the support for the warrant. Modalities specify the degree of support for the claim, and rebuttals specify the conditions under which the reasoning might fail. With these elements the reasoning about the potato could be captured in the following way:

G = potatoes belong to the genus *Solanum* and many members of that genus, for example nightshade, have poisonous foliage and berries,
B = biochemical laboratory studies of closely related species and genera show that,
W = closely related plants contain closely related biochemical compounds,
M = it would seem very possible that,
C = potatoes have poisonous foliage,
R = the biochemistry of the potato more closely resembles that of non-poisonous *Solanum*, for example the eggplant.

Several features of the TRJ scheme are significant. First, the backing for a warrant may change without substantially altering the reasoning.[2] In the example the laboratory backing could be replaced with a theoretical backing. Second, the addition or deletion of items in the grounds need not substantially alter the reasoning. Discovering that a species thought to be poisonous is not could leave the reasoning in tact. Third, the rebuttals provide a guard against specified circumstances and contain the seeds for new reasoning.

A further feature of the TRJ scheme is that the devices of the deductive paradigm remain available. In particular the devices of modal logic are relevant. Consider the features that give the TRJ scheme its texture, the backing and rebuttals. In the case of B it seems reasonable to think that that it is to be found in a base of information other than that containing G. Thus, G and B are distinct. This other base of information can be thought of as a world to which the rest of the reasoning has access. In the case of R a similar approach seems reasonable. R contains reasonings that would be good reasonings in some other possible world. Thus, the reasoner that reasons in accord with the TRJ scheme would have access to not only the actual world containing G, W and C, but also other possible worlds containing B and R. Thus, it would seem appropriate to enlist the support of modal logic in further developing the TRJ scheme. In the remainder of this presentation I will focus upon the modal system T.[3]

A syntactic characterization of T can be given by adding to the

"Given grounds, G, we may appeal to warrant, W (which rests on backing, B), to justify the claim that C — or the presumption (M) that C — in the absence of specific rebuttal (R)."

Grounds G → Warrant W → Modality M → Claim C

Backing B

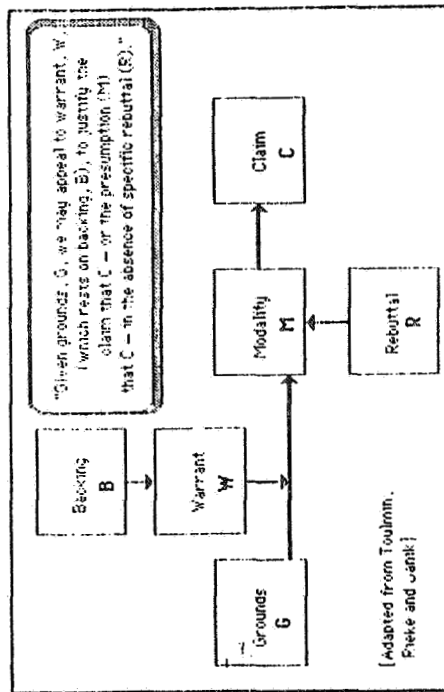Rebuttal R

[Adapted from Toulmin, Rieke and Janick]

Diagram 2

A claim is analogous to a conclusion in the deductive model; it is the sentence to be established. The grounds are analogous to the premises of the deductive paradigm, but they are restricted to the data being considered. Warrants establish the relevance of the grounds to the

In this illustration each world has access to itself and one other world, but the access relation is not symmetric. The starting sentences are in the boxes of each world circle. Normal propositional rules apply within each world. Note that ◇◇p is a thesis but □p is not. W_1 is the designated or actual world.

Diagram 3

A bit more formally, a T model is an ordered triple $\langle W, R, V \rangle$ where W is a set of worlds, R is a dyadic reflexive relation defined on the members of W and V is a value assignment such that

1) For any atom, p_i, and any $w_i \in W$, either $V(p_i, w_i) = T$ or $V(p_i, w_i) = F$,

2) For any wff α and any $w_i \in W$, $V(\sim\alpha_i, w_i) = T$ if $V(\alpha_i, w_i) = F$, otherwise $V(\sim\alpha_i, w_i) = F$,

3) For any wffs α and β and any $w_i \in W$, $V((\alpha \vee \beta)_i, w_i) = T$ if either $V(\alpha_i, w_i) = T$ or $V(\beta_i, w_i) = T$, otherwise $V((\alpha \vee \beta)_i, w_i) = F$,

4) For any wff α and any $w_i \in W$, $V(\Box\alpha, w_i) = T$, if for every $w_j \in W$ such that $w_i R w_j$ $V(\alpha_j, w_j) = T$, otherwise $V(\Box\alpha, w_i) = F$,

5) For any wff α and any $w_i \in W$, $V(\Diamond\alpha, w_i) = T$, if for any $w_j \in W$ such that $w_i R w_j$ $V(\alpha_j, w_j) = T$, otherwise $V(\Diamond\alpha, w_i) = F$.

The wff α is T-valid if and only if for every T-model $\langle W, R, V \rangle$ and

---

formation rules of the propositional calculus the rule that:

If p is a well formed formula, then □p and ◇p are wffs.

The axioms of the propositional calculus should also be extended to include two new axioms:
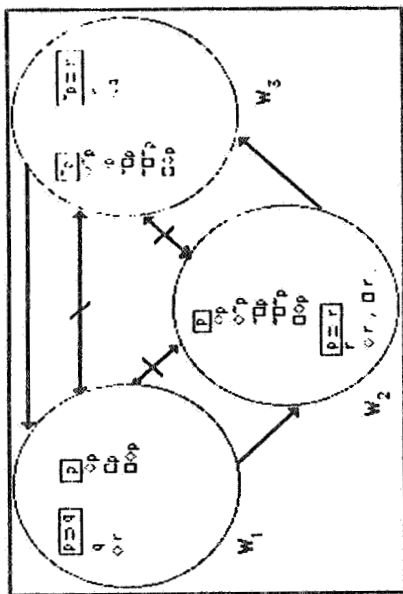
T1 □p ⊃ p
T2 □(p ⊃ q) ⊃ (□p ⊃ □q).

To the rules of the propositional calculus should be added the rule of necessitation:

If α is a thesis, then □α is a thesis.

The system T can also be developed as a natural deduction system.[4]

The semantics of T can best be understood in terms of a game. (Diagram 3 may be helpful.) The game contains a caller, any number of players and sheets of paper for each player. On each sheet of paper are the letters that represent contingent atomic propositions. The caller can call out any wff she wants, provided that the call has been suitably prepared. Suitable preparation for a call requires that every subformula of the called wff has been previously called. The players can be arranged in any way one likes. This is important for the "seeing" relation. The players can be arranged so that every player can see every other player, or so that one player can see every other player, or so that if player A can see player B and B can see C, then A can see C, or any other pattern.

The game is played according to a set of rules that determine whether a player is to keep her hand down or put it up. If an atom is called and the atom is on her sheet, the player raises her hand. If a negated atom is called, the player raises her hand if the atom is not on her sheet. If a disjunction is called, the player raises her hand if she has raised her hand for either subformula. If a material implication is called, she raises her hand if she did not raise her hand for the antecedent subformula or if she raised her hand for the consequent subformula. If the call is for the possibility of some proposition (◇p), then she raises her hand if she saw any player's hand raised for the subformula. If the call is for the necessity of some proposition (□p), then she raises her hand if every player that she could see raised her hand for the subformula. A call is successful in a setting if each player raises her hand for the call. A call is T-successful if in all settings each player would raise her hand

113

every $w_i \in W$, $V(a_i, w_i) = T$.

The system T helps to account for several of the distinctive features of the TRJ scheme.

First, one can consider a world, W_2, such that it contains the sentences that are the backings, B, for the warrants, W. Within W_2 there would be, for example, sentences that concern laboratory studies, the relations between plants and the relations between biochemical compounds. W_2 would be accessible to the world, W_1, where the actual reasoning takes place. However, W_2 may not be the only world to which W_1 has access. W_1 may also have access to W_3 which contains theoretical claims about species and biochemical compounds. Thus, both W_2 and W_3 might contain the appropriate Ws and Bs to be used in the reasoning in W_1. This allows for a clear sense in which the reasoning in W_1 that concerns the Gs and Cs found there may remain substantially the same even if B changes. If for any reason the B found in W_2 is found to be no longer acceptable, then it is still possible to accept the warrant W because W can be backed by a B found in W_3. This ability to have multiple backings for a warrant also explains to some degree why some arguments are thought to support their conclusion to a higher degree than the least supported premise.

The possibility of multiple backings means that even if one of the backings is removed another is available. Thus, even if the backings are deemed weak the availability of two or more backings increases the strength of the reasoning. The metaphorical structure for this sort of reasoning is not captured by the image of the chain. Rather it seems that the metaphorical structure that captures this sort of reasoning is a cable.

Second, the rebuttals seem to be a rather obvious instance of an appeal to a possible world. A rebuttal contains the core of reasoning that would be correct in some other world, W_4, such that the claim of that reasoning would be contrary to the claim being put forward in the actual world, W_1.

Third, the interpretations of "□" and "◊" can be specified in terms of knowledge. Such an interpretation would specify that an agent, s, knows the sentence, p, just in case p is true in every world to which s has access 5 (This sort of interpretation is often represented by $K_s p$.) However, this is not the only interpretation in terms of knowledge

Using the perspective of the caller, one might interpret the whole system as an agent. Within this interpretation a sentence, p, would be known just in case □p is true in the designated world, say W_1, and p would be known to be possible just in case ◊p is true in W_1. From the point of view of the caller (system) the designated world need not remain fixed. Thus, from the caller's point of view it could come to be known that W_1 was not the actual world and that some other world, W_2, was the actual world. This idea of changing points of view as to what world should be the designated or actual world is clearly connected to the rebuttals, R, of the TRJ scheme of good reasoning.

The advantages of the combination of the TRJ scheme with the resources of modal logic mitigate some of the criticisms of expert systems raised by Schank and Nickerson.

An expert system is a rule based, inference making system that contains a data base and an inference engine. For convenience the data base is divided into two parts a working memory, WM, and a production memory, PM (Diagram 4). The WM contains the sentences entered by the user and the sentences inferred from the rules in the PM by the inference engine. The rules in the PM have a uniform form:

IF <WM pattern>

THEN <WM change>.

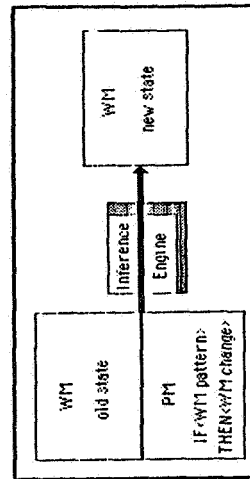Thus, an ordinary expert system has a flat texture.

Diagram 4

Such flat expert systems can seem to be rather limited. As Schank puts it:

... the reason we value human experts is that they can handle a situation in which the events take a new twist, or that doesn't fit the text book rules. Expert systems, while potentially useful, are not a theoretical advance in our goal of creating an intelligent machine. Real intelligence demands the ability to learn, to reason from experience, to *shoot from the hip*, to use general knowledge, to make inferences from gut-level intuition. Expert systems can do none of these. They don't improve as a result of experience. They just move on to the next if/then rule.[6]

Nickerson offers a distinction that allows one to put the problem in a more general way.

An important distinction to be maintained in this process is that between general intelligence and expertise. As these terms are generally used, they connote related but quite different things. *Intelligence* has to do with the ability to learn; *expertise* connotes knowledge that has already been acquired. *Intelligence* refers to general intellectual competence; *expertise* connotes in depth understanding of a specific and, typically, narrow domain. Intelligence rests in a set of cognitive abilities for abstraction, classification, generalization, drawing inferences and analogies, and so forth; expertise is the ability to access and apply information about a given topic on demand.[7]

An extended expert system using the TRJ scheme and the resources of modal logic could escape some of these criticisms. Some of the criticisms would, of course, still stand; nothing has been said of learning, intuition or general competence. However, such an extended expert system could make inferences, could respond to some new twists (through multiple backings or rebuttals), could use information from more than one domain (world), and could to some degree shoot from the hip (by recognizing possibilities and changing designated worlds).

To return to the opening comments, the central issue in an examination or implementation of expertise is good reasoning. Good reasoning need not be limited to the flat texture of the deductive paradigm. The TRJ scheme and the resources of modal logic can be used to generate a model of good reasoning that has a greater texture. However, such a model of good reasoning tells us very little about how humans, whether experts or not, actually do reason, and it should be remembered that, at least at times, reasoning that does not fit into any model of good reasoning can produce great results. This ought not to argue against further research on good reasoning or expert systems; it should only serve as a reminder that there *is* a great gap between what is and what *ought* to be.

Notes

1) S. Toulmin, R. Rieke and A. Janik, *An Introduction to Reasoning*, New York: Macmillan, 1979. The way in which the TRJ scheme is developed in this essay is not a way that was intended by the authors.

2) The use of multiple lines of reasoning in science has been referred to as robustness and triangulation. See the following: W. Wimsatt, "Robustness, reliability, and observation" in M. Brewer and B. Collins (eds.), *Scientific Inquiry and the Social Sciences*, San Francisco: Jossey-Bass, 1981, pp.124-163. Susan Leigh Star, "Triangulating Clinical and Basic Research: British Localizationists, 1870-1906," *History of Science* 24(1986): 29-48. D. Rochowiak, "Darwin's psychology: triangulating on habit," under review.

3) The development of T is in accord with G. E. Hughes and M. J. Cresswell, *An Introduction to Modal Logic*, London: Methuen, 1968. See also, R. Turner, *Logics for Artificial Intelligence*, West Sussex, England: Ellis Horwood, 1984.

4) See, K. Konyndyk, *Introductory Modal Logic*, Notre Dame: University of Notre Dame Press, 1986.

5) The development of epistemic logics has received a good deal of attention from the AI community. See, J. Y. Halpern (ed.), *Theoretical Aspects of Reasoning about Knowledge*, Los Altos: Morgan Kaufmann, 1986.

6) R. C. Schank with P. Childress, *The Cognitive Computer: On language, learning and artificial intelligence*, Reading, Mass.: Addisno Wesley, 1984, p. 34.

7) R. Nickerson, *Using Computers: Human factors in information systems*, Cambridge, Mass.: Bradford Book / MIT Press, 1986, p. 302.

"ANALYST-CENTERED" MODELS FOR SYSTEMS DESIGN, ANALYSIS, AND DEVELOPMENT

A. P. Bukley
Dr. Richard H. Pritchard
Steven M. Burke
P. A. Kiss

The BDM Corporation
2227 Drake Avenue
Huntsville, AL 35805

Abstract

Much has been written about the possible use of Expert Systems (ES) technology for strategic defense system applications, particularly for battle management algorithms and mission planning. Other potential applications of ES include space missions planning as well as the management of extremely large software simulations. It is proposed that ES (or more accurately, Knowledge Based System (KBS)) technology can be used in situations for which no human "expert" exists, namely to create design and analysis environments that allow an analyst to rapidly pose many different possible problem resolutions in "game-like" fashion and to then work through the solution space in search of the optimal solution. Portions of such an environment exist for expensive AI hardware/software combinations such as the Xerox LOOPS and Intellicorp KEE systems. This paper will discuss efforts to build an "analyst-centered" model (ACM) using an ES programming environment, ExperOPS5 for the Apple MacIntosh, for a simple missile system trade-off study. By "analyst-centered", it is meant that the focus of learning is for the benefit of the analyst, not the model. The model's environment allows the analyst to pose a variety of "what if?" questions without resorting to painstaking and time-consuming programming changes. Although not an ES per se, the ACM would allow for a design and analysis environment that is significantly superior to that afforded by current techniques.

A. INTRODUCTION

The January 1986 issue of Technology Review magazine had the following provocative words on the cover: "After 25 Years Artificial Intelligence has failed to live up to its promise and there is no evidence that it ever will." The cover article, authored by Hubert and Stuart Dreyfus, enumerates the many failures to build AI programs that truly "think like people". These authors are particularly hard on so-called "expert systems", pointing out that human experts can always "beat" the machine at checkers, and that they rely much more on intuition than on a set of "rules" that may be articulated on a computer. While it is contended that the jury is still out on ES technology and its ultimate utility, the authors agree that the value of human expertise and its interface with machine systems has occasionally been lost in the hype that continues to surround the marketing of AI technology. It is perhaps instructive for those with practical problems such as the design and development of complicated

PRECEDING PAGE BLANK NOT FILMED

military hardware and software or the operation of a shuttle or space station to present to the AI community some near-term examples of how ES and other AI technology can be put to use now.

B. THE BIRTH OF THE ANALYST-CENTERED MODEL CONCEPT

The Xerox Corporation has developed a knowledge programming system called LOOPS (Lisp Object-Oriented Programming System). In a course designed by Xerox to teach LOOPS, a game called Truckin' is used to teach the concepts to students. The course is described in Reference 1. Truckin' is a variation on Monopoly in that the overall goal of the players is to end the game with the most money; however, there is one major difference: instead of playing the game themselves, the students create "intelligent" players who follow a set of rule-based behaviors in playing against each other and the game clock. Early LOOPS courses would end with a mass Truckin' competition using players developed by all students, often with revealing results about odd player behaviors and weaknesses.

The Truckin' game has since been dropped from the course because Xerox Special Information Systems found that students devoted too much time to the competition and not enough to learning LOOPS. Nevertheless, the game provided an excellent paradigm of an "intelligent" military simulation in that all the key elements were there: conflict, competition for resources, time factors, strategies, and object-like players. It was hypothesized that a LOOPS environment could perhaps be used to model real-world military systems that were not sufficiently defined to be amenable to conventional modeling techniques. Such a model would incorporate the elements of an ES without being a true "expert"; that is, it would be a Knowledge Based System that complemented the human analyst by serving as a rapid prototyping environment for her/him. This basic idea became the seed from which the "analyst-centered model" grew.

One of the enduring challenges of military systems analysis is attempting to build simulations of weapon systems that are incompletely (and more often than not, vaguely) specified. Likewise, a major challenge would be to devise the optimal way to get people to Mars, or any other planet, if one were so inclined; especially because a space mission of that magnitude has never before been attempted. In both of these examples there is a sufficiently large number of unknowns through which the designers and analysts must sort in order to produce sensible and practical solutions to merit the employment of some kind of "expert". The use of "exploratory programming environments" such as AI environments for handling situations like the examples sited above has been ably discussed by Beau Shiel in Reference 2. Unfortunately, using these environments has its drawbacks: the Lisp machines are expensive ($50-100K each). They are special-purpose, single-user computers. Some of the software required to implement such environments costs almost as much as the machines themselves. To become facile in any of the dialects of Lisp and/or LOOPS, Flavors, or KEE takes a good programmer or analyst the better part of a year to do. Finally, project officers are understandably skeptical that the investment of over $400K for labor, hardware, and software plus a year's time is going to yield a tool of appreciably more value than conventional programming methods and are reluctant to fund such projects.

118

If there were perhaps a less expensive and quicker way to produce such programming environments, their use might become more attractive. An inexpensive, rapid prototyping capability for ACMs using a personal computer would represent a great leap forward, enabling the presentation of results quickly, and thus illustrating the value of the expenditure of additional funds for a more detailed version of the ACM. The recent commercial availability of Lisp and OPS5, called ExperLisp and ExperOPS5, respectively, for the MacIntosh and IBM PC seemed to hold the promise of being able to do exactly that. The MacIntosh version was chosen in the hopes of quickly producing a version of "MacLOOPS". However, it was quickly discovered that neither the MacIntosh hardware nor the ExperLisp and ExperOPS5 software were even remote competitors for a Xerox 1186 Dandelion and LOOPS. The Smalltalk-like cartoon graphics and the pull-down menus using the mouse were about where the resemblance ended. However, the system developed can be used to create a very rough prototype of an ACM and to demonstrate the utility of the concept. In the following section of this paper, the simple missile model built on the MacIntosh as well as potential applications for an ACM are discussed.

C. APPLICATIONS OF AN ANALYST-CENTERED MODEL

The ACM rapid prototyping utility was tested via a very simple interceptor simulation. The goal of the effort was to develop an inexpensive KBS prototype to aid in missile system design trade-off analyses as well as to prove concept feasibility. Both goals were accomplished. The KBS provided a learning environment for the human analyst by acting as a "smart" interface between the analyst, a database, and a system simulator, as shown in Figure 1. This goal differs from the traditional use of an ES, which is to capture the ruleset of a human expert in a given area. This means that the KBS is designed to perform within a specified and very limited domain at the approximate level of the human expert. By contrast, the KBS designed performed within a broader domain by sacrificing its level of expertise. Using the ACM, an analyst can test many designs or hypotheses rapidly. The manipulation and experimentation made possible by the ACM environment make useful synthesis (the ability to see the whole problem) and system insight much more likely on the part of the human analyst. This can lead to better-directed choices for future research.

For example, suppose a missile analyst wanted to use a certain high transmissivity material as a sensor lens in a particular interceptor so that its effect on the system's overall performance could be assessed. The KBS would alert the analyst, for example, that a lens made from this material is structurally unsound at the usual aperture size, or that the material is strongly water absorbing and cannot be used in the atmosphere without a protective coating. In spite of this feedback information, the analyst may choose to proceed with exercising the simulation using the enhanced transmissivity value, noting that research is required to improve the structural strength or to provide a coating.

A critical difference between an ES and an ACM is now seen: namely, the ACM does not alter the database or the simulation, but alerts the analyst to a potential problem through the KBS interface. The analyst,

119

not the KBS, decided what the next step would be, and hopefully, learned something in doing so. The ACM provided detailed, pertinent information in an area not necessarily familiar to the analyst. In a sense, the function of the KBS is to serve as a "smart front end" to the database and the simulation, allowing the analyst to serve as the ultimate rulemaker. Furthermore, were this ACM to be implemented on a general-purpose computer capable of using Lisp or OPS5 (a VAX or a MicroVax, for example) the KBS (written in Lisp) could call up the missile simulation (written in FORTRAN) and the database (written in DBMS language), thus acting as an intelligent configuration management system.

The simple simulation implemented in the prototype ACM mentioned earlier consists of a planar two degree-of-freedom point mass simulation of a missile intercepting a ballistic target. Features of the simulation include a model of gravitational effects, Newtonian equations of motion, a ground launched interceptor, effects of aerodynamic drag, proportional navigation guidance, and two flight phases (boost and terminal homing). The analyst has the capability to choose various thrust levels, different initial conditions, type of guidance, and the commanded acceleration limit. A very basic set of rules is used to execute the simulation and alert the analyst to the results. A simple graphics display of the simulation was also developed. Although quite elementary, the prototype ACM demonstrated the potential capabilities of such a system.

Another area in which an ACM may find potential application is that of extended space mission planning. For example, in order to carry out a successful mission to Mars, many factors must be taken into consideration. There are many transportation options to consider. The method of propulsion must be decided. How or whether to create an artificial gravitational field for the astronauts in transit is another question. The types of mission module configurations that are optimal for such a mission, as well as the configuration of the surface modules must be resolved. The entire issue of life support is a critical consideration. Obviously, with so many factors to consider, there is indeed no one expert on missions to Mars. This type of problem lends itself quite well to analysis in an ACM environment wherein a database with all the options for all the phases of the mission are contained is linked with a mission (or mission phase) flight simulation. A KBS can be used to interface these two modules with the analyst and present her/him with all the options and the implications of the choices made.

Another example of a potential application of an ACM is in the area of simulation management. As is known by anyone who has ever developed or worked with a complex multi-routined simulation, things can quickly get out of hand once the simulation reaches "critical mass". The proposed National Test Bed (NTB) will be host to many hundreds of simulations. An ACM-type KBS is being designed by BDM to help select the routines and simulations required to run an analysis of a particular set of experiments in the NTB.

D. SUMMARY

The rapid prototyping value of the ACM approach is clear. Several lessons were learned with the MacIntosh experiment. First, the ACM

120

implementation on the Mac totally overwhelmed the machine. A larger machine with enhanced capability is required to produce a usable ACM. It was also learned that a KBS may be coupled to a simulation in a meaningful way to obtain useful results. However, it was learned that rulesets may not be an absolute requirement, depending on the application, and that the object-oriented environment may simply be a better one for building executable designs. The path for moving to larger, more general-purpose machines is relatively clear. Programming languages other than Lisp and OPS5 may be used where appropriate in implementations of ACM's. Ultimately, an analyst will be able to ascertain the system impact of a small change in a component material, computational capability, or partial failure.
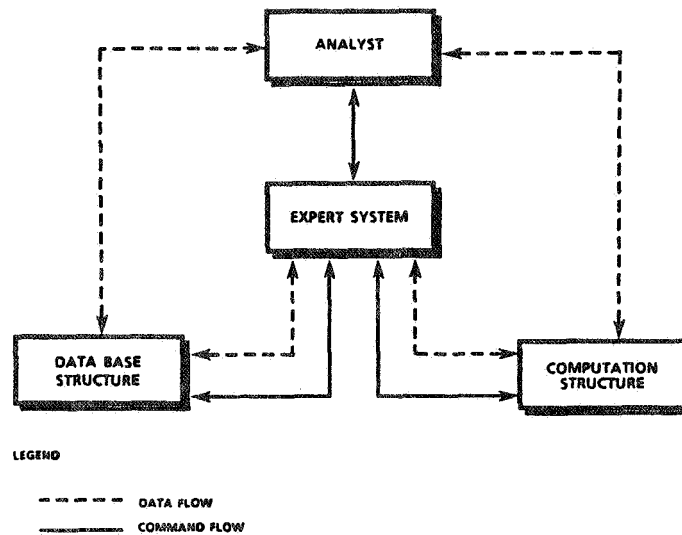
Figure 1. A Generic ACM Structure

BIBLIOGRAPHY

Stefik, M., D. G. Bobrow, S. Mittal, and L. Conway, "Knowledge Programming in LOOPS: Report on an Experimental Course", The AI Magazine, Fall 1983, pp 3-13.

Sheil, B., "Power Tools for Programmers", Datamation, February 1983, pp 131-144.

AVIONIC EXPERT SYSTEMS *

Forouzan Golshani
Department of Computer Science
Arizona State University
Tempe, Arizona 85287
Telephone: (602) 965-2855

ABSTRACT

At the heart of any intelligent flight control system, there is a knowledge-based expert system. The efficiency of these knowledge bases is one of the major factors in the success of aviation and space control systems. In future, the speed and the capabilities of the expert system and their underlying database(s) will be the limiting factors in our ability to build more accurate real-time space controllers.

We propose here a methodology for design and construction of such expert systems. We note that the existing expert systems are inefficient (in fact, slow to an extreme) in dealing with non-trivial real-world situations that involve a vast collection of data. On the other hand, current databases, which are fast in handling large amounts of data, cannot carry out intelligent tasks normally expected from an expert system.

Our system will provide the power of deduction (reasoning on situations) along with the efficient mechanisms for management of large databases. In our system, both straightforward evaluation procedures (used for data handling purposes) and sophisticated inference mechanisms (used for deduction) will coexist. Our design methodology is based on mathematics and logic, which ensures the correctness of the final product.

PRECEDING PAGE BLANK NOT FILMED

AVIONIC EXPERT SYSTEMS

1. INTRODUCTION

Expert systems for avionics purposes are expected to perform under unusually strict circumstances and as such they have to confront a number of challenges not expected from ordinary expert systems. We briefly look at some of these problems first, and examine the specialized techniques that are required to deal with them.

At the heart of any intelligent flight control system, there is a knowledge-based expert system. The efficiency of these knowledge bases is one of the major factors in the success of aviation and space control systems. In future, the speed and the capabilities of expert systems and their underlying database(s) will be the limiting factors in our ability to build more accurate real-time space controllers.

Some main characteristics of aviational systems are:

o   accuracy, reliability and the power of rapid recovery:
Ordinary expert systems generally operate in an off-line advisory mode. In contrast, avionics expert systems have to meet the constraint of keeping pace with the (rapid) events in their environment. Due to their "on-line" nature, they should also meet the high standards of reliability, and preferably have a great deal of potential for recovery after inevitable failures.

o   Ability to cope with time constraints (i.e. efficiency):
Since these systems are *real-time*, the acquisition of flight status information should ideally be done simultaneously with the operations of the inference engine for making decisions in order to accommodate these real-time constraints. While we do not address here the problem of parallelism in expert systems, we will emphasize the efficiency issues.

o   Intensity of data (and knowledge):
There is a great of information that the system collects in dealing with constantly changing situations. Depending hw

sophisticated the system is intended to be, the size of the underlying databases varies.

We should bear in mind that the problem of communication between these databases (when there are more than one databases) is not a trivial one. Sheer volume of information and the issue of redundancy force many constarints.

While, at this moment, a total automatic control is neither technically feasible nor socially acceptable [ChNa-85], statistics indicate that automation does help in the safty of aviation. A report by International Civil Avaition Organization [ICAO-84] shows that the percentage of aviation accidents that can be attributed to human error has been steadily on the increase whereas the number of accidents caused by the machine has declined. Figure 1, taken from [ChNa-85], represents these facts. This is one of the motivations for developing more advanced control systems that can help the pilot and others in reducing the risk of errors.

The current software technology employed for automated flight control systems (e.g. those made by Sperry) are primarily fast database systems. We note that conventional database systems are mainly concerned with storage and retrieval of data, and the efficiency of these activities. Generally, all of the information must be explicitly stored and there is no mechanism for deriving new facts from the existing data. In expert systems, the emphasis is largely on the deduction of new facts as these systems should be not confined to the data stored. In addition to giving precise and complete answers to questions, expert systems for aviation purposes should be able to cope with certain types of tasks such as:

1. The system should be able to make predictions for certain possible situations. It sould answer queries of the type: "What would be the consequence if the event X happens?"

2. Given that there is concern for the occurrence of a certain event, the system must indicate the possible causes of that event. This amounts to answering queries such as: "Why would the event X happen?"

3. At an advisory capacity, it should suggest measures for preventing certain unwanted situations. Here we deal with questions of the type: "What can prevent the occurrence of the event X?"

4. On many cases, more than one strategy may be helpful. The system should be capable of indicating which alternatives can be candidates for solutions in addition to the definite answers obtained from the database?

While current AI techniques (in principle) can provide the machinery to cope with the above, we note that, at least with the present technology, they are not efficient enough for answering ordinary database queries in systems involving large amounts of data. In the case of simple database queries, we have straightforward computation where we know which actions must take place. Therefore, there is no need for random searches and tentative reasoning which require a great deal of computation time.

We propose here a methodology for design and construction of such expert systems. We note that the existing expert systems are inefficient (in fact, slow to an extreme) in dealing with non-trivial real-world situations that involve a vast collection of data. On the other hand, current databases, which are fast in handling large amounts of data, cannot carry out intelligent tasks normally expected from an expert system.

Our system provides the power of deduction (reasoning on situations) along with the efficient mechanisms for management of large databases. In our system, both straightforward evaluation procedures (used for data handling purposes) and sophisticated inference mechanisms (used for deduction) will coexist. Our design methodology is based on mathematics and logic, which ensures the correctness of the final product.

126

2. METHODOLOGY

As outlined above, we emphasize correctness and accuracy in addition to efficiency of operation. For this reason we have first developed a mathematical framework for the design and implementation of our system which will ensure correctness. Simply put, instead of having verification and debugging at the end, we have *correctness by construction*.

Expert systems are seen as dynamic objects, where updates change the state of the database, and the states are used for reasoning and answering queries. Thus, we deal with them at two levels: dynamic and static. For the dynamic part we have developed a modal logic system. Modal logic originally began as a vehicle to deal with necessity and possibility. There is a collection of possible worlds considered with an accessibility relation to determine which world is accessible from which. A proposition is called "necessary" if it can be satisfied in all admissible worlds, and it is called "possible" if it is satisfied in some. The domain of interpretation (or the universe) of the modal system is the set of database instances, and the accessibility relation is determined by the update functions [Gol-86].

The instances of the expert system (i.e. the states of the dynamic system) are seen as a collection of sets together with a collection of functions mapping these sets to each other. Those familiar with mathematical ideas will recognize that we would be dealing with a many-sorted algebra. We use the signature of the algebra as the specification for the type checker and the syntax checker of the database language. Having a type-checker will enable us to detect "type errors" statically prior to the evaluation of queries. Computation power is provided by including a sufficiently rich collection of operations (such as arithmetic, set theoretic, etc.) which would be fixed across all applications. Programs (queries) are then simply expressions which are built up out of the symbols in the signature together with the operation symbols and which comply with the precise formation rules given by the query language. Integrity constraints are expressed as boolean valued expressions that must hold in all instances (or

algebras). Notice that we propose a sound mathematical model for construction of expert systems (i.e. a modal system of algebras).

The power of deduction is provided by allowing inference rules which are activated by programs (queries) or by users. The inference rules are also expressions of type boolean. Although the deduction rules can be invoked by the language processor, it is possible to define operators which explicitly trigger the inferencing mechanism. [Gol-84]

It must be emphasized that, despite a mathematical approach, the resulting language is extremely simple and easy-to-use. Users (crew) interactively make updates to the knowledge-base, and the system asks questions and clarifies facts as inconsistencies arise.

As a simple prototype, we are implementing a sufficiently rich subset of the above proposed expert database system. The system will be menu-driven so that users need not memorizing any commands. Due to space limitations, details of the underlying framework has been left out. Each one of the papers [Gol-83, Gol-84, Gol-85, Gol-86] contains several related examples that demonstrate the simplicity of the notation of our language and the system.

3. REFERENCES

[ChNa-85] Chambers A B, Nagel D C
"Pilots of the Future: Human or Computer?"
IEEE Computer, Vol. 18, No. 11, pp 74-87, November 1985.

[Gol-83] Golshani F
"A Mathematically Designed Query Language"
Research Report DoC 83-1, Imperial College, London UK

[Gol-84] Golshani F
"Tools for the Construction of Expert Systems"
Proc. of 1st Intl. Workshop on expert database systems,

Kiawah Island, SC, October 1984.

[Gol-85]   Golshani F
            "Growing Certainty with Null Values"
            Journal of Information Systems, Vol. 10, No. 3, pp 289-98,
            1985.

[Gol-86]   Golshani F
            "Specification and Design of Expert Database Systems"
            In "Expert Database Systems" (Kerschberg, ed.),
            Benjamin Cummings Publ. Co., 1986, pp 369-381.

[ICAO-84]   International Civil Aviation Organization
            "Accident Prevention Manual"
            1st Ed. Doc. 9422-AN/923, ICAO, Montreal, 1984. (Cited in
            [ChNa-85])

Knowledge Elicitation for an Operator Assistant System in Process Control Tasks

Guy A. Boy[1], ONERA/CERT-DERA,
2 avenue Edouard Belin, 31055 Toulouse Cedex, France.

Abstract

This paper presents a knowledge-based system (KBS) methodology designed to study human machine interactions and levels of autonomy in allocation of process control tasks. In practise, users are provided with operation manuals (paper KBS) to assist them in normal and abnormal situations. Unfortunately, operation manuals usually represent only the functioning logic of the system to be controlled. The user logic is often totally different. User logic integration is difficult, long, incomplete, and sometimes impossible. This paper focuses on a method for eliciting user logic to refine a KBS shell called an *Operator Assistant* (OA). If the OA is to help the user during operations, it is necessary to know what level of autonomy gives the optimal performance of the overall man-machine system. For example, for diagnoses that must be carried out carefully by both the user and the OA, interactions are frequent, and processing is mostly sequential (analytical reasoning). Other diagnoses can be automated (situation patterns), in which case the OA must be able to explain its reasoning in an appropriate level of detail. The optimal level of autonomy can be determined experimentally following an iterative process: testing a specific level of autonomy / building the corresponding level of explanation / experimental evaluation. OA structure has been used to design a working KBS called HORSES (Human - Orbital Refueling System - Expert System). Protocol analysis of pilots interacting with this system has revealed that the a-priori analytical knowledge becomes more structured with training and the situation patterns more complex and dynamic. This approach can improve our a-priori understanding of human and automatic reasoning.

Keywords: Knowledge acquisition, user logic, levels of automony, human-machine interactions, protocol analysis, situational and analytical behavior, on-line expert system.

Introduction

Operators controlling dynamic systems use mechanical or human aids. On the one hand, they are provided with paper operation manuals or users' guides. On the other hand, additional aid often comes from other human operators called assistants, copilots, or collaborators. Knowledge-based system (KBS) technology permits the design of a new kind of operator assistance. It provides great flexibility in knowledge editing, modifying and managing. Transfering knowledge from a paper format into a computer format is not straightforward. First, using an operation manual is not the same as using the identical information presented on a computer. Second, a computer is more powerful than a simple set of pages. It is potentially able to provide very advanced processing. A paper operation manual can be seen as a knowledge base, while the human operator is an information processor. A KBS includes these two components: knowledge base and inference engine (processor). The main difficulty still remains to include the "know-how" of the operator in the KBS.

Generally, most operation manuals include only the functioning logic of the system to be controlled. These operation manuals are developed by operation engineers and designers. Most users find the operation manuals provided with computers difficult and boring, for example.

User logic is quite different and evolves rapidly with practice. It is generally very difficult to include this user logic in a paper operation manual for several reasons: complexity of concepts; difficulty of acquiring this knowledge and logic; volume of the corresponding knowledge, etc. This type of logic is acquired iteratively along with experience. Thus, expert systems iteratively modified with learning will be good candidates for acquiring this logic.

This paper introduces a methodology for implementing expert systems where situational behavior is an important feature in a diagnosis process. Beginners are shown as analytical information processors, while experts are usually much more situationally-oriented (Dreyfus, 1982). Moreover, in process control applications expert systems must deal with time. Monitoring is a very important part of diagnosis whenever more complex situation patterns result in less complex analytical reasoning. Situational pattern matching evolves with learning. A protocol analysis of pilots interacting with a KBS for on-line fault diagnosis (Boy, 1986) has shown how patterns become more sophisticated with training. The complexity of the patterns is related to the number, the flexibility and the dynamic aspect of parameters involved in the pattern. Analytical behavior is usually represented by IF-THEN rules. The resultant rule base can be represented by a tree. Skill acquisition can be simulated by transfer of parts of the analytical reasoning into situation patterns, i.e. learning from practice.

The main focus of this paper is to define situation patterns, and a method for eliciting them from the human expert. Basically, the concept of situation patterns is dynamic, i.e. each pattern is related to a quantitative or qualitative model developed by the user.

Definition of an Operator Assistant

Operator Assistant function can be defined in several ways. In an aircraft for example, a copilot shares the work with the pilot, but not the ultimate responsibility. The pilot can consult the copilot on any point concerning the flight but will take ultimate decisions. If the pilot delegates a part of his responsibilities to the copilot, then the copilot will take this delegation as a task to execute. Moreover, the pilot can stop the execution of a copilot task at any time, if it is necessary. However, a copilot may have personal initiatives, e.g. testing parameters, being aware of the actual situation, predicting deducable failures, etc. A copilot may process the information included in an operation manual at the request of the pilot. He should be able to explain, at an appropriate level of detail, results of his processing.

Human Operator Models

Rasmussen's performance model provides a good framework to analyze human-machine interactions at three levels of human behavior: skill-based, rule-based, and knowledge based (Rasmussen, 1984). Acquisition of skills, both sensori-motor and cognitive, is the result of long and intensive training. At the skill level the knowledge is *compiled.* Rule-based behavior is still an operative level dealing with specified plans. At this level the knowledge is *interpreted.* The knowledge-based behavior is the level of the intelligence. At this level the knowledge is based on general plans that can be *adapted.* Most actual expert systems work at Rasmussen's second level because it is not easy to acquire knowledge from the skill level, the real situational level of expertise (Dreyfus, 1982). The intermediate level is the easiest to formalize and elicit from expert explanations. Indeed, as teachers, human experts decompile their knowledge to explain the "how" and "why" of their own behaviors. The result of such decompilation is easily transferable to a computer with an "IF... THEN..." format. This does not necessarily capture expert knowledge and behavior. Rather, it provides an analytic representation of the expert's knowledge and logic.

The situational behavior of the human expert makes him fast in "problem solving" and unique. Unfortunately, this type of behavior has proved to be difficult to elicit from the expert. This is the famous bottleneck of knowledge engineering. Interview techniques for skill-based knowledge

132

elicitation are not appropriate, as they are no longer regulated at a representative level and it is difficult to specify the procedures used (Leplat, 1986). In contrast, direct observation of spontaneous behavior is certainly a very interesting technique.

Levels of Autonomy of an Operator Assistant System

Previous studies on supervisory control show automation as an interface between the human operator and the machine being controlled (Sheridan, 1984). Here, we introduce the KBS as a new element in *triangular interactions*, i.e., human operator – machine being controlled – KBS interactions. The KBS is intended to perform copilot tasks and "behaviors", as described previously. Moreover, the KBS is considered as a student and the human operator as a teacher. The problem then is to design a method for transferring the knowledge of the human expert (e.g., fault diagnosis) to the KBS.

The *Operator Assistant* system includes situation recognition and analytic diagnosis. At the begining of the elicitation process the situational part is very small and the analytic part huge. During the elicitation process, the analytic part is progressively "automated" and corresponding automated functions are transformed into "situational" patterns. To develop the elicitation method, the following levels of autonomy of the KBS, interacting both with a human and a machine, have been introduced (Boy, 1986).

- Level 0 refers to the classical paper operation manual, computerized or not. The KBS is reduced to a simple knowledge base.

- Level 1 is characterized by a KBS connected to the system being controlled. It helps by determining possible situations (contexts) which the user then selects. The KBS is able to start an analytical diagnosis interacting with the user. It asks the same questions, requests the application of the same procedures, and gives the same diagnoses as in the previous level. The user answers the questions, applies the procedures or not, and can stop the reasoning at any time

- At level 2, the KBS determines contexts and suggests one which the human operator "should" follow. All the possible contexts are presented with a likelihood judgement. In the analytic diagnosis process, some questions are automated, i.e. the KBS checks the truth of the corresponding requested information, and presents the selected answer to the user. Everytime a function is automated, a corresponding deeper knowledge is created in the KBS. This automation process may imply creation of a new situation pattern.

- At level 3, the KBS selects a context automatically. It presents procedures to apply and the user approves or not. If the user approves then the KBS applies the procedure and sends the message "executed" to the user.

- At level 4, the analytic diagnosis process of the KBS is implemented as follows: all the questions which do not necessitate a user confirmation are totally automated, the KBS checks everything without any user's confirmation, the procedures are applied automatically (the user does not see any procedure statement). The situation recognition part is totally automated.

- At level 5, everything is automated, the KBS gives its advice directely about the situation and the possible diagnoses. It is able to explain its reasoning.

The method for capturing expert knowledge progressively is based on the search for the optimal level of autonomy of the KBS, for a given level of knowledge of the man-machine interactions. A theoretical representation underlying the approach is shown in figure 1. The abscissa represents a continuum of levels of autonomy, from the operation manual to a totally automated Operator Assistant system, and the ordinate represents the performance of the human-machine system. The performance of the human-machine system can be related to the time pressure, the

133

cost of finding a solution to a given problem, or any combination of performance criteria. Each curve in figure 1 corresponds to a given level of knowledge. The lowest curve represents a poor level of knowledge. In this case, we assume that the performance of the human-machine system increases with the autonomy of the KBS (i.e. KBS behavior will become more and more straightforward) until an optimum is reached. After this optimum, taking into account the assumed poor level of knowledge, if the autonomy of the KBS keeps increasing, then generally the user is going to be confused and may frequently lose the control of the situation. This phenomenon is represented by a decreasing curve following the optimum. The optimum can be moved to the right if the user is very well trained, which means that the level of knowledge of the user on the system to be controlled has increased. It is necessary to keep in mind the possible adaptation of the human to the machine. The present goal, however, is to provide an assistant to the user. Thus, we want to increase the level of knowledge of the KBS. In trying to find the optimum level of autonomy for a given level of knowledge, it is obvious that the level of knowledge is going to increase. Thus, in studying differents levels of autonomy, we are going to jump iteratively from one curve to a higher one. Ultimately, if everything is known about the system to be controlled under all operating conditions, then the optimum corresponds to complete autonomy, i.e. the upper curve of figure 1.

Knowledge Representation and Processing

An OA system includes two kinds of processes: situation recognition and analytic reasoning. The first process is implemented as a pattern matching. The patterns are called situations. The second process is a rule-based inference system including basic rules and meta-rules. A chunk of knowledge in an OA is represented in figure 2.

Situational Knowledge

A *situation pattern* is a location or position with reference to environment, a condition, a case, a combination of circumstances, a state of affairs of special or critical significance in the course of a play

Let the situation pattern S_j defined by a set of facts $\{ \mu_{i,j} \}$ and a aggregation operator π_j:

$$S_j = \{ \mu_{1,j} \dots \mu_{n,j} / \pi_j \}$$

where $\mu_{i,j} = \{ v_i, TF_{i,j}, \langle SP_{i,j} \rangle \}$,

v_i is the (numerical or logical) variable "i",

$TF_{i,j}$ is the tolerance function of v_i in S_j,

$\langle SP_{i,j} \rangle$ is a set of "useful" specific parameters,

e.g., the quality of information given by v_i to define S_j.

A *tolerance function* (TF) is an application of a set V into the segment [0,1]. The elements of V are called "values", e.g. the values indicated by a numerical variable. V is divided in three sub-sets: preferred values $\{TF(v)=1\}$, allowed values $\{0<TF(v)<1\}$, and unacceptable values $\{TF(v)=0\}$. The actual model of tolerance function is able to take into account logical variables, e.g position of a valve or truth of a statement, and numerical variables, e.g. temperature or pressure. An ordinal scale has been defined on [0,1] in accordance with the above definition of the three sub-sets. This definition is appropriate where natural language concepts must be modeled and manipulated. In fact, a tolerance function is a membership function related to the concept tolerance. This is a fuzzy representation of a variable (Zadeh, 1983). The major attraction of this representation is found in its ability to handle the vagueness inherent in categories of language, e.g. "the pressure P1 is close to the maximum".

When a situation pattern matches the actual situation the system automatically infers a *context*

In OA, a context is defined as a field of application of a sub-set of analytic rules (island of analytical knowledge).

Analytical Knowledge

OA analytical knowledge includes *objects, functions, rules* and *meta-rules* An object can be a *numerical* or *logical variable*, a *question* from the system to the user, *procedure* to apply (by the user or by the KBS), and *diagnosis* generally given by OA. These objects have a frame representation. Several functions have been implemented (Boy, 1984), e.g., *ask, check*, and *send* The user can create new functions very easily. A rule or a meta-rule includes a list of contexts in which the rule is valid, a symbolic statement, and a descriptive statement. Meta-rules are used by the processor to select an appropriate strategy in the inference process on the selected rules. Some meta-rules are accessible to the user, e.g. the user can change from a forward chaining strategy to a backward chaining strategy. Others are automatic, i.e. a meta-rule clears away sub-sets of rules not relevant to the actual state of the situation.

Knowledge Processing

The basic structure of OA is a multi-level classical Information Processing System (Newell and Simon, 1972). It includes a Processor, a Knowledge base, and an Interface (with the operator and the system to be controlled). The processor includes two main parts: the situation recognition module (monitoring) and the diagnosis inference engine. A third part is being built: a procedure executor module. OA does not take into account plan generation. It is assumed that this part is taken into account by the user himself. The situation recognition module is a fuzzy pattern matcher, i.e facts coming from the process being controlled are matched to critical situation patterns. At each instant, if a situation is matched, then one context is infered and the corresponding sub-set of diagnosis rules is selected. When one or several sub-set(s) of rules is (are) active, the diagnosis inference engine works on it (them) until a result is found. Both forward and backward chaining are available in the analytic part of OA. When the system works autonomously the basic strategy is forward chaining. The user can interrupt the inference process at any time He or she can ask for explanation. He may ask OA to verify the truth of a diagnosis, while the KBS is working in forward chaining. The inference is symbolically uncertainty-driven, i.e. the uncertainty is not characterized with numbers. Everytime the user is uncertain about an answer to a question from OA, OA takes this answer as "uncertain" and eventually backtracks to this "uncertainty node" in the event of a bad diagnosis. The user interface is easy-to-use and menu-driven.

Experiments

Experimental Support

The Orbital Refueling System (ORS) used in the space shuttle to refuel satellites has been chosen as an example of a system to be controlled. An ORS simulator, a malfunction generator, and an Operator Assistant KBS, called Human-ORS-Expert System (HORSES), were used to implement knowledge acquisition experiments (Fig.3). These functions are concurrent processes, communicating through a shared memory feature on a MASSCOMP computer. The malfunction generator generates simulated malfunctions for the ORS, and introduces them at appropriate times into the ORS simulation. Graphical interfaces are implemented on an IRIS 1200 graphic system and a MASSCOMP workstation. They provide a good tool for eliciting user knowledge to improve HORSES.

Experimental Protocols

Two groups of four pilots participated in the experiment. The first group was naive in ORS operations whereas the second was knowledgeable in that domain Sessions were videotaped for subsequent analysis. Pilots were asked to oversee the transfer of fuel from one container to another and were given simulated displays and controls as they are currently implemented in the

135

ORS. Each pilot was involved in five sessions of three hours, performed on five days. The first session was training only. Fifteen runs were proposed to pilots during the three following sessions. For each run there was a probability of getting a failure. Faults were leaks, sensor biases, and avionics failures. During these experiments, two levels of autonomy of the KBS have been tested: level-0 and level-1. The last session was a debriefing during which individual interviews were performed.

Experiments involved subjects being asked to think aloud while interacting with ORS. Protocols (the chronological history of all the pilot's actions and system responses) were transcribed for later analysis. Verbal reports were very useful to understand what variables and processes were used when pilots performed diagnosis tasks, i.e. the analytic reasoning. But, as the situation recognition process is very interconnected with the diagnosis inference, verbal information from the pilots during the process control task was also very useful for identifying situation patterns. Verbalizing information is shown to affect cognitive processes only if the instructions require verbalization of information that would not otherwise be attended to (Ericsson and Simon, 1980). Thus, the experimenter did not ask any question during operations.

Results and Discussion

Results are essentially based on protocol analysis. They rely on the use of three criteria: the importance of an event during a run, its frequency, and the number of pilots dealing with it (Robert, 1985). In the analysis, the following three topics were considered: operation manual versus KBS; beginners versus experienced users; and SRAR model (see below) for fault identification. As the number of subjects was very limited, the results reported here should be considered as tendencies. They are primarily intended to illustrate the utility of the technique.

Operation manual versus KBS

Pilots using the operation manual could be separated in two groups. The first group used an *operation-manual-based approach*: "Pressure P1 is not normal, it sould be according to the graph, I take the procedure to diagnose the fault". They recognized critical situations by matching observations to situation patterns included in the operation manual. They used the operation manual very carefully and diagnosed faults according to the knowledge available in the operation manual. Detection and subsequent reasoning were very straightforward. The second group used an *intuition-based approach:* "Losing pressure P1, I have a leak in tank 1, close immediately valve 4". They recognized situations, not necessarily included in the operation manual, and diagnozed the fault without using the operation manual. The first group was slower than the second in diagnosing faults, when the fault was easy to detect. The second group failed more often. The first group always applied procedures, whereas the second was generally problem solving oriented.

Pilots using the KBS as an Operator Assistant could be divided in two groups. *situation recognizer users* and *diagnosis processor users* In the former group, pilots used the KBS as a situation recognizer They tended to rely on KBS advice about the situation. However, they also used the explanation facility very often. They generally used the KBS with a forward chaining strategy In the latter group, pilots used the situation recognizer after having detected a problem by themselves. They acknowledged the results given by the situation recognizer and started the corresponding analytic processing. These pilots generally guessed a diagnosis and asked the KBS to verify it (backward chaining). The situation recognizer users can be associated with operation-manual-oriented users, and the diagnosis processor users with intuition-oriented users.

Beginners versus Experienced Users

Experienced people monitored the system being controlled with more sophisticated situation patterns than beginners. Once they recognized a situation, experts implemented short sequences

of tests. The analytical reasoning they employed at this stage is minimal, by comparison with beginners. Experts and beginners also implemented this necessary analytical reasoning differently. Beginners followed the operation manual flowcharts in a forward chaining fashion. Experts tested possible diagnoses or new situation patterns for which analytical reasoning was familiar, i.e. they implemented a backward chaining process.

SRAR Model

From the experiments to date, it is evident that people use *chunks* of knowledge to diagnose failures (Fig. 2). A chunk of knowledge is fired by the matching of a situation pattern with a percieved critical situation. This matching is either total or partial. After situation recognition, analytic reasoning is generally implemented. This analytical part gives two kinds of outputs: diagnosis or new situation patterns. Fault identification can be represented by the *Situation Recognition / Analytical Reasoning* (SRAR) model (Fig. 4).

The chunks of knowledge are very differents between beginners and experts. Beginners' situation patterns are poor, crisp and static, e.g., "The pressure P1 is less than 50 psia". Subsequent analytical reasoning is generally important and time-dependent. On the one hand, when a beginner uses an operation manual to make a diagnosis, his or her behavior is based on the *pre-compiled engineering logic* he has previously learned. On the other hand,, when he or she tries to solve the problem directly, the approach is very declarative, i.e., using the system first principles. With practice, beginner subjects were observed to get a personal procedural logic (user logic), either from the pre-compiled engineering logic or from a direct problem solving approach. Neves and Anderson (1981) called this process *knowledge compilation* Conversely, experts' situation patterns are sophisticated, fuzzy and dynamic, e.g., "During fuel transfer, a fuel pressure is close to the isothermal limit and this pressure is decreasing". This situation pattern includes many implicit variables defined in another *context*, e.g., "during fuel transfer" means "in launch configuration, valves V1 and V2 closed, and V3, V4, V7 open". Also, "a fuel pressure" is a more *general* statement than "the pressure P1". The statement "isothermal limit" includes a *dynamic* mathematical model, i.e., at each instant, actual values of fuel pressure are compared *fuzzily* ("close to") to this time-varying limit [P_{isoth} = f(Quantity, Time)].

Moreover, experts take this situation pattern into account only if "the pressure is decreasing", which is another dynamic and fuzzy pattern. It is obvious that experts have transferred part of analytical reasoning into situation patterns. This part seems to be related to dynamic aspects. Thus, dynamic models are introduced in the situation patterns, with leaning. It is also clear that experts detect broader sets of situations. First, experts seems to fuzzify and generalize their patterns. Second, they have been observed to build patterns more related to the task than to the functinal logic of the system. Third, during the analytical phase, they disturb the system being controlled to get more familiar situation patterns, which are static most of the time, e.g., in the ORS experiment, pilots were observed to stop fuel transfer after recognizing a critical situation.

Conclusion

This study demonstrates that protocol analysis can be used to elicit user knowledge. In the experiments with the ORS, the "acquired" knowledge from different sources, i.e.. different pilots and behaviors, was combined. However, both the protocol analysis and the standardization procedure were done.manually and were very slow. Protocol analysis also has the disadvantage that it gives very low-level information on the situation recognition and analytic reasoning processes.

On the other hand, deep knowledge decompilation can be very difficult and sometimes impossible with available tools. The separability of meaningful chunks of knowledge from the whole is a major problem, particularly in fields such as medicine where the system to be analyzed is not man-made. In the case of man-made systems, however, it should be possible to obtain chunks of deep knowledge during system development and testing. With the aid of an appropriate

knowledge design tool, the design engineer would archive the knowledge he uses during the development of the system to be controlled. His deep "decompiled" knowledge of the functioning of the system would provide the basis for an OA which could then be tested in triangular interactions with users and the system to be controlled, to establish the optimal level of autonomy of the OA

References

Bov, G.B., (1986), "An Expert System for Fault Diagnosis in Orbital Refueling Operations", AIAA 24th Aerospace Sciences Meeting, Jan., Reno, Nevada.

Dreyfus, S.E., (1982), "Formal Models vs. Human Situational Understanding: Inherent Limitations on the Modeling of Business Expertise", *Office: Technology and People*, 1, pp. 133-165

Ericsson, K., & Simon, H, (1980), "Verbal Reports as Data", *Psychological Review*, 87, 3, pp. 215-251.

Leplat, J., (1986), "The Elicitation of Expert Knowledge", NATO Workshop on *Intelligen Decision Support System*.

Robert, J.M., (1985), "Learning by Exploration", *2nd IFAC/IFIP/IFORS/ IEA Conference or Analysis, Design, and Evaluation of Man Machine Systems*, Villa Ponty, Varese, Italy, September 10-12.

Sheridan, T.B., (1984), "Supervisory Control of Remote Manipulators, Vehicle and Dynamic Processes: Experiments in Command and Display Aiding", in *Advanced in Man-Machine Systems Research*, JAI Press inc., Vol. 1, PP. 49-137

Zadeh, L.A., (1983), "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems", *Memorendum No. UCB/ERL, M83/41, Electronics Research Laboratory*, College of Engineering, University of California, Berkeley, CA.
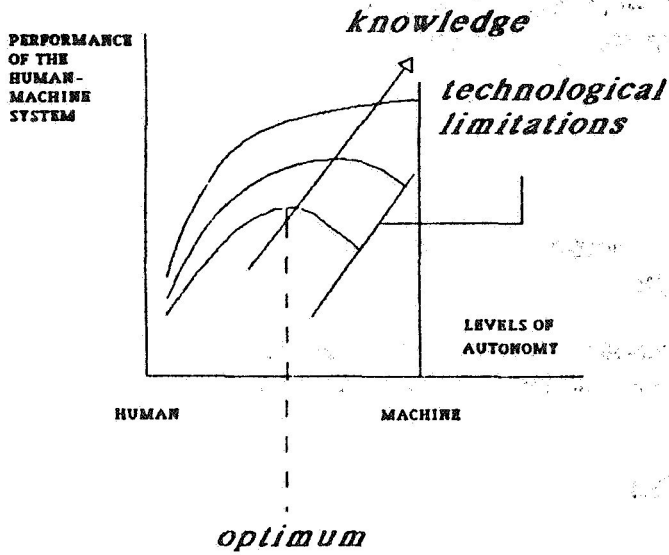
Figure 1. Levels of Autonomy vs.
Human-Machine System
Performance
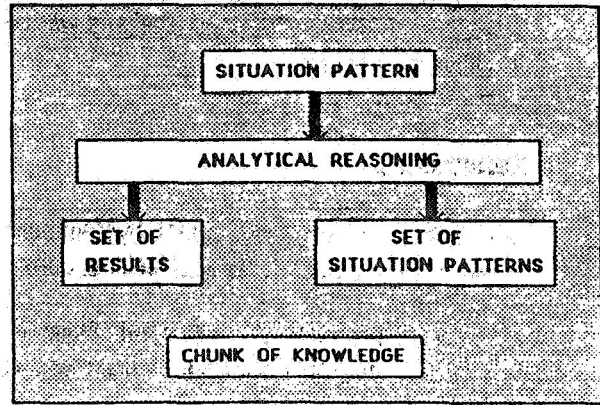
RESULT : - DIAGNOSIS
- ADVICE
- PLAN

Figure 2 : Chunk of Knowledge in OA
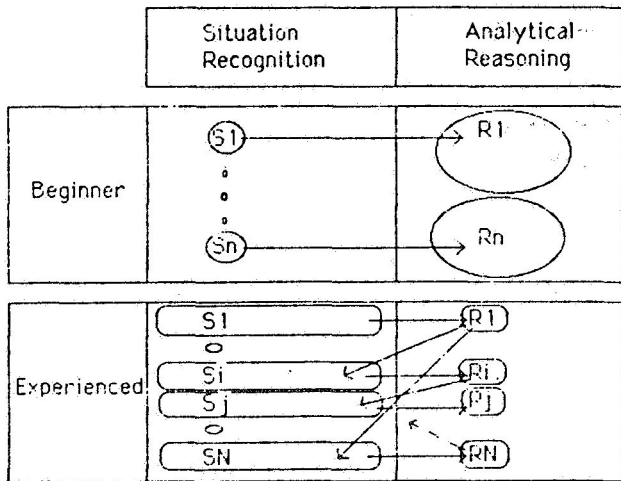
Figure 4 SRAR Model

Beginner a few small situation patterns and
huge analytical reasoning islands

Experienced users many complex and fuzzy
patterns and analytical knowledge
organized as small islands
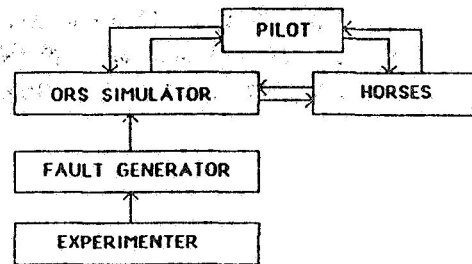
Figure 3. Experimental Support

139

MACHINE VISION FOR REAL TIME ORBITAL OPERATIONS

ABSTRACT

Machine vision for automation and robotic operation of Space
Station era systems has the potential for increasing the
efficiency of orbital servicing, repair, assembly and docking
tasks. This report describes a machine vision research project
in which a TV camera is used for inputing visual data to a
computer so that image processing may be achieved for real time
control of these orbital operations. Classical image analysis
techniques were investigated and found to be much too slow for
typical operations. A technique has resulted from this research
which reduces computer memory requirements and greatly increases
computational speed such that it has the potential for develop-
ment into a real time orbital machine vision system. This tech-
nique is called "AI BOSS" (Analysis of Images by Box Scan and
Syntax) and its description is included in this paper.

Submitted for presentation at:

Conference on Artificial Intelligence for Space Applications
Space and Rocket Center Marriott Hotel
Huntsville, Alabama 35805
November 13 & 14, 1986

Sponsors:

NASA - Marshall Space Flight Center
University of Alabama in Huntsville

Author:

Frank L. Vinz
NASA-MSFC
EB44
Huntsville, Alabama 35812

MACHINE VISION FOR REAL-TIME ORBITAL OPERATIONS
by
Frank Vinz

INTRODUCTION

The development of machine vision techniques for industrial automation has been a long standing goal which has only been met with restricted success. Machine vision is used to denote the ability of a device to process visual information so that scene interpretation can be made and decisions result which allow the device to accomplish a non-trivial task. Video systems are a natural choice for the sensing element of a machine vision system because they are commonly available, can be made to function in diverse environments and are relatively cheap. However, a TV camera can acquire such a vast amount of visual data in such a short period of time that processing systems may not have the ability to store all visual data nor could they perform calculations fast enough for scene analysis. The solution of these problems is fundamental to the development of machine vision.

This report describes a particular application of machine vision which has potential for use in advanced versions of the Orbital Maneuvering Vehicle (OMV) that will be in operation during the Space Station era. A classical two-dimensional Fast Fourier Transform signal processing technique for possible use in machine vision systems is summarized. A much faster technique has been developed by the author which employs a syntactic pattern recognition scheme. It has the potential for reducing data storage requirements by perhaps one order of magnitude and providing an even greater decrease in computation requirements.

MACHINE VISION REQUIREMENT

A likely application of machine vision capability is for orbital docking, servicing and repair. Many of these tasks are to be accomplished by the OMV as illustrated in Fig. 1. It would be carried to low earth orbit by the Space Shuttle and manually operated from a remote ground station during critical station keeping and docking maneuvers. Visual data for the ground operators will be provided by onboard TV cameras whose video signals are transmitted to the remote control station by means of several communication links. Docking maneuvers can be difficult tasks for a human operator to perform for a number of reasons such as:

(1) The OMV must be flown in six degrees-of-freedom (ie, x,y, and z translations and roll, pitch, and yaw rotations).

(2) The target vehicle may be disabled so that docking alignment aids such as transponders or light patterns do not function.

(3) The target vehicle may have lost its attitude stabilization capability and could be spinning, coning, or tumbling. This is almost certain to be encountered on debris capture operations.

(4) Anticipated time delays of up to 2 seconds needed for transmission of the TV [1] picture can seriously complicate remote control during critical alignment operations just prior to docking mechanism engagement.

142
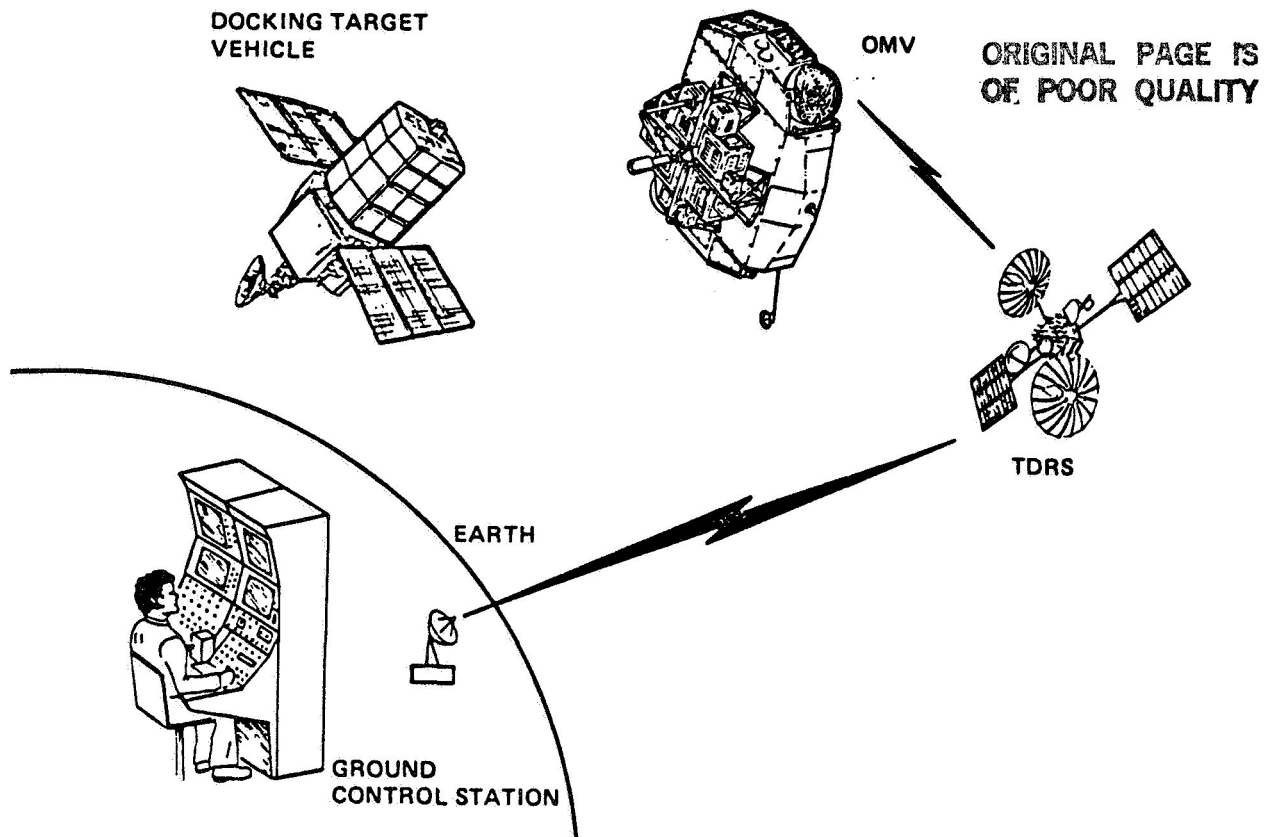
FIG 1 OMV BASELINE OPERATION

The onboard TV camera is being considered as the sensor input to the
machine vision system since video is an OMV baseline requirement.
Thus machine vision for automatic docking is a viable evolutionary
growth possibility providing the additional onboard processing
requirements are reasonable. A machine vision capability for the OMV
offers the following autonomous operation advantages:

(1) Independence from the operation of docking alignment aids
    such as transponders or light patterns.

(2) Independence from communications links including TDRSS.

(3) Communication time delays for vehicle control are eliminated.

(4)Large costs associated with mission control operations and
    remote control operator training are either eliminated or
    substantially reduced.

SCENE ANALYSIS BY FAST FOURIER TRANSFORMS

An initial investigation of machine vision techniques included
frequency transform methods. Capabilities of an ideal machine vision
system for recognizing orbital vehicles include:

(1) Independence of the target image position in the field of
    view.

(2) Independence of orientation (ie,the image may be upside down
    or sideways).

143

(3) Independence of relative size of the image (ie, distance to the target is not critical for identification).

(4) Independence of the image illumination intensity.

(5) Independence of the apparent distortion of a 3-D object when viewed in 2-D by oblique or perspective viewing angles.

(6) Immunity to the background against which the image is viewed.

Elementary machine vision techniques such as template matching used in assembly line inspection do not provide these capabilities (Ref. 1). However Fourier transforms of an image are independent of the objects position and orientation (1 and 2 above). By normalization, they can be made to be independent of relative size as well (3 above). The use of Fourier transforms provide some reduction in the difficulty of recognizing images because of this invariance to displacement, orientation, and size.

The computation requirements for use of Fourier transforms can be substantial. A reasonable amount of picture resolution for identification of orbiting vehicles requires on the order of 256 horizontal by 256 vertical picture elements or pixels. A single TV picture with this resolution would contain 65,536 pixels. The calculation of two-dimensional (2-D) discrete transforms of this many points requires 4,295,000,000 multiplications. Use of the fast Fourier transform (FFT) technique reduces these multiplications to 1,049,000 or a reduction of about 1/5000 (Ref. 2, 3 & 4). The computation time of image identification for dynamics even as slow as orbital docking requires completion at least once a second. An array processor was used 2D-FFT calculations of simple video images. Although the array processor had the capability of processing 30 million floating point operations per second it never the less required several seconds to complete a 2-D FFT of the 256 x 256 elements contained in a single video frame. A comparison of 2D-FFT calculations of several simple binary images was made and the results are shown in Fig. 2.

SCENE ANALYSIS BY SYNTACTIC PATTERN RECOGNITION

An alternative to the Fourier transform approach to image analysis and target vehicle identification has been developed using techniques derived from syntactic pattern recognition. The origin of syntactic pattern recognition can be traced to the study of formal language theory by Noam Chomsky in the 1950's and to the early attempts to develop computer based language translators (Ref.5). This technique employs a tree graph for representation and description of a scene. It is considered to be a very efficient way to characterize an object in a scene since only a small fraction of the total image data need be permanently stored. There is a corresponding reduction in computation requirements as compared to the 2-D FFT technique.

The syntatic method devised is unique in that it does not have the limitations of possibly omitting major sections of the image which may be the case with vertical or horizontal line scanning reported in the
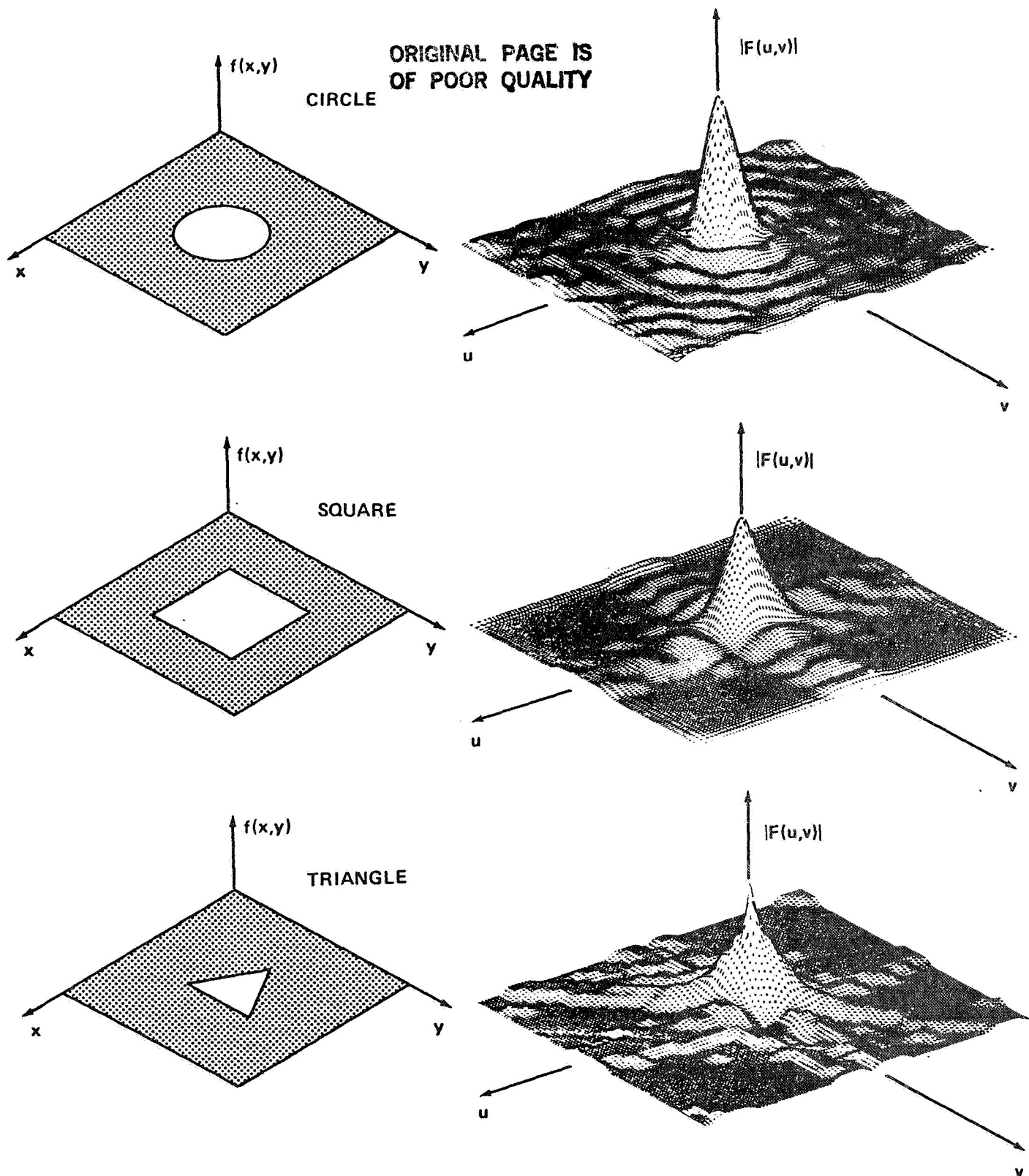
BINARY IMAGES, f(x,y), AND THEIR FOURIER SPECTRA, F(u,v)

FIG 2

145

literature. To prevent this possible omission, a box scan is employed which starts at the centroid of the image and increments outward in larger and larger layers until the complete scene is examined. An example of this scan process on a 8 x 8 scene matrix is shown in Fig. 3. The scene includes a "T" image consisting of 16 binary "1's" while all other possible pixels are background elements having low level video intensity and have been pre-processed as binary "0's". The complete scene is examined by a 3 x 3 scan window using the box scan technique and the resultant tree graph is also shown in Fig. 3. The longest branches of the tree have end points which correspond to the major end points of the "T" image on the 8 x 8 scene. This tree graph representation has a capability of uniquely describing an image without requiring storage capacity for the complete 8 x 8 matrix scene. This capability has significant reduction in memory for high resolution scenes having a large number of pixels as a 256 x 256 matrix. An even greater significance is the reduction of computations required for image recognition as will be shown. The method is referred to as Analysis of Images by Box Scan and Syntax or "AI BOSS" and it is felt to have a good potential for autonomous control of orbital servicing vehicles during critical docking and assembly operations.

Specific procedures for implementing the "AI BOSS" technique involves the following steps:

(1)   By means of a computer-compatible TV camera having pre-processing for binary slicing of a video image, obtain a digitized representation of the scene. Each pixel representing the target image is denoted by a binary "1" and the background pixels are a binary "0".

(2)   Compute the centroid pixel of the target image.

(3)   Create the root and initial branches of the tree graph by centering a 3 x 3 pixel scan window at the centroid. Fig. 3 illustrates this process. Proceed to examine in numerical sequence each of the 9 pixel locations of the scan window. The centroid pixel will be the root "R" of the tree graph providing it is a binary "1". The initial branches of the tree graph are any of the surrounding 8 pixels of the scan window which are also a binary "1". The branches are placed on the tree graph in a left to right order in the sequence in which they were scanned. Each tree graph element is identified by its corresponding scene matrix row and column. In case the centroid is not a binary "1" then the root of the tree graph is the first binary "1" which occurs in the box scan sequence after the centroid is examined.

(4)   The box scan is then moved so that the scan window is sequentially centered about the 8 pixel locations identified in (3) above. All binary "1" pixels on this relocated scan window which are horizontally, vertically, or diagonally adjacent (ie,neighbors) to previously identified tree elements are included on the graph as branches of these earlier identified elements. Each pixel so identified is drawn on

146

SCENE

TREE GRAPH

SCAN WINDOW

Tree graph of "T" image.

FIG 3

SCENE

TREE GRAPH

SCAN WINDOW
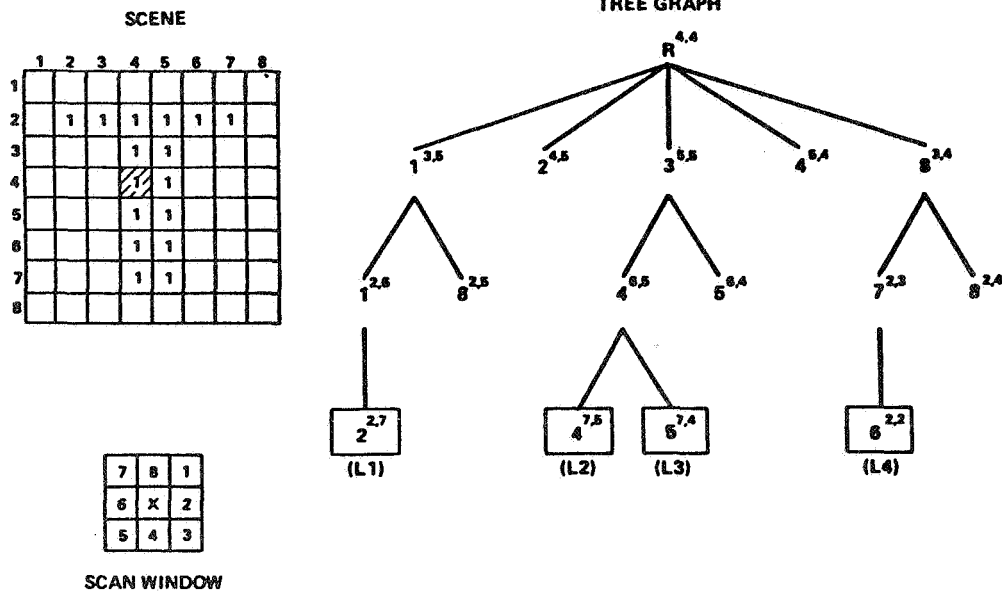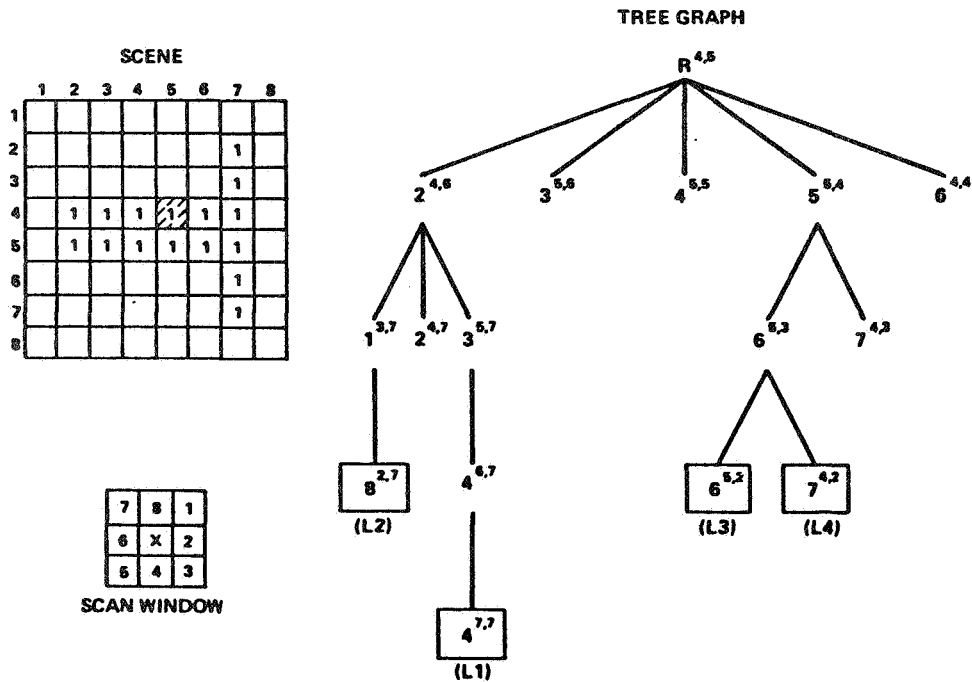
Tree graph of "T" image rotated 90 deg CW.

FIG 4

147

the tree graph and denoted by its row and column on the scene matrix as in (3) above.

(5) The box scan examines the next outer layer by incrementing one pixel to the right and one pixel up from the starting point of the preceding box scan layer and then performing a new box scan. This box scan procedure is depicted to the right. The scan process continues to grow until all pixels on the complete scene matrix are examined. In the case that the image centroid does not coincide with the center of the scene matrix, then the box scan must continue as though there existed additional pixel layers outside the scene. This insures that all scene pixels will be examined for potential inclusion in the tree graph.

FRAME EDGE
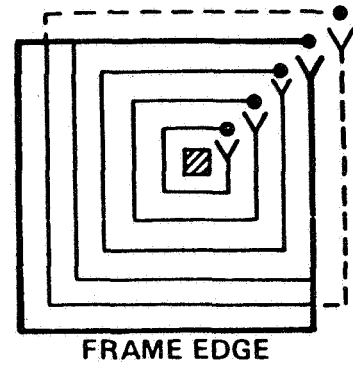
(6) After a pixel is identified as an element of the tree graph, it cannot appear elsewhere on the tree.

(7) Once elements of a tree graph are identified, subsequent scan window searches are made only from these elements. The order of scanning from these identified elements is dete mined by the sequence in which they were identified by the rules of the box scan. In this way, there will be fewer problems with disjointed elements in the process of building the tree graph.

(8) In the case pixels are found which are not neighbors to existing tree graph elements, then they are temporarily tagged for auxiliary searches to determine where they should be placed. Each "tagged" pixel will be examined after each pixel search in the box scan procedure. If a candidate pixel does not have a neighbor that is an element of the tree, it will continue to be tagged for auxiliary searches. In case several pixels are tagged for auxiliary searches, they will be examined for possible incorporation into the tree in the sequence by which they were originally tagged. After completion of all box scans and auxiliary searches, if tagged elements remain which have no neighbors that are elements of the tree graph, then these tagged elements are discarded and are considered to not be a part of the image. They may be the result of noise in the video picture or they may be bright background objects.

(9) An image is then characterized by the length of the major branches on the tree graph and by the branch end point locations (ie, the row and column numbers from the scene matrix). For the examples studied in which the frame matrix is 8 x 8, the four longest distinct branches were selected as major branches and were sufficient to identify the image.

148

Distinct branches are separated from each other by a maximum
number of tree elements and are also positionally spread out
over the tree shape. Refer to Fig 13 for an example of the
use of distinct branches. In the case where distinct bra
ches of equal length are in contention as major branches,
the order of selection is left to right as they appear on
the tree.

(10) For more realistic cases where the scene matrix may have
many more elements such as 256 x 256, then more than four
major branches may be selected to better characterize the
image. To provide distinction between end points in this
case, the major branches selected should have at least 10 or
20 unique elements at the end. Otherwise the branch end
points may correspond to adjacent elements of the image in
the scene matrix and would not be very effective describers.

(11) Images of orbiting targets may be recognized by the relative
length of each major branch. This constitutes a powerful
pattern recognition capability in that general shapes are
characterized by some of the same features a person uses
such as: elongation, compactness, symmetry, relative length
of appendages, aspect ratios, etc.

(12) The method exhibits a generally consistent mapping property
that is somewhat analogous to comformal mapping. The clock-
wise sequence of the image extremities in the scene matrix
is usually identical to the left to right sequence of major
end points on the tree graph. This characteristic is
clearly shown in Fig. 6 through Fig. 17.

(13) For those targets selected for closer inspection, servicing,
or docking, the chaser vehicle must rotate itself for rel
tive alignment by the use of the length and the end point
locations of the major branches. Comparison of both the
length of the major branches and the location of the end
points of each major branch will provide a template matching
for unique identification of the image.

The "T" image shown in Fig. 3 was examined for several orientations by
use of "AI BOSS". The following conditions were investigated:

Fig. 4 "T" pattern rotated 90 deg.CW.
Fig. 5 "T" pattern rotated 180 deg.CW.
Fig. 6 "T" pattern rotated 270 deg.CW.
Fig. 7 "T" pattern rotated 315 deg. CW.
Fig. 8 "T" pattern with 2 pixels shifted .

All of these cases consistently show four major tree graph branches
and all are approximately equal in length. The end points of each of
the tree branches correspond correctly with the extremities of the "T"
image on the scene matrix. This correspondence between branch end
points and image extremities is a strong feature of the "AI
BOSS" method and is a direct result of the box scan technique.

149
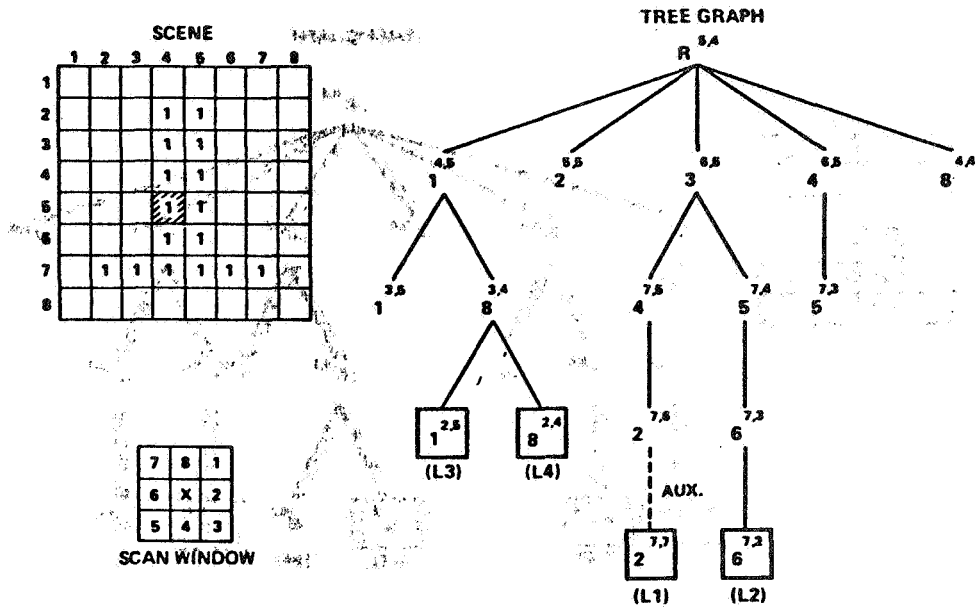
Tree graph of "T" image rotated 180 deg CW.

FIG 5
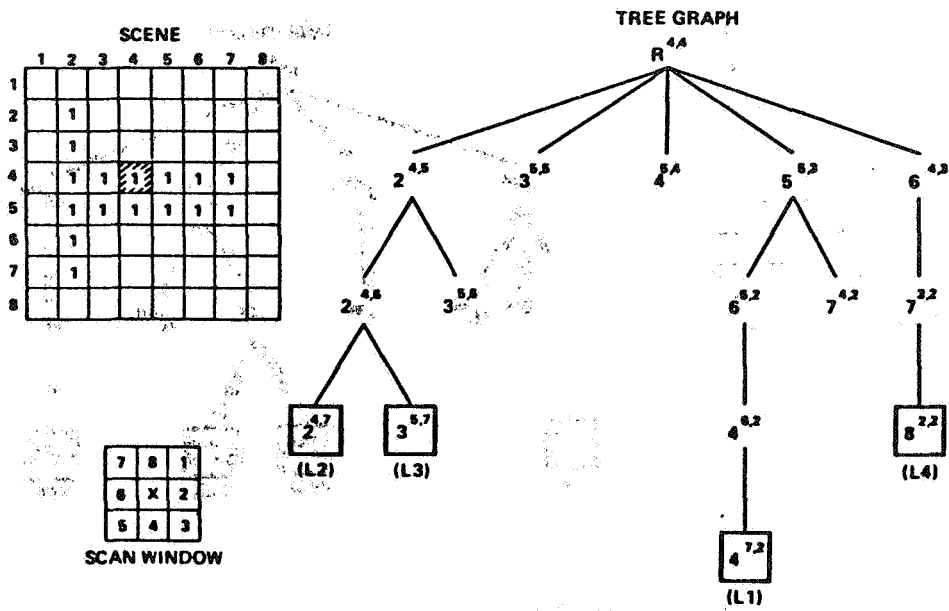
Tree graph of "T" image rotated 270 deg CW.

FIG 6

150

SCENE
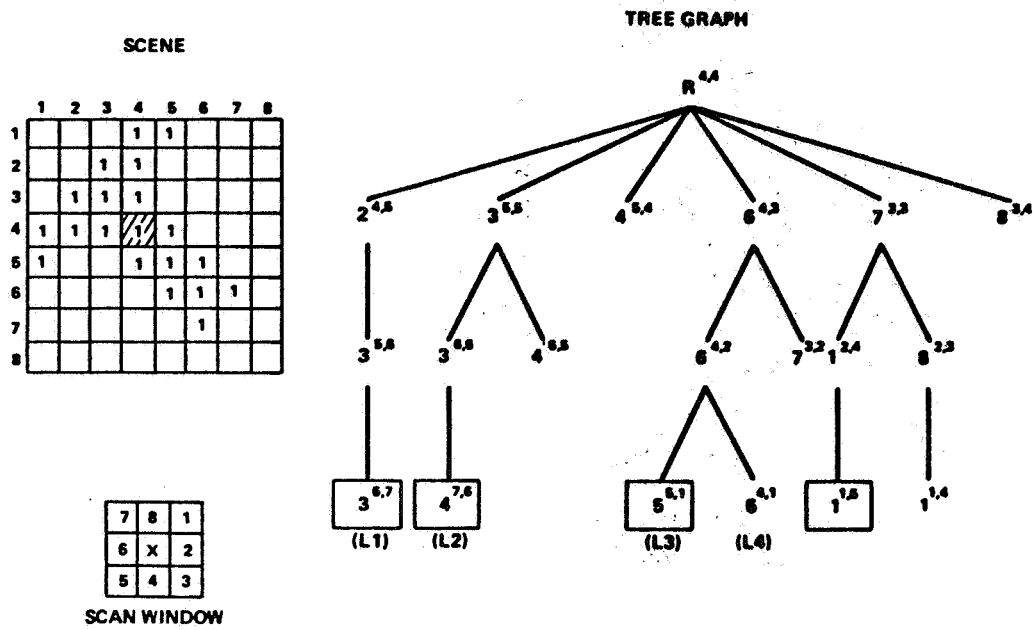
TREE GRAPH

Tree graph of "T" image rotated 315 deg CW.

FIG 7

SCENE
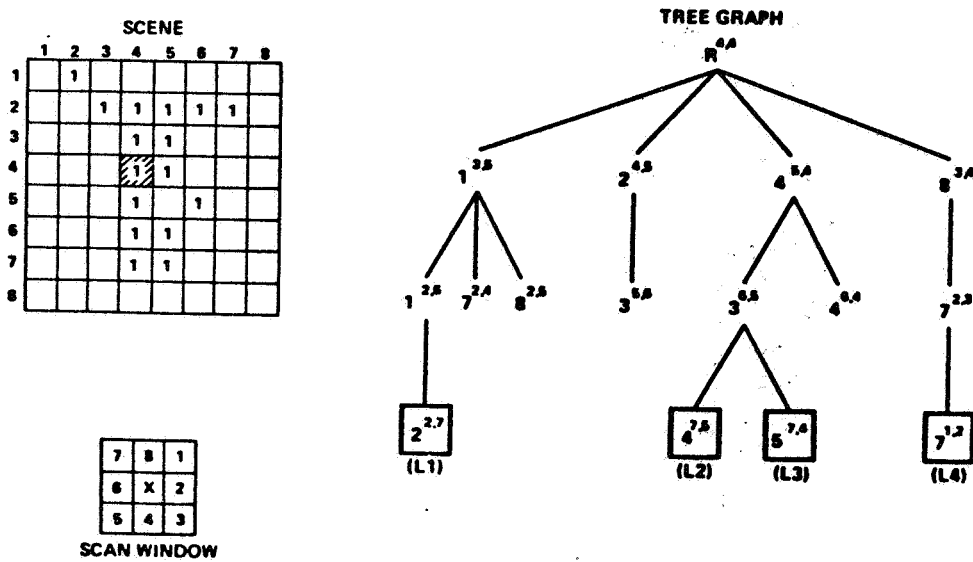
TREE GRAPH

Tree graph of "T" image with two pixels shifted.

FIG 8

Data obtained from comparisons of tree graph test cases with the reference tree graphs may be used to provide proper orientation between the chaser and target vehicles. Comparisons of branch length ratios and branch end points are used for performance measurements. The performance measurement may have plateaus and local minimums which must be contended with in the chaser vehicle roll search control law. An indication of the performance measurement for the "T" patterns is depicted in Fig. 9 for the five test cases studied.
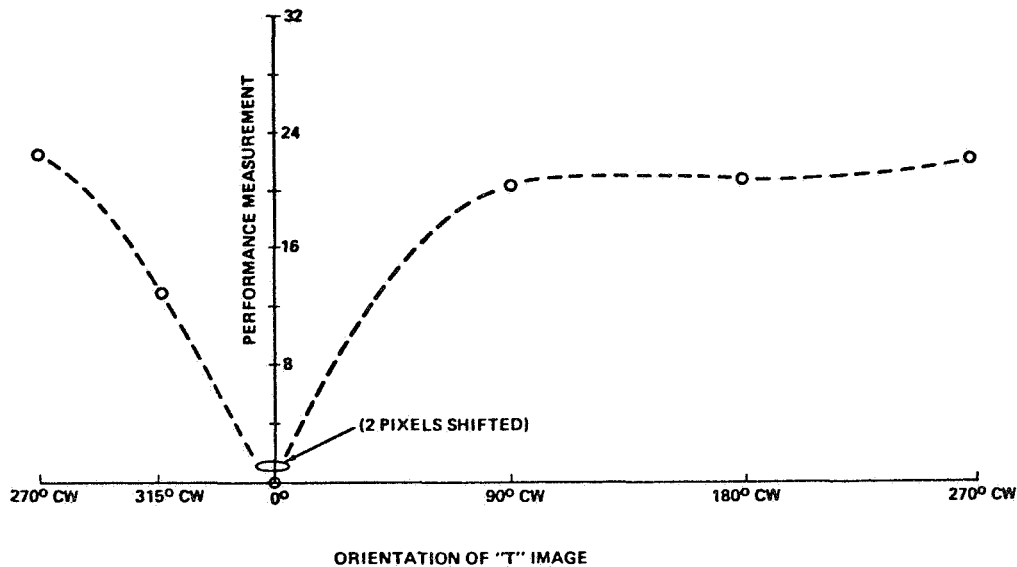
Performance measurement for chaser roll search of "T" pattern.

FIG 9

SCENARIO OF AUTONOMOUS DOCKING MISSION

Implementation of the "AI BOSS" machine vision technique on the OMV has the potential of providing a viable autonomous docking capability. (Ref. 6) The minimum equipment required to provide this OMV enhancement appears to consist of a computer compatible TV camera having 256 x 256 pixels, computer memory of 64 k for storage of a single video frame in real time (either 1/60 or 1/30 second), additional computer memory of about 16 k for image processing, computer processing speed fast enough to complete the "AI BOSS" computations in less than 1 second, and associated control laws for docking such as a profile of range versus range rate. All other requirements are OMV baseline requirements and are not unique for autonomous docking.

A scenario for autonomous docking by means of "AI BOSS" will include the following sequential procedures:

(1) Information on the orbits of the target and chaser vehicles will allow an automatic approach to less than one mile.

152

(2) As the target vehicle is visually acquired on the TV camera field of view, the chaser vehicle controls its pitch and yaw body rotations so as to keep the target centered.

(3) The approach to the target continues until the target image fills up an appreciable part of the TV field of view. Pitch and yaw commands must continue to maintain the target in the center of the view. Calculations of the image centroid by real time storage of video frames and at a speed of at least once per second will be required.

(4) The chaser will maintain a station keeping distance from the target by analysis of the video data. A border of perhaps 10 to 20 pixel widths on the outer portions of the video frame will be used to maintain the station keeping distance. The extremities of the target image must be maintained within this border yet they will not be allowed to shrink more than an additional 20 pixel widths inside the border.

(5) The chaser computer memory will have reference tree graphs of the target vehicle as viewed from representative directions. Typical reference views would be from each of six sides plus eight corners. The "AI BOSS" scan technique also must be employed for generating the reference tree graphs.

(6) When the station keeping distance is obtained, the tree graphs will be obtained for the target in question. By comparison with the length ratios of the major tree bra ches, the system will determine which reference view it is most likely near.

(7) A chaser vehicle roll search will be made to provide a better match to the selected reference tree graph. The search must also command chaser vehicle horizontal and vertical motions also in order to improve the performance measurement. While this search is in progress the station keeping distance as well as pitch and yaw centering must be maintained.

(8) The search must continue until a very good correlation with one of the reference tree graphs results.

(9) By recognizing which view has been matched, the chaser can then implement a stored maneuver to allow it to become aligned with the docking mechanism side of the target vehicle. The station keeping distance must be maintained until completion of this step. In case the target vehicle is not attitude stabilized, the chaser must command itself so as to track these uncontrolled motions in order to maintain this reference position.

(10) The reference image data is then switched to a different format corresponding to image information on the docking mechanism or docking alignment device. The approach to a

153

docked position is then accomplished with a pre-programmed closure maneuver by again using the image analysis, chaser vehicle roll orientation, and range control provided by "AI BOSS".

A procedure for a typical orbital docking operation is shown in Fig 10.

PRE- PROCESSOR	DETECTOR	FEATURE EXTRACTOR	CLASSIFIER AND AUTOMATIC SEARCH

PRE-PROCESSOR DETECTOR

| TV CAM |→| BINARY SLICER |

FEATURE EXTRACTOR

1. FIND CENTROID
2. DO BOX SCAN
3. BUILD TREE GRAPH

SET OF TREE GRAPHS FROM SELECTED VIEWS OF REF. TARGET

TREE GRAPH OF DOCKING AID ON REF. TARGET

CLASSIFIER AND AUTOMATIC SEARCH

1. ACQUIRE A DISTANT TARGET
2. APPROACH & CENTER WITH Θ & Ψ MOTIONS
3. AT SUITABLE RANGE INITIATE FEATURE EXTRACTOR
4. CORRELATE WITH REF. TARGET BY CALCULATION OF PERFOR- MANCE MEASUREMENT (PM) [F (BRANCH LENGTH, LOCATION OF END PTS.)]
5. SELECT MOST LIKELY VIEWPT. & HOLD FIXED X THRU 9.
6. AUTOMATIC SEARCH MANEUVERS TO MINIMIZE PM USING ϕ, Y, & Z MOTIONS
7. PREPROGRAMMED MANEUVERES TO MOVE TO DOCKING SIDE OF TARGET
8. SWITCH TO DOCKING AID REFERENCE
9. SEQUENTIALLY MINIMIZE PM USING MOTIONS:
    a. ϕ
    b. Ψ AND Y
    c. Θ AND Z
10. HOLD ϕ, Ψ, AND Θ AND MINIMIZE PM WITH Y & Z WHILE APPROACHING DOCKING LATCH WITH PREPROGRAMMED X.

FIG10 IMPLEMENTATION OF "AI BOSS" FOR AUTONOMOUS DOCKING

CONCLUSION

The application of syntactic pattern matching for autonomous operation of an orbital servicing vehicle can dramatically reduce onboard computer requirements. Machine vision for automatic orbital docking is a complex and computation intensive task using classical signal processing techniques such as 2-D FFT's. The "AI BOSS" technique described embodies powerful intuitive or heuristic pattern recognition capability by identifying such image shapes as elongation, compactness, symmetry, relative lengths of appendages, aspect ratios, etc. Yet "AI BOSS" requires much less computer memory and computation than classical signal processing techniques. Further investigation of this technique using a NASA orbital docking simulator will expand and quantify the merits of this unique approach.

154

REFERENCES

1. Cohen, Paul R. and Feigenbaum, Edward A.: The Handbook of Artificial Intelligence, Volume III. Heuristech Press, Stanford, California, 1982.

2. Paupoulis, Anthanasios: Signal Analysis. McGraw-Hill Book Company, New York, 1980.

3. Griffiths, J. W. R., Stocklin, P. L., and Van Schoonveld, C.: Signal Processing. Academic Press, London, 1973.

4. Gonzalez, Rafael E. and Thomason, Michael G.: Syntactic Pattern Recognition. Addison-Wesley, Reading, Massachusetts, 1978.

5. Gonzalez, Rafael E. and Wintz, Paul: Digital Image Processing. Addison-Wesley, Reading, Massachusetts, 1977.

6. Vinz, Frank L., Brewster, Linda L., and Thomas, L. Dale: Computer Vision for Real-Time Orbital Operations. NASA Technical Memorandum Number TM 86457, Marshall Space Flight Center, Huntsville, Alabama, 1984.

AUTOMATIC OBJECT RECOGNITION

H. S. Ranganath
Computer Science Department
The University of Alabama
in Huntsville
Huntsville, AL 35899

Pat McIngvale and Heinz Sage
U. S. Army Missile Command
Redstone Arsenal, AL 35809

Abstract

Geometric and intensity features are very useful in object recognition. An intensity feature is a measure of contrast between object pixels and background pixels. Geometric features provide shape and size information. This paper presents a model based approach for computing geometric features. Knowledge about objects and imaging system is used to estimate orientation of objects with respect to the line of sight.

Introduction

Geometric features of an object (target) are derived from its shape and size. Shape and size of an object as seen in a digital image depend on the following:

1. Position of the sensor (altitude and look-down-angle)
2. Horizontal and vertical fields-of-view (HFOV, VFOV)
3. Position of object within the field-of-view
4. Object orientation with respect to the line-of-sight

For example, the length (L) and height (H) of an object as seen in the image depend on its position within the FOV and its orientation. Therefore, L and H can not be taken as constants (This is also true for other geometric features). This paper presents a knowledge-based method for estimating object orientation with respect to the sensor's line of sight, and its position within the field-of-view.

PRECEDING PAGE BLANK NOT FILMED

Resolution Calculation

In this section horizontal and vertical pixel resolution on the image plane are calculated. The sensor is located at S at an altitude of h. The range to the center of FOV is R.

$$\therefore \text{ Look down angle} = \alpha = \sin^{-1}(h/R)$$

Let, Horizontal field-of-view = HFOV degrees

Vertical field-of-view = VFOV degrees

The image plane is perpendicular to the line of sight as shown in Figure 1. Let D be the distance of the image plane from the camera along the line of sight (LOS).

$$D = \frac{h \cos (VFOV/2)}{\sin (\alpha + VFOV)}$$

$$\text{Horizontal resolution (HR)} = \frac{2D \tan(HFOV/2)}{256}$$

$$\text{Vertical resolution (VR)} = \frac{2D \tan (VFOV/2)}{240}$$

HR and VR are pixel resolution on the image plane, not on the ground. The input image has 240 rows and 256 columns.
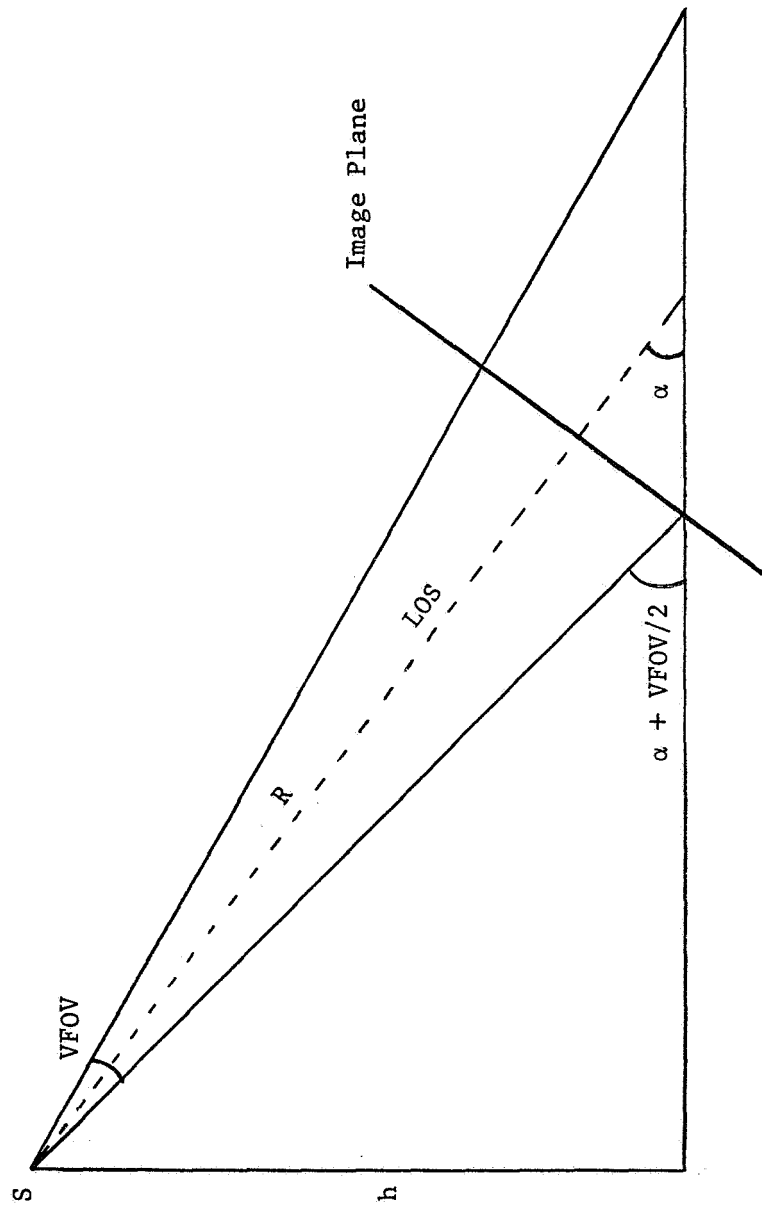
FIGURE 1. RESOLUTION CALCULATION

Projection of Target Onto the Image Plane

Shape and size discriminants can be computed from the projection of the target on the image plane. For example, the length in number of pixels is equal to the ratio of the extent of the target along x-axis to the horizontal pixel resolution. The steps involved in projecting the target on the image plane are given below.

Target Description:

The target to be recognized is described in its own local co-ordinate system as illustrated in Figure 2. A rectangular box of dimension 1 X W X h, used for simplicity, is described by its eight vertices.

$$P_1 = (0,0,0)$$
$$P_2 = (0,w,0)$$
$$P_3 = (1,w,0)$$
$$P_4 = (1,0,0)$$
$$P_5 = (1,0,h)$$
$$P_6 = (1,w,h)$$
$$P_7 = (0,w,h)$$
$$P_8 = (0,0,h)$$

If a tank's base and turret are modeled as two rectangular blocks, it can be described by sixteen points. More points may be used for better accuracy, if desired.

Aspect Angle:

If the targets aspect angle θ is known (usually not known, can be estimated from length and height), rotate the target about y-axis by θ degrees as shown in Figure 3. This rotation changes P_i to P^1_i, for i = 1,2,.., 8.

$$x^1_i = x_i \cos\theta - z_i \sin\theta$$

160
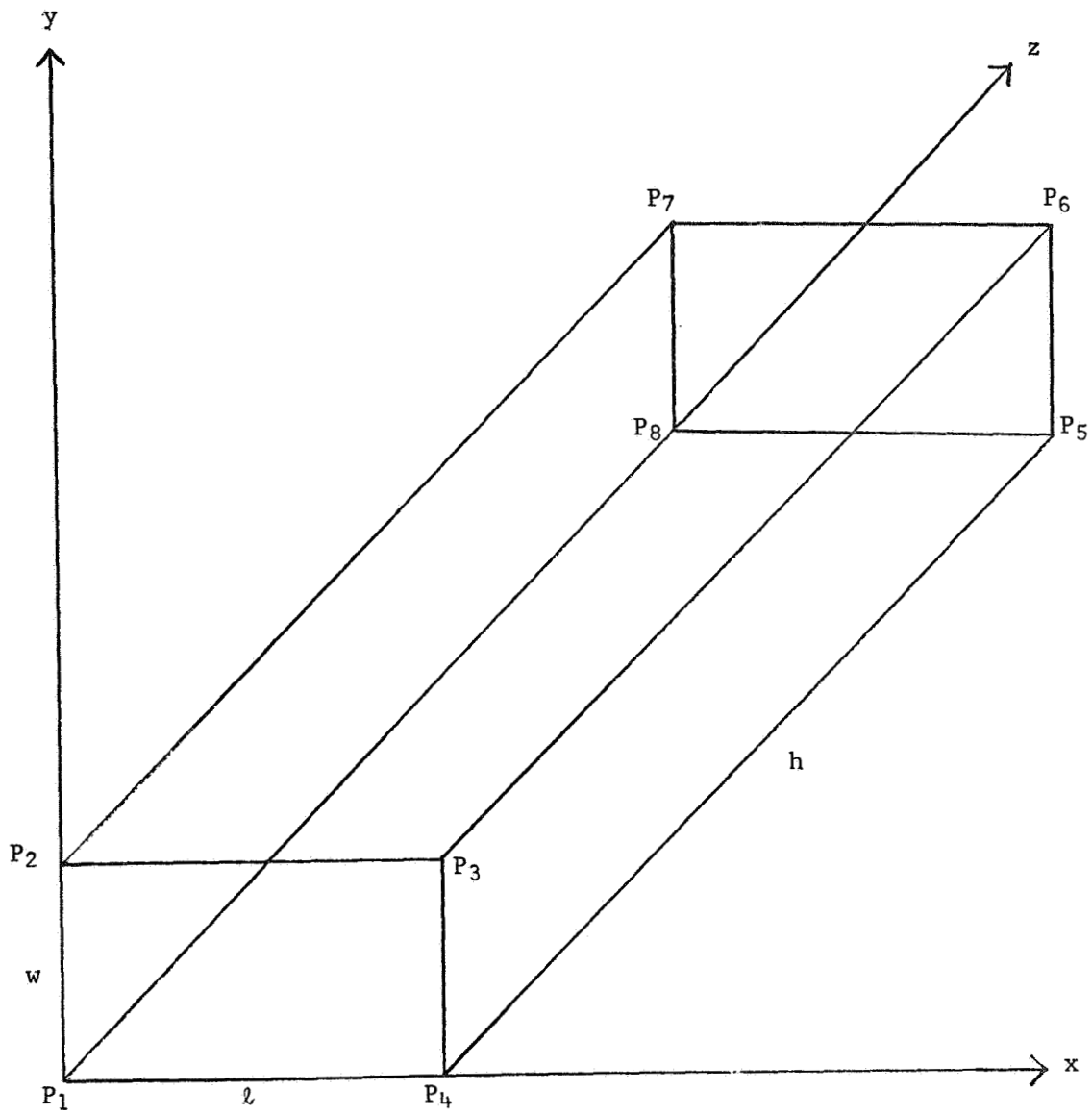
FIGURE 2. OBJECT DESCRIPTION
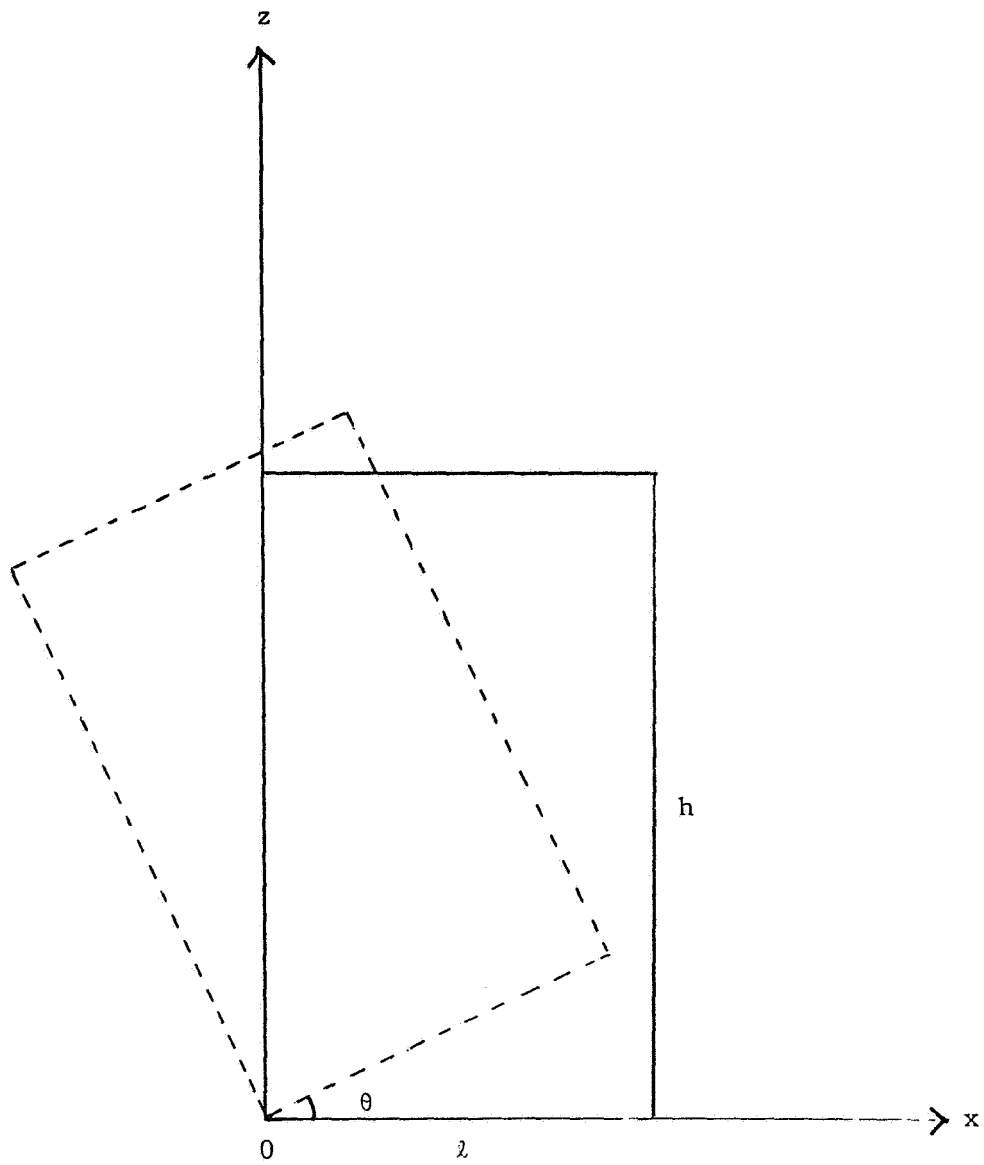
FIGURE 3.   ASPECT ANGLE

162

$$y^1_i = y_i$$

$$z^1_i = x_i \sin\theta + z_i \cos\theta$$

In the matrix form P^1_i is given by,

$$P^1_i = \begin{bmatrix} x^1_i & y^1_i & z^1_i \end{bmatrix} = \begin{bmatrix} x_i & y_i & z_i \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

Target in camera co-ordinates:

Transform the target to the camera co-ordinate system. The camera co-ordinate system is shown in Figure 4. xy-plane is parallel to the image plane and the z-axis is along the line of sight. The point P^1_i becomes P^2_i in the camera co-ordinate system. P^2_0 is still at the origin. This transformation is accomplished by rotating the yz object plane about x-axis by α degrees (look down angle).

$$x^2_i = x^1_i$$

$$y^2_i = y^1_i \cos\alpha - z^1_i \sin\alpha$$

$$z^2_i = y^1_i \sin\alpha + z^1_i \cos\alpha$$

In the matrix form P^2_i is given by

$$P^2_i = \begin{bmatrix} x^2_i & y^2_i & z^2_i \end{bmatrix} = \begin{bmatrix} x^1_i & y^1_i & z^1_i \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix}$$

Back projection:

Using back projection, locate a known vertex on ground and translate the target to the ground. The bottom left corner (IMAGEX, IMAGEY) of the potential target can be mapped to the ground. This translation changes P^2_i to P^3_i.

163
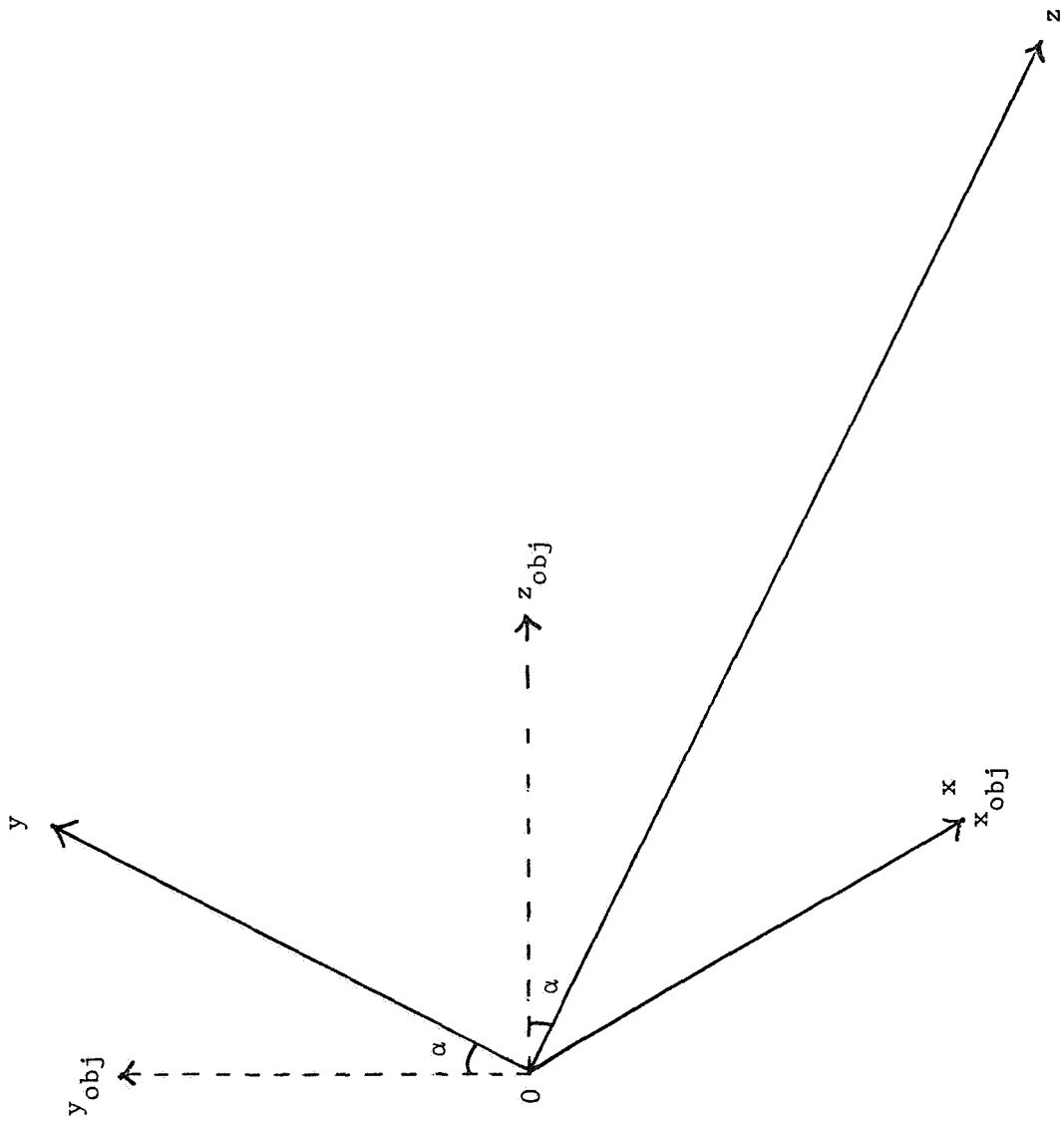
FIGURE 4. OBJECT IN CAMERA CO-ORDINATES

$$x^3_i = x^2_i + dx$$

$$y^3_i = y^2_i + dy$$

$$z^3_i = z^2_i + dz$$

where

$$dx = (IMAGEY - 128) \ HR \ (dz/D)$$

$$dy = (120 - IMAGEX) \ VR \ (dz/D)$$

$$dz = R \ COS \ (VFOV/2 - (VFOV/240) \ IMAGEX)$$

In the matrix form, P^3_i is given by

$$P^3_i = \begin{bmatrix} x^3_i & y^3_i & z^3_i \end{bmatrix} = \begin{bmatrix} x^2_i & y^2_i & z^2_i \end{bmatrix} + \begin{bmatrix} dx & dy & dz \end{bmatrix}$$

Target on projection plane:

Project the vertices of the target onto the projection plane. Let

$$(x^4_i = x^3_i \ (D/z^3_i)$$

$$y^4_i = y^3_i \ (D/z^3_i)$$

Any desired size and shape information as appears in the image can be obtained from the above projection.

165

Orientation estimation

The procedure for estimating orientation and position of an object is given below:

Step 1: Detect all homogeneous regions (blobs) of appropriate size. Each blob is taken as a potential object of interest.

Step 2: From the bottom-left pixel of the potential object, determine its position within the sensor's field of view (back projection).

Step 3: Knowing position, length and height, obtain the possible object orientation from the look-up table.

Step 4: Project the object on to the image plane and compute the desired geometric features of the object from its projection.

Conclusion

Back projection technique is used to determine the ground position of the object which appears as a blob in the input image. Length and height of the blob in the image are used to estimate the corresponding object's orientation with respect to the sensor's line-of-sight. By knowing objects position and orientation, accurate computation of geometric features is possible. This improves the performance of object recognition system.

References

1. Heinz G. Sage, Pat H. McIngvale, and William M. Malcolm, "Preliminary concept investigation of a man-in-the-loop automatic target cuer", Technical Report RG-84-9, U. S. Army Missile Command, Redstone Arsenal, AL 35898, January 1984.

2. Kenneth R. Castleman, "Digital Image Processing", Prentice-Hall, 1979.

3. Rafael C. Gonzalez and Paul Wintz, "Digital Image Processing", Addison-Wesley Publishing Company, Inc., 1977.

Two Dimensional Convolute Integers

for

Machine Vision and Image Recognition

by

Thomas R. Edwards
TREC, Inc   Huntsville, AL.

Machine vision and image recognition require sophisticated image processing prior to the application of Artificial Intelligence. Two Dimensional Convolute Integer Technology is an innovative mathematical approach for addressing machine vision and image recognition. This new technology generates a family of digital operators for addressing optical images and related two dimensional data sets. The operators are regression generated, integer valued, zero phase shifting, convoluting, frequency sensitive, two dimensional low pass, high pass and band pass filters that are mathematically equivalent to surface fitted partial derivatives. These operators are applied non-recursively either as classical convolutions (replacement point values), interstitial point generators (bandwidth broadening or resolution enhancement), or as missing value calculators (compensation for dead array element values).

These operators show frequency sensitive feature selection scale invariant properties. Some of these operators are rotationally invariant allowing orientation independence, while others are rotationally sensitive yielding directional characteristics. As partial derivative operators, new features hitherto unknown are generated which will aid significantly by reducing the time required to match a test image with a catalogued image. The new features may indeed generate new classes of uniqueness.

Such tasks as boundary/edge enhancement and noise or small size pixel disturbance removal can readily be accomplished. For feature selection tight band pass operators are essential. The results from a theoretical test case and an experimental image with jitter clearly display the state of the art advance that Two Dimensional Convolute Integer Technology represents. The ability of a vision system to identify and follow the boundaries of a road without being confused by small size particles at the curb or loosing sight of the curb/edge altogether is significantly enhanced by these operators. The ability of an Artificial Intelligence applications package to quickly identify a feature is considerably improved when a tight matched band pass operator clearly enhances the specific feature of interest.

The Image Processing environment that TREC, Inc has established as an Incubator company in the Johnson Research Center of UAH, will also be discussed.

FAILURE MODES AND EFFECTS ANALYSIS AUTOMATION

Cynthia H. Kamhieh
Dannie E. Cutts
BOEING COMPUTER SERVICES
HUNTSVILLE AI CENTER
HUNTSVILLE, AL

R. Byron Purves
SPACE STATION PROGRAM
BOEING AEROSPACE COMPANY
HUNTSVILLE, ALABAMA

ABSTRACT

A failure modes and effects analysis assistant has been implemented as a knowledge-based system and will be used during design of the Space Station to aid engineers in performing the complex task of tracking failures throughout the entire design effort. The three major directions in which automation was pursued were the clerical components of the FMEA process, the knowledge acquisition aspects of FMEA, and the failure propogation/analysis portions of the FMEA task. The system is accessible to design, safety, and reliability engineers at single-user workstations and, although not designed to replace conventional FMEA, it is expected to decrease by many man-years the time required to perform the analysis.

October 7, 1986

INTRODUCTION

The design of any system requires the compilation of information regarding ways in which components or subsystems (items) of the design may fail and the effect that these failures will have on related items and the system as a whole. This knowledge is essential in evaluating the safety and reliability aspects of the design, weighing alternatives at the component or part level, and capturing information that will be valuable in performing system diagnostics following design implementation. The complexity of the compilation task varies with the complexity of the system and the level of completeness required of the failure analysis. Factors that increase the level of difficulty in performing the analysis include design changes, incomplete or functional specification of components, and the combinatorial complexities that result from multiple, sequential, and partial failures.

Several approaches to the acquisition and analysis of failure/fault information have been proposed and many are in widespread industrial use. These include general techniques such as failure modes and effects analysis (FMEA) and fault tree analysis, as well as specialized methods such as event sequence analysis, failure modes and effects criticality analysis (FMECA), and common cause analysis. Due to the specialized nature of the latter three techniques, only FMEA and fault tree analysis will be discussed here.

Fault tree analysis is a quantitative, deductive failure analysis technique that is conducted top-down from an undesired event through the hardware design. All potential multiple, as well as single, failure points can theoretically be identified, with the resulting fault tree yielding a powerful representation of all events that may lead to a given failure (top-level event). The deductive approach of fault tree analysis is applicable during early or preliminary design phases when bottom-up approaches are unwieldy or inaccurate. A further advantage of fault tree analysis over FMEA is that the concise, quantitative failure information can be represented in a graphic format that is more easily understood than FMEA with its large amount of associated qualitative information.

The most serious disadvantage of fault tree analysis is that it requires the design, safety, and reliability engineers to deductively derive and specify all possible single and multiple failure points, making the process heavily dependent upon the experience and talent of individual engineers. Fault tree analysis is less complete than FMEA, in the sense that qualitative information such as the time to repair an item or the method to detect a given failure is not derived during the analysis.

Failure modes and effects analysis methodology is defined in MIL-STD 1629A and ARP-926A. FMEA is basically a bottom-up, inductive approach to failure analysis, and is used to support safety and reliability engineering as well as design testability and maintainability analysis. Low-level, single point failures are identified, and analysis proceeds upward with the identification of failure effects at each level of the design. With respect to other fault/failure analysis techniques, the major advantage of FMEA is accuracy resulting from consideration of all single point hardware failures. In addition, FMEA results

170

in a large body of qualitative information that is unavailable when more quantitative methods are used, such as corrective action in response to the failure, predictability assessments, and recommended detection methods.

A major disadvantage of FMEA is that it is relatively difficult to apply during early or preliminary design phases. At early points in design activity, it is essential to be capable of analyzing failure information in a top-down manner, with lower level items being treated as functional entities. It has been noted that when using FMEA in a top-down analysis, there is a tendency to identify "impossible" failure modes (1). Furthermore, there have been few instances in which FMEA has been used to assess multiple point failures (2, 3, 4, 5). This drawback can be quite serious, particularly in "human-in-the-loop" systems in which human behavior can be a contributing factor to failure and the end effects are not only upon system functionality but human life as well. A third disadvantage of FMEA is that inconsistencies may easily arise, particularly when performing analyses on complex designs. There are disagreements among engineers in the use of failure mode nomenclature, such as whether a given failure should be classified as a failure to operate or a failure to operate in sequence. A final criticism of FMEA is that it is labor-intensive, particularly regarding the clerical aspects of deriving and recording the FMEA information.

Due to the fact that the advantages of FMEA were perceived to outweigh both its shortcomings and the relative advantages of fault tree analysis, FMEA was chosen as the failure/fault analysis technique to use during preliminary design of the Space Station. Both the qualitative information yielded by FMEA and the completeness of the FMEA technique are seen as essential to a successful design effort. However, it was decided that automation of the FMEA process is both feasible and desirable. Two basic aspects of FMEA that have proved amenable to automation using knowledge-based techniques are the clerical components of the FMEA process, which can be quite labor-intensive, and the failure propagation analysis components of the process, upon which the accuracy and completeness of FMEA depend.

Automation of the FMEA process is essential due to the number of man-years currently required to perform FMEA as primarily a manual procedure (6). At present, forms outlining the required information are distributed to design engineers who have specific knowledge of the items under study. Information required by the format includes component name, failure mode, effect of failure on next higher assembly, effect of failure on crew, time to repair, corrective action, and many other details of item failure. Using CAD systems, functional block diagrams, experience, and judgment, the design engineers complete the FMEA forms. The forms are then compiled into a relational database which is analyzed for accuracy and omissions by safety and reliability engineers. Failure modes are analyzed for effects throughout the entire design, criticalities are assigned, design changes, if any, are recommended, redundancy requirements are addressed, and the process is repeated until the design has been finalized and all failure analysis has been completed and meets safety and reliability standards.

Among the shortcomings of the present method for performing FMEA are the fact that inconsistencies arise due to the large number of engineers performing

171

the analysis, efficiency problems arising from the application of this labor-intensive process to a large, complex system such as the Space Station, and problems of incompleteness when the attempt is made to pinpoint multiple and sequential, as well as single, point failures. These issues were chosen for resolution through the use of a knowledge-based system. The first two problems are primarily of a clerical nature, and an intelligent data entry interface to the engineer performing FMEA has proven to be a feasible solution. The latter point, however, requires a powerful knowledge representation for successful resolution, and has proven to be the greater challenge of the three for the current implementation of the FMEA "Assistant".

APPROACH

The relational database approach that was formerly used to collect and store FMEA information seemed adequate for clerical purposes, but did not allow for much automation of the failure mode analysis portion of the safety/reliability engineering task. For this reason, the first area of study was that of knowledge representation. A knowledge representation that would provide for conciseness and storage efficiency, as well as power in manipulating the available knowledge at any point in the analysis task, was required. Closely related to this concern was the need to build a friendly and robust interface to the FMEA Assistant in order to aid the engineers in entering and manipulating FMEA data. The third area of endeavor for the current phase of the FMEA Assistant project was to provide an array of tools to aid the safety/reliability engineers in analyzing the FMEA information, as well as testing hypotheses about the data at each stage of completeness.

Knowledge Craft (Carnegie Group, Inc.) was chosen as the software package to be used for the prototyping effort because of its frame-based knowledge representation capabilities, the full access that it provides to the common lisp environment, and its well developed user interface features. The FMEA Assistant prototype is currently hosted on a Symbolics 3670.

KNOWLEDGE REPRESENTATION

The representation language CRL (7) was chosen for prototyping the FMEA Assistant. Due to the hierarchical structure of the preliminary FMEA data (system, subsystem, etc.), the decision was made to represent the FMEA system structure, as a whole, by frames containing attributive information in a semantic network. The representation allows structural relationship knowledge to be inherited via IS-A links throughout the system specification. An example of this is:

```
                                              Fire Detection
                                                  \
                          ECLSS                   Waste Management
                        /                      
                      /        Thermal              etc.
                    /
Common Module              Electrical Power
                    \
                      \    Data Management
                        \
                          etc.
```
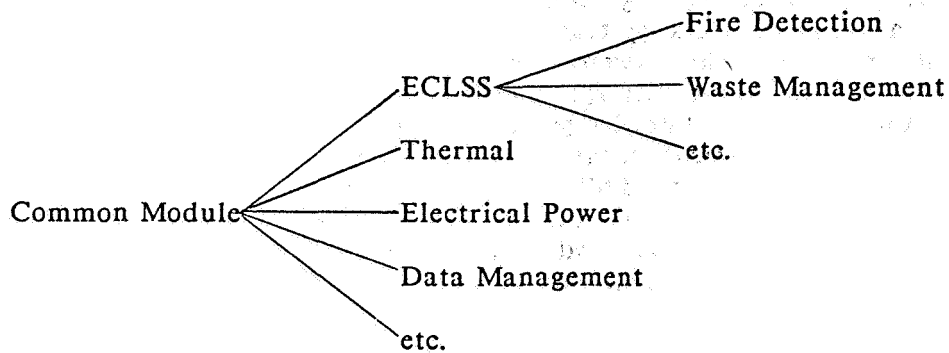
Other, more "causal", links are present throughout the knowledge base, indicating failure mode relationships among items. For instance, a fail-open failure mode in the heat exchanger component of the heat acquisition assembly (thermal control system) will result in a loss of freon in the heat acquisition assembly. This loss of freon in the heat acquisition assembly (fail to operate) will in turn cause decreased heat transfer from the thermal control system. The effects of a specific failure mode are thus traceable throughout the system structure. By defining relationships between the items in the FMEA network, knowledge of causal relationships can be efficiently extracted from or inserted into the knowledge base. When new items in the FMEA network are created by the engineer, he is required to enter the failure effect information corresponding to each failure mode possessed by that item. In addition, attributes may be derived or inherited through both user and system defined inheritance paths via the default inheritance characteristics of the representation or through procedural attachment.

Prototypes representing generic items (e.g. pumps, valves) are defined in such a way as to provide for default descriptions of attributes for that item. This capability has proven to be quite important for failure mode attributes, in particular, since not all possible types of failure mode exist for any given item. Furthermore, the ability to represent item prototypes has greatly eased the task of knowledge acquisition. For instance, it is not necessary for the design engineers to specifically state that each instance of valve possesses all the attributes inherent to valves in general.

KNOWLEDGE ACQUISITION

Item prototypes corresponding to items with generic core attributes were constructed by the knowledge engineer with the help of design engineers and formed the initial base of the FMEA Assistant.

An intelligent editor was designed to aid design engineers in entering FMEA data. Among the issues that had proven to be a problem in the past was the lack of consistency among the design engineers in describing failure modes. Originally, the engineers were requested through Space Station Program FMEA guidelines to use one of five standard failure modes in describing any given

173

failure. These included fail open, fail close/short, fail to operate, fail to operate in sequence, and premature operation (6). In many cases during preliminary design, there appeared to be inconsistencies regarding the category in which to place a particular failure type, as well as instances in which the language used to describe the failure mode of a given item did not clearly correspond to any of the five failure mode categories. A second problematic area appeared to be that of specifying the cause-effect relationships. In situations where events may be a cause in one context and an effect in another, it becomes difficult to specify causality, particularly when dealing with large networks of these relationships. Another major difficulty for the design engineer in performing FMEA appeared to be the enormous clerical load resulting from the paper FMEA format used during early design. This format required that the engineer enter identical information many times for items that differed only slightly in behavior. The time period between completing the form and receiving feedback from design engineers responsible for subsystem coordination and safety/reliability engineers was also quite long.

The intelligent editor was designed in an attempt to solve these problems and lessen the burden being placed on the design engineers by manual methodology. When adding an item to the FMEA network, the engineer is required by the editor to graphically place the item within the structural hierarchy, aiding him in specifying cause-effect relationships through spatial cues. Following placement of the new item within the network, a window is displayed to the engineer that generally follows the format of the traditional FMEA form. Information that may be derived through inheritance from higher level items in the FMEA network or from the item prototypes, through procedural attachment, or through the graphical placement of the item in the FMEA network, is already placed in the form and need not be added by the engineer. The engineer is required to complete all remaining information. Automatic range, cardinality, and error checking is performed by the editor where it is appropriate. Pop-up displays of help information, available options, and default values are available to the engineers at any point in the data entry process. Following completion of the FMEA form, the information is incorporated into the frame structure of the new item and forms a new node in the FMEA network.

FMEA NETWORK MANIPULATION

At any point, the safety, reliability, and design engineers may use the FMEA Assistant to examine failure mode propagation and to perform "what-if" tests on the available FMEA knowledge. A variety of tools are provided to make this process as simple and clear as possible.

Portions of the FMEA network may be selected and displayed through mouse control. Individual items in the network may be selected for study and lists of available option for that item are displayed in pop-up menus. For instance, if failure mode propagation behavior is under investigation for a particular item, a list of all of the item's failure modes is displayed and a given failure mode may be induced simply by clicking on that menu item. The failure mode is then graphically propagated through the network. At this stage of the project,

single failure point analysis has been implemented. Multiple point failure will be addressed in the near future.

CONCLUSION

Failure modes and effects analysis automation is both an essential and a feasible endeavor. In the design of a system as large as the Space Station it would be virtually impossible, and certainly very expensive, to accomplish an adequate analysis of all possible failure modes using a manual methodology. Yet in view of the fact that human life on board the station, as well as vital research and development, will be dependent upon an accurate understanding of possible failure points and corrective actions, it is imperative that such an analysis be performed prior to placing the station in use.

An important additional feature of the FMEA Assistant is that it comprises a continual, "living", representation of the Space Station FMEA process. As design progresses, the degree of knowledge present within the FMEA system grows and its performance increases accordingly. At termination of the design activity, a full record of failure mode tracking is available for flexible analysis and use. Following placement of the Space Station in orbit, such information could form the basis for intelligent diagnostic and maintenance systems.

The FMEA Assistant represents a major step toward more efficient and accurate methods by which to conduct FMEA for the Space Station. Methods by which to study multiple, sequential, and partial failures remain to be incorporated into the system. Further refinement of the prototyping technique to describe generic items must await later design phases in which low level items, now treated as functions, will be more fully specified.

REFERENCES

1. Goddard, P. L. and Davis, R. Automated FMEA Techniques. Hughes Aircraft Company, Technical Report No. AD-A154161, December, 1984.

2. Pearson, K. L. and Babin, R. S. Integrated Reliability and Safety Analysis of the DC-10 All Weather Landing System. Proceedings, Annual Reliability and Maintainability Symposium, 1973.

3. Bjoro, E. G. Safety Analysis of an Advanced Energy System Facility. Proceedings, Annual Reliability and Maintainability Symposium, 1982.

4. Lance, J. R., Mutone, G. A. and Bjoro, E. G. Safety Analysis of an Advanced Energy System Facility. Proceedings, Annual Reliability and Maintainability Symposium, 1980.

5. Hedin, F., Le Coquiec, A., LeFloch, C., Llory, M. and Villemeur, A. The Failure Combination Method. Proceedings, Annual Reliability and Maintainability Symposium, 1981.

6. Boeing Aerospace Company, Preliminary Failure Mode and Effects Analysis Report - WP 01. Boeing Aerospace Company, Space Station Program, Document No. D483-50045-1, December, 1985.

7. Knowledge Craft Reference Manual, Version 3.0. Carnegie Group, Inc., Pittsburgh, PA, October, 1985.

INTELLIGENT TEST INTEGRATION SYSTEM

J. Sztipanovits, S. Padalkar,
J. Rodriguez-Moscoso, K. Kawamura
Vanderbilt University
Nashville, Tennessee 37235

B. Purves, R. Williams, H. Biglari
Boeing Aerospace Company
P.O. Box 1470
Huntsville, Alabama 35807

ABSTRACT

This paper describes a new test technology which has been developed for space system integration. The ultimate purpose of the system is to support the automatic generation of test systems in real-time, distributed computing environments. The Intelligent Test Integration System (ITIS) is a knowledge-based layer above the traditional test system components which can generate complex test configurations from the specification of test scenarios.

INTRODUCTION

Testing is a key factor in the quality of products and is very important in the design and implementation of high stake systems, which are typical in the space industry. Testing of complex systems is a difficult and costly process which requires sophisticated test technology. A well-known rule of thumb is that the cost distribution of software development is approximately 40% design, 20% coding and 40% test [1]. In the case of complex, hybrid systems that consist of closely coupled hardware and software components, the cost of testing can be even higher, or even worse, unpredictable. These facts necessitate the intensive improvement of test methodologies as one of the most important elements of the development technology.

Test technology reflects the level and nature of the System Under Test (SUT). Different test equipments and methods are applied for testing VLSI circuits and mechanical systems. A very complex, therefore expensive, test methodology has to be used at system integration. System integration is the phase at which the interaction of heterogeneous subsystems, which are built by using strongly different technologies, must be tested. Test systems for system integration often require large, distributed computer configurations, and the test system includes complicated, distributed, real-time software.

This paper summarizes the results of our research on the development of the Intelligent Test Integration System (ITIS). The primary purpose of ITIS is to dramatically decrease the cost of building test systems for system integration. ITIS is a knowledge-based layer above the low-level test system components which can generate complex test configurations from the symbolic description of the test scenario. Beyond this basic capability, the test process itself is expanded by new facilities, such as intelligent result analysis and dynamic reconfiguration. We begin our discussion with summarizing the main requirements for ITIS. Next, the

system configuration and the basic software components are presented, which is followed by the summary of the current state of the research.

STATEMENT OF THE PROBLEM

The most important properties of system integration from the aspect of testing are: **heterogeneity, complexity,** and **changing environment.**

1. The subsystems to be integrated often are different in their physical nature and are developed by different organizations. This diversity requires flexible test environments, that include computing systems, complex instrumentation, and well-developed interface technology.

2. Space systems are increasingly complex, which implies that the number of subsystems is large and they interact in an unobvious way. Complexity in testing means that a large number of physical parameters have to be monitored, complicated constraints have to be continuously checked, and many test cases have to be defined.

3. Integration of any large-scale system requires precise knowledge of the state of subsystem development. The subsystems are usually not completed at the same time, therefore, the test engineer has to evaluate the criticality of the subsystem, and, if necessary, simulate it in order to prevent the delay of the integration effort.

If the test system developers use traditional software tools for the test development, the process will be extremely expensive because:

1. Design and implementation of distributed, real-time software systems demand highly trained programmers who can master the complex computer configuration.

2. As a consequence of the heterogeneous SUT's, the programmers have to be trained (and retrained) to use the various interface techniques.

3. An extensive documentation system has to be developed and maintained in order to keep track with the changes in the specification and state of the subsystems.

4. Since test systems are complex, their integration is a lengthy and demanding process. Frequently, test integration requires highly qualified test engineers and software engineers instead of test technicians.

AI techniques can assist in solving these problems by providing a "knowledge level" which configures and controls the operation of the low level system components. The ultimate goal of our research was to develop a prototype system, ITIS, which extensively supports the test system integration by using AI techniques. The system has been implemented in the framework of Multigraph Architecture and by using the corresponding distributed programming model, which are described in a companion paper [2].

TEST SYSTEM CONFIGURATION

ITIS has been designed for real-time, distributed computing environments. The prototype system has been installed on the computer network of the Space Station Laboratory at Boeing Aerospace Company in Huntsville, Alabama. (See Figure 1.)

The configuration includes a VAX/750, and two MVAX-II computers, connected by an ETHERNET link. The VAX systems run under the VMS/DECNET operating system. An HP 9836 computer and a Real-Time Data Acquisition System serve as an intelligent Remote Terminal Unit (RTU) in the test system. The RTU is connected to the VAX/750 and to one of the MVAX-II's by a RS232 interface. The two other main components of the configuration are an Apollo workstation and the AYDIN graphics system.

Some of the functional components of the prototype test integration system have been dedicated to specific computers. The real-time database and the low-level data processing programs are allocated on the VAX/750. The system is supported by a SCADA system (Supervisory Control and Data Acquisition), which has a graphic input/output device (AYDIN). The SUT, which is a controller for a physical subsystem of the Space Station, is allocated on the Apollo workstation. (The controllers are implemented by the Application Generator system developed by the Boeing Aerospace Company.) The simulator of the subsystem runs on one of the MVAX-II's. The RS232 link between the MVAX-II and the RTU makes it possible for the measurement data to be collected either from the simulator or from the physical system.

STRUCTURE OF THE INTELLIGENT TEST INTEGRATION SYSTEM

On the top level, ITIS is composed of Autonomous Communicating Objects (ACO), which are allocated dynamically on different VMS nodes. The object types used in ITIS are: (1) Rule Network Object (RNO), (2) Procedural Network Object (PNO), and (3) Real-time Database Object (RDO).

RNO's are rule-based systems that perform forward-chaining triggered by facts that are sent to them as messages. PNO's represent procedural networks and their interfaces, and can be used for implementing complex signal processing systems, or simulation systems. RDO is an "object shell" around a real-time database, which provides a unified interface to the database itself. A detailed description of the object types is given in [2].

Obviously, the actual components and structure of the test system are determined by the SUT and by the particular test scenario. As it was discussed before, the main problem of test integration is that due to the heterogeneous subsystem structure, large number of test cases, and different states of the subsystem development, the number of test system alternatives may be extremely high. Separated development of the different alternatives, their continuous upgrading and maintenance are very expensive and sources of inconsistencies.

The essence of our approach can be summarized as follows:

179
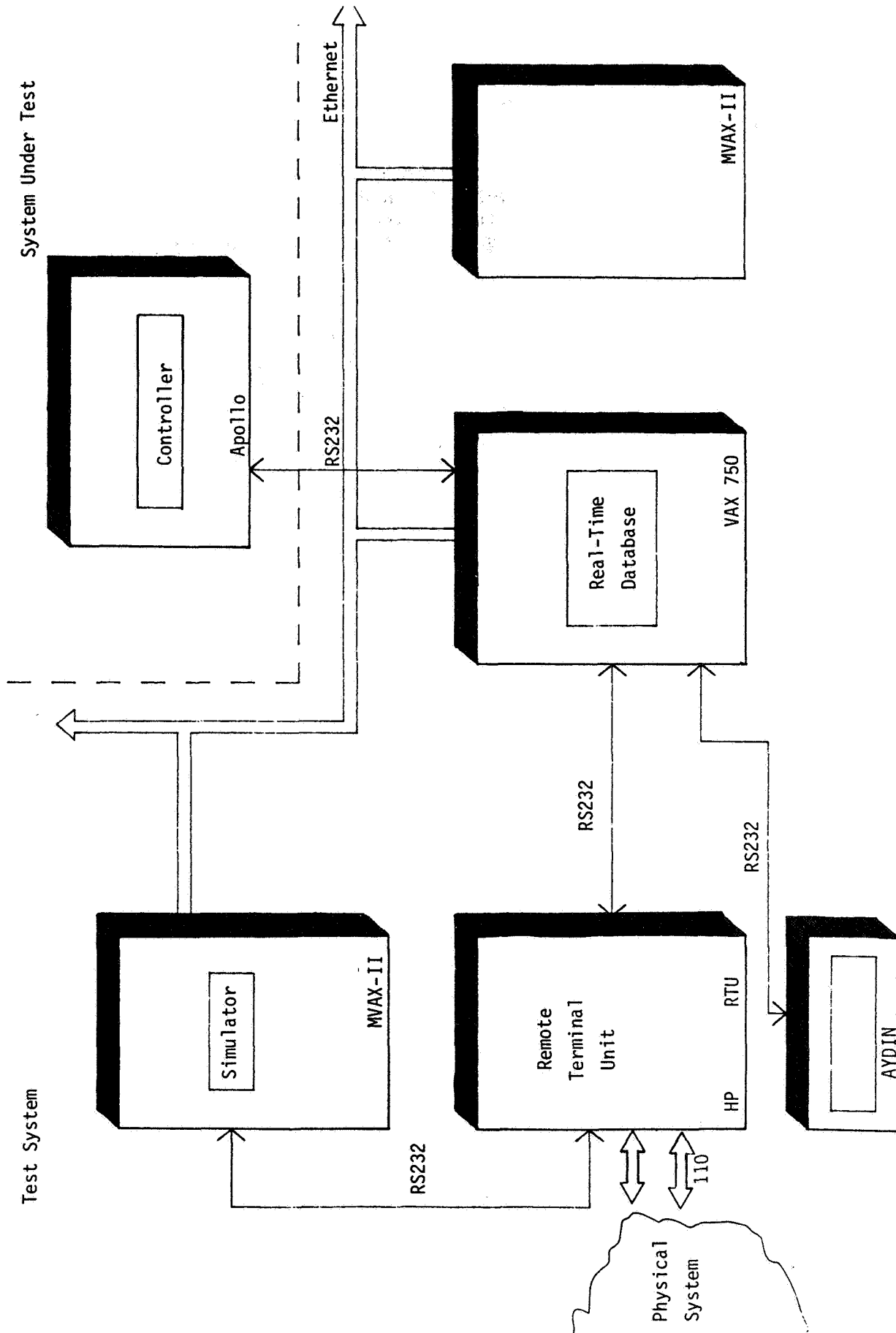
FIGURE 1. System Configuration

1. A generic test system structure has been defined whose components are communicating objects.

2. The objects are associated with hierarchically structured knowledge bases. The knowledge bases describe in declarative form the various test system components, the corresponding selection rules and the properties of the system configuration (e.g., processor nodes, interface specifications, etc.)

3. A particular version of the test system is generated by a top-down building process: first the scenario description is processed and the top-level test system components are selected. These components will receive the relevant information derived from the scenario specification and interpret it. The result of the interpretation is the selection and allocation of lower level system components, and the same process is applied recursively. The test system, which has been built by the generation process, has its own operator interface, which is used for experiment control.

4. Taking advantage of the Multigraph Execution Environment [2], the test system can be reconfigured dynamically, as a response to the detection of significant events in the test process.

The top level functional structure of ITIS can be seen in Figure 2.

The Operator Interface Object is a RNO, which communicates with the operator through a nested menu interface. The menus present alternatives for characterizing the test scenario, such as goals of the test, state of the various subsystems, pass/fail criteria, etc. The answers of the operator are formed, and sent to the Test System Builder (TSB) object.

TSB is also a rule network object which is responsible for generating the test system configuration. On this level, the test system generation means: (1) the selection of the top level objects, (2) decision about their allocation, according to configurational constraints and resource allocation rules, and (3) sending specification parameters to the top level objects for further interpretation.

The Test Signal Generator (TSG), System Under Test Interface (SUTI) and Test Analyzer (TA) objects are usually procedural networks. The specification parameters, which are sent to them after allocation, are interpreted in the context of their local knowledge base, and the appropriate version of the procedural network is built.

The TSG object is typically a procedural network which has a dynamic control interface. The operator can select test cases through this control interface. The structure of the SUTI object depends on the test scenario. Two of the typical alternatives are: (1) SUT is an external hardware, therefore only the appropriate input/output interfaces have to be built, (2) SUT is an external hardware controller, but has to be simulated for the system to be controlled. In the latter case, not only the input/output interfaces, but also the simulator system have to be generated. The TA object in the prototype system has two levels: a low-level procedural network checks the pass/fail criteria, and a connected rule network object
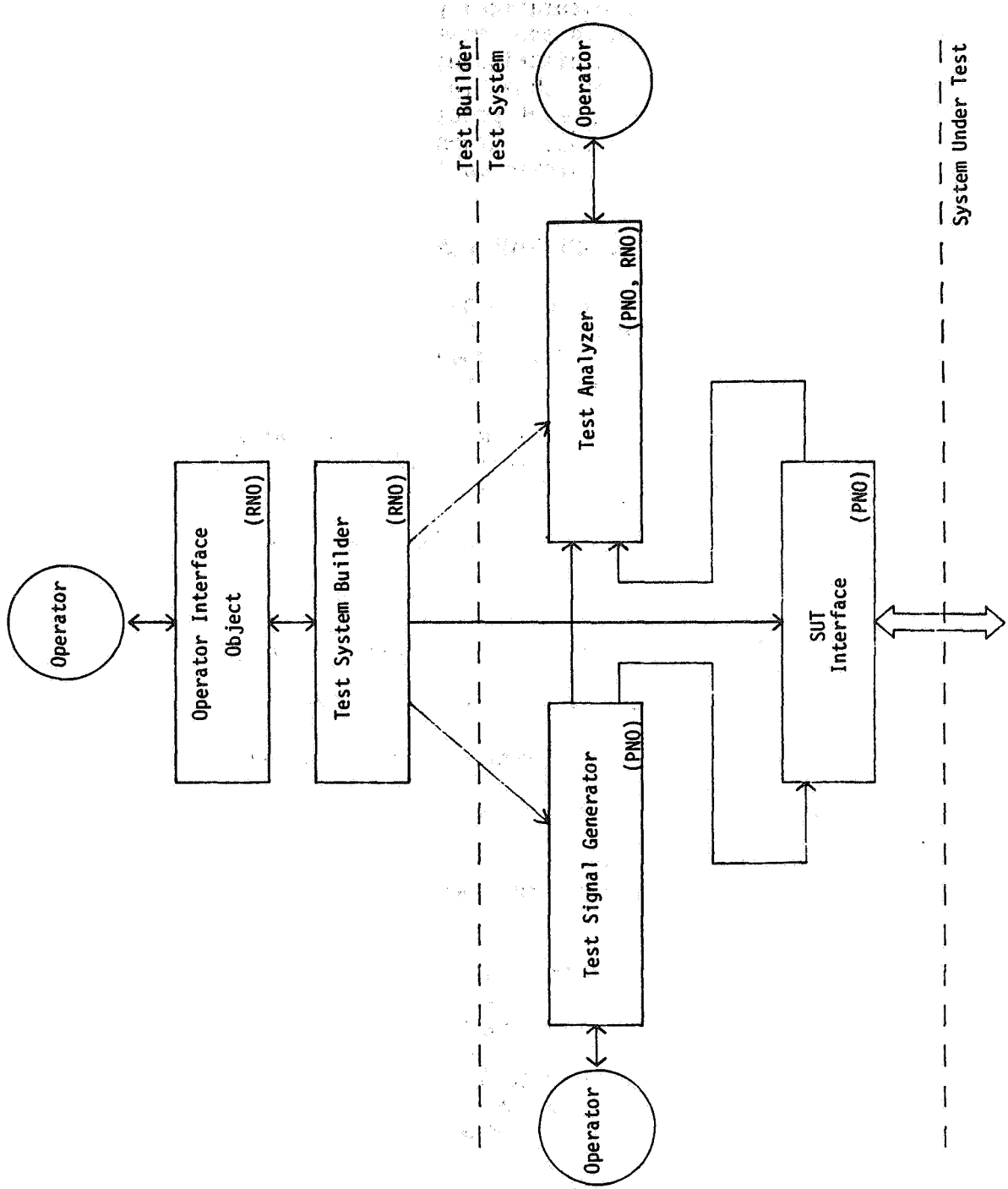
FIGURE 2. Generic Structure of ITIS.

interprets and qualifies the results for the various test cases.

The most important advantage of the ITIS is that the actual version of the test system is generated automatically from the knowledge bases. The user does not have to design and implement the complex control flow and synchronization scheme of the distributed test program. The knowledge bases are strictly declarative: they include selection rules, and the specification of the logic structure of procedural networks. In order to support test system visibility [3], a graphic monitor has been developed which can present the actual test system configuration.

TEST INTEGRATION FOR THE SPACE STATION COMMON MODULE THERMAL SYSTEM

For testing and demonstrating the capabilities of ITIS, a prototype system has been developed. The SUT for the prototype system is a hierarchical controller for the space station common module thermal system [4]. The block diagram of the system configuration can be seen in Figure 3.

1. The Test System Builder (TSB) prompts the operator with questions, which describe the required test scenario. The basic scenario alternatives in the the prototype system are:

    -   The thermal system is in simulated or hardware form.

    -   The thermal simulation model is series or parallel.

    -   The thermal local controllers are in simulated or hardware form.

    -   Test signals are generated by software or hardware.

    -   The analyzer for the test system has one or two levels.

    -   The goal of the test is stability or accuracy analysis.

    Based on the answers, TSB selects the main test system components, determines their specifications, and decides the basic system allocation.

2. If the operator requires the thermal system to be simulated, then the top level structure of the test system generated corresponds to Figure 3. The reason RDO is used for coupling the simulator with the controller is that we wanted to monitor the state of the thermal system by using the SCADA's graphic output facility. This approach ensures that the operator interface of the test system configuration will not change if the physical model of the thermal system is used.

3. The operator interface of the generated test system makes it possible for the operator to select a particular load profile, which will be sent to the simulator by the TSG object. The TA object checks whether the temperature values are in the predefined range, and sends a message to the operator if a particular test fails.

As we have mentioned before, although the simulator, the controller

183
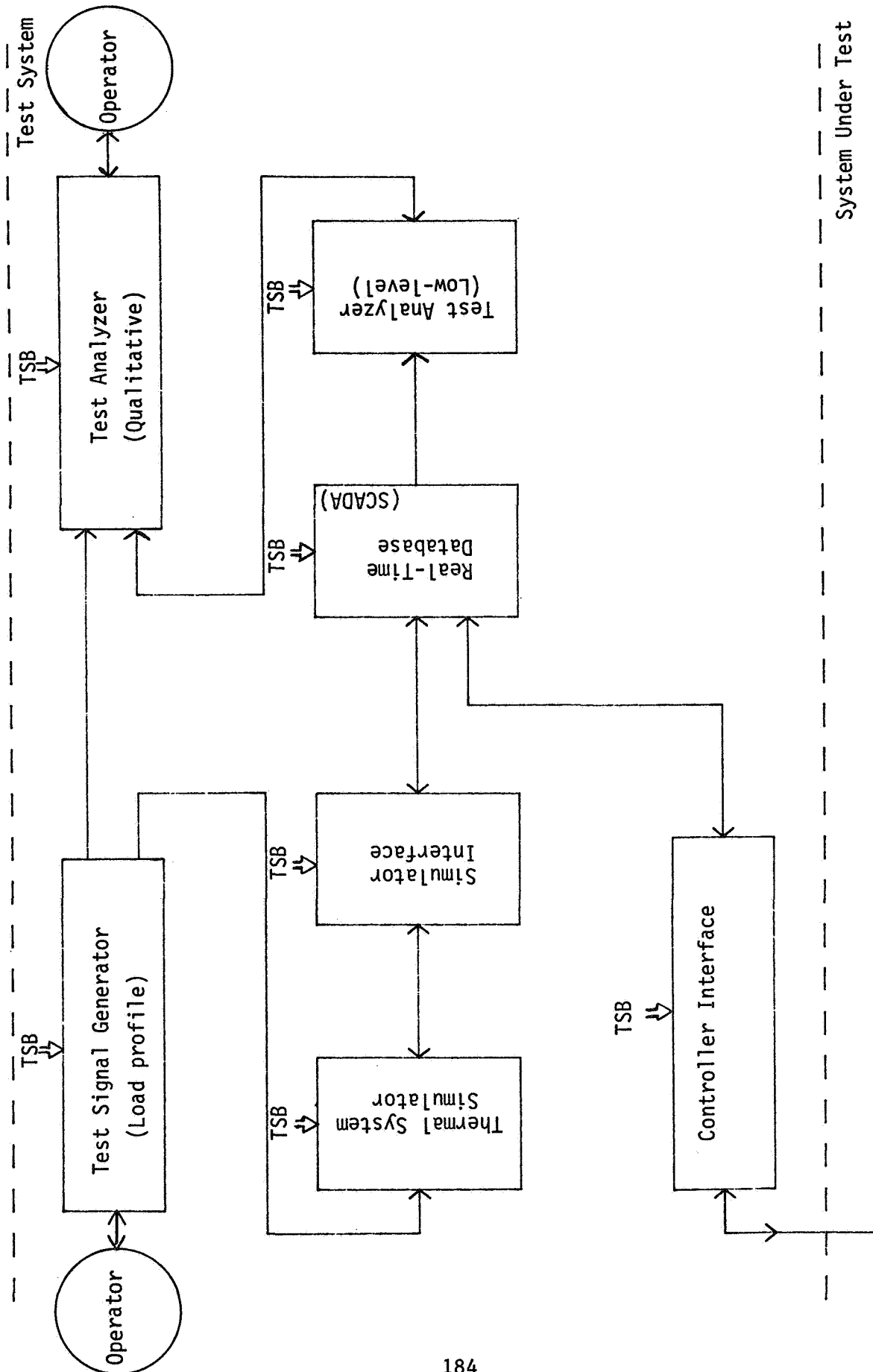
FIGURE 3. Test System of the Thermal System Controller

and the RDO are usually allocated on different VAX nodes, the actual allocation is transparent to the operator. The necessary intertask synchronization and communication functions are generated automatically.

SUMMARY

The prototype of ITIS has been implemented for the Space Station Laboratory of the Boeing Aerospace Company, in Huntsville, Alabama. ITIS is a high-level AI control layer above a distributed test system configuration, which provides the following services:

1. The appropriate version of the test system is generated automatically by using the specification of the test scenario.

2. ITIS supports the dynamic allocation of the test system components and is able to reconfigurate the system for selected run-time events.

3. The system includes hierarchically structured knowledge bases, which are defined in problem-specific representation languages.

The prototype system has proven that the approach dramatically improves the productivity of test technology for system integration. The next step of the research will be to provide graphic tools for building and modifying the knowledge bases.

REFERENCES

[1] G. Pomberger, Software Engineering and Modula-2, Prentice Hall International, 1986.

[2] J. Sztipanovits et al., "Programming Model for Distributed Intelligent Systems," Proc. of the Conference on Artificial Intelligence for Space Applications, Huntsville, AL, 1986, this issue.

[3] F.G. Danner, "System Test Visibility - Or Why Can't You Test Your Electronics?," Proc. of the International Test Conference, pp. 635-639, 1983.

[4] F. Lewis, J. Cheng, K. Davey, G. Vachtsvanos, B. Purves, "On The Hierarchical Control of the Space Station Common Module Thermal System," Proc. IEEE CDC, Athens, 1986.

ON RECOGNIZING IGNORANCE

Richard J. Greene

General Research Corporation

How can an expert system reason about its own ability to deal with a particular problem ? Ideally, an expert system ought to rapidly recognize that a particular problem is beyond its abilities and defer to another, perhaps human, expert . This capability is extremely important in domains where expert systems may control life-critical processes such as air traffic control, medicine, strategic defense, and manned space exploration. In short, an expert system should behave as a human expert and call for help when the situation requires it. Even if one assumes an expert system can sense its limitations, can it do this quickly enough to matter ? Paradoxically, human experts often do recognize rapidly when a problem is beyond their ken and do not try to solve it. How this occurs without exhaustive searching is still an open question.

This paper shall :

* Survey the methods used by knowledge engineers to infuse an expert system with knowledge of its own limitations

* Employ computability theory to analyze the general problem of meta-knowledge and to give insight into the efficacy of specific solutions.

2.0   Notions About Ignorance

The calculus of probabilities quantifies ignorance as well as certainty. For example, if one asserts " a five appears" and then throws a die, we can say that the assertion is ignorant five-sixths of the time. "A five

doesn't appear" is clearly a less ignorant statement. An assertion is free of ignorance if and only if one can assign a probability of 1.0 or 0.0 to it.

The situation is much more complicated when we do not know the probabilities. In this case , the only method of validation is empirical. Should the cardinality of the set of possible outcomes be unknown, we have no way of assessing our limitations and quantifying our ignorance.

3.0 Impact of Ignorance On Expert Systems

Our initial notions about ignorance suggest that any expert system containing heuristic information cannot be sound, much less complete. Of course, this idea agrees with common sense : human experts are neither sound nor complete within their domains of expertise. However, knowing this, human experts are often able to detect unsound reasoning or ,better yet, recognize without trying to solve the problem that the prospects for a sound solution are dim. For example, a mathematcian knows that if the determinant of a matrix equals zero , then the matrix has no inverse.

4.0 Technique For Recognizing Ignorance

One can recognize ignorance *a priori*   by employing metaknowledge[1] . For example, if you were asked to name the capitals of all fifty staes, you might quickly answer "I don't know them all" without knowing of which ones you are ignorant or even trying to name any. In short, you seem to have second order knowledge about what you know and what you do not know. Expert systems may employ metaknowledge to bring "promising "

subsets of knowledge to bear upon a problem rather than "try everything". Ideally, an expert system ought to deal with two distinct problems :

* The competence problem : can a given problem solver solve a given problem?

* the construction problem : what is the solution to the given problem?

An expert system then employs a "metaknowledge" base for reasoning about the competence problem and a knowledge base for reasoning about the problem's solution. If one factors expert problem solving this way , all seems well : we simply solve the competence problem separately from the construction problem. In this way the expert system "knows" its limitations. The crucial question here is itself a competence question : does such a general "competency test" exist ?

5.0   The Possibility Of A General "Ignorance Recognizer"

We cannot, in general, construct an algorithm which takes a knowlede base , inference procedure , a problem description as input and produces a yes/no competence decision. The ability to do this implies a solution to the halting problem for Turing machines. To see this, simply rephrase the problem to read a Turing machine and input string as input and "halt" for competence decision.

We can approach the competence problem from another direction and derive similar results. If the Church-Turing Thesis is correct, then any finite description is describable in first order predicate calculus [2]. Thus any expert system can be recast in first order predicate calculus and viewed as a theorem prover. According to Goedel's Incompleteness Theorem, we know there are true but unprovable

189

theorems[3]. If a general "ignorance recognizer" existed, we could use it to recognize our inability to prove these theorems before a theorem prover tries unsuccessfully to prove it. Yet, this contradicts the assumption of "unprovable" since the "ignorance recognizer" is also a theorem prover. Thus the existence of a general "ignorance recognizer" is informally shown to be impossible. Yet, are specific "ignorance recognizers" possible in principle?

6. Possibility Of A Specific "Ignorance Recognizer"

Given that a general ignorance recognizer is impossible, might we construct a specific ignorance recognizer on a case-by-case basis? The answer to this question involves two types of analysis :

* the recognition of domain specific ignorance in principle

* the recognition of domain specific ignorance when heuristic information is present

The recognition of domain specific ignorance is , in principle, possible because the Halting Problem results merely refute the possibility of a general ignorance recognizer. No assertions are made regarding a specific Turing machine and a specific class of input strings. Thus , in principle, given a specific expert system and a specific class of input instances , it is possible for the system to recognize its limitations. The theory of computability does not tell us how to do this, unfortunately .

On the other hand, can an expert system recognize its limits when heuristic inference is present? In this case we can assert that it cannot.

We cannot even assert with certainty that a given inference is sound . A conclusion based on a series of potentially unsound inferences must itself be suspect. In short, we reach a conclusion (i.e. assert a

190

proposition is true) based on a series of potentially unsound inferences.

Thus, unsoundness is inherent in expert systems. We cannot guarantee an expert system will infer only true(valid) assertions.

Yet, we can nevertheless obtain a greater degree of control over this situation. One technique is to view the input data set as a vector and apply pattern recognition techniques to rapidly classify it as "probably can do" or "probably cannot do" and proceed accordingly. A "minimum distance" classifier with a suitable distance metric serves as the basis of this approach. Basically, the classifier views input data instance as a point in Euclidean n-space and uses the distance to known solved problems (prototypes) as a guide in estimating the expert system's ability to solve a specific problem. If an input instance is sufficiently "far" from its closest prototype, then the expert system may recognize this as ignorance.

Although not a panacea , this approach may offer a real-time remedy to the ignorance recogition problem. I am currently investigating these topics and hope to report my findings .

References

1. Davis,Randall and Douglas Lenat *Knowledge-Based Systems in Artificial Intelligence* , Mcgraw Hill,Inc ,New York 1982

2. Jackson,Philip *Introduction To Artificial Intelligence* , Dover Publications,Inc ,New York 1985

3. Hofstatdler, Douglas *GODEL,ESCHER, BACH* Random House,Inc,New York 1979

Automated Practical Reasoning Systems

Michael Lewis

By "practical reasoning system" I mean a system, be it human or mechanical, which makes rational decisions about what to do or what acts to perform in the face of well-specified circumstances. The final decision made is of the form "I ought to do A" where the "I" may be a human or computer and the "A" is the act to be performed. I shall call this sort of ought a "genuine ought", since it will actually entail the system performing the act, or, as it were, A'ing. Not all oughts, though, entail the performance of some act. Some oughts enter into practical deliberation, although their act-objects are not performed. E.g., a system might infer at one moment that it ought to flip a certain switch, but then other information might ensue which defeats or overrules the decision to flip the switch. But then again even later information might ensue such that the system infers that it ought afterall to flip the switch. I shall call these kinds of oughts (the kinds which can be overruled given further information) "defeasible oughts". With enough information presented over a reaonable amount of time, the final defeasible ought becomes the genuine ought and the sytem acts accordingly.

Below I sketch a system which infers genuine oughts based on the informal account of decision-making and action presented above. The system was implemented in PROLOG at the Advanced

Computational Methods Center at the University of Georgia and is
available for demonstation (Cf Lewis [2]).

Consider the following first approximation of rules for
practical decision-making (adapted from von Wright [3]):

1 System S ought to do A if and only if the doing of A will
have (or result in) the property P and it ought to be that
P.

2 System S ought not do A if and only if the doing of A will
have (or result in) the property P and it ought to be that
not-P.

Note that we have the word "ought" applying to the performance of
acts in the left side of the biconditional, and have it applying
to properties or results of acts in the right side of the bicon-
ditional. Thus, we must distinguish a third kind of ought, viz.,
the "normative ought". The distinction between normative oughts,
on the one hand, and genuine and defeasible oughts on the other,
is familiar in philosophical ethical theory. It is normative
that one keep promises. It is defeasible that I keep this parti-
cular promise. And it is genuine that I keep this particular
promise if all the relevant information is in and the act of
keeping the promise is not overruled.

The problem with our first approximation is that the same
act may very well ought to be performed and ought not be
performed. Suppose that the flipping of a switch results in
illumination and also results in easy detection by a second

party. Further suppose that illumination is desirable (or normative) and that easy detection is undesirable (or non-detection is normative). It follows by the above rules, then, that S ought to flip the switch and ought not flip the switch. Of course, the system cannot do both. Our first approximation must be refined so that it can make some sort of rational decision when an ought and an ought-not conflict. I.e., it needs some way to decisively infer a genuine ought over some reasonable period of time.

A simple way to adjudicate between conflicting oughts is to determine which result is more desirable, e.g., illumination or non-detection. Thus, as a second approximation, consider the following rules of practical decision-making (adapted from Castaneda [1]:

> 1' System S genuinely ought to do A if and only if there is a natural number $\underline{n}$ and some normative system N such that S $ought_{N(n)}$ do A and it is not the case that there is some normative system N' and natural number $\underline{m}$ such that S $ought\text{-}not_{N'(m)}$ do A and $\underline{m}$ is greater than or equal to $\underline{n}$.

> 2' Otherwise, S genuinely ought not do A.

This approximation requires that results of actions (or normative oughts) be numerically weighted. The numerical weights assigned to results of actions are metaphorical expressions of the import or degree of desirability attributed to those results. In the

example of flipping the switch, then, the system simply needs to know what is more desirable, illumination or non-detection. If the desirability of illumination has a weight of 5, and the desirability of non-detection has a weight of 3, then the following two ought-statements are issued:

S ought$_{\text{illumination}(5)}$ flip the switch.

S ought-not$_{\text{non-detection}(3)}$ flip the switch.

From 1', since 3 is not greater than or equal to 5, S genuinely ought to flip the switch, and finds itself actually flipping the switch.

The problems with this second approximation are two-fold: First, it is still possible that the same action both ought to be performed and ought not be performed. One can easily imagine cases in which a system assigns a weight of 5 both to illumination and non-detection. In such cases, the system is stalemated. The second problem, which is related to the first one, is quite simply that practical decisions about what to do are often not this simple and straightforward. When one is in a quandary, and one is seriously reflective and conscientious, all kinds of considerations come into play. To illustrate this point, consider a story with which in some way I'm sure we all can identify:

> Tom, let us say, is a tenth-grader who has promised Sally that he will come to see her. His mother forbids him to see her. Tom, then, is in a quandary. Should he defy his

mother and see Sally anyway?   Or should he not   visit Sally?
He and Sally really have   a   thing for each other -- his
"heart" says go see her.   Further, he knows that Sally will
be quite hurt if he doesn't see her.   If he sees Sally,
though, he will have to lie to his mother.   If he doesn't
see her and breaks it off, though, he might be forfeiting a
chance at a lifetime of happiness.   Etc, etc...

Of course, this story is a bit melodramatic.   However, the plot
or decision procedure is equally applicable to investment in
stocks and bonds or the purchase of a new car or flipping a
switch. The point of the story is that several normative oughts
are at play, viz., keeping promises, obeying parents, following
one's feelings, hurting another human being, telling lies or
fabrications, pursuing happiness, etc.   Given this kind of
information, how might a system decide what is it's genuine
ought?   Somehow, it contemplates each horn of the dilemma, and
over some satisfactory period of time, one horn eventually out-
weighs the other and the system acts accordingly.   The contem-
plation of a reflective and conscientious system involves a care-
ful weighting of each normative ought, and culminates in a sort
of global collective weight of seeing Sally and a global weight
of not seeing Sally. The ought with the greater weight is the
genuine ought.   This sort of practical deliberation is captured
in the following third approximation for rules of practical
decision-making:

1'' S genuinely ought to do A if and only if there is some natural number k of normative systems N_1, N_2,...,N_k collectively having weight $\underline{n}$ (written $N(k,n)$) which issue S ought$_{N(k,n)}$ do A and it is not the case that there is some number 1 of normative systems N'_1,N'_2,...,N'_1 collectively having weight $\underline{m}$ which issue S ought-not$_{N'(1,m)}$ do A and $\underline{m}$ is greater than or equal to $\underline{n}$.

2'' Otherwise, S genuinely ought not do A.

In our story, the rational course for Tom to take is to examine the results of his seeing Sally or not seeing Sally and to determine the import of those results. E.g., how important is it to him to keep promises? How important is it to obey parents? To not hurt another human being? Each normative ought is weighted and contributes to the collective weight of the issuance of the ought and the ought-not. This sort of deliberation is spelled out in 1'' and 2''.

Problems remain, however. (i) The notion of "collective weight" needs to be clearly spelled out. To date the system considers two parameters when computing collective weight: the average of the particular weights and the number of weights involved. E.g., if three normative oughts of weight 5 issue "S ought to A" and two normative oughts of weight 5 issue "S ought-not A", then the system infers that it genuinely ought to do A, since there are three strikes for doing A and two for not doing A. (ii) Chained results, i.e. results of results ... of

results of actions must enter into the computation of collective weight. E.g., flipping the switch results in illumination, illumination results in easy guage-reading, easy guage-reading results in efficient decision of action, while illumination results in easy-detection and efficient decision of action results in non-detection. The consideration of chained results is tantamount to "good foresight". To date the system can handle the consideration of chained results. (iii) After all the information is in, given a suitable amount of time, and the deliberation has taken place, it is nonetheless possible that the system is stalemated. In such cases, the system must randomly choose one horn of the dilemma and act accordingly. This calls for one final approximation of our rules for practical decision-making:

1''' If there is some number k of normative systems N_1, $N_2, \ldots, N_k$ collectively having weight $\underline{n}$ which issue S ought$_{N(k,n)}$ do A and some number l of normative systems $N'_1, N'_2, \ldots, N'_l$ collectively having weight $\underline{m}$ which issue S ought-not$_{N'(l,m)}$ do A, then

(i) if $\underline{m}$ is greater than $\underline{n}$, S genuinely ought to do A;

(ii) if $\underline{m}$ is equal to $\underline{n}$, S randomly chooses between doing A and not doing A;

(iii) otherwise, S genuinely ought not do A.

The system running so far instantiates 1'''. The factual knowledge base upon which the system deliberates consists of just two kinds of clauses:

has(Act, Property).

normative(Property, Weight).

From a collection of these kinds of facts, the system infers both the defeasible oughts and the genuine ought. The author is not perfectly satisfied yet with the system's computation of collective weight. This is considered one of the problems for further reflection and research. It is hoped that this line of research will facilitate the transference of human decision-making capabilities to machines.

References

[1] Castaneda, Hector-Neri: The Structure of Morality, Charles C. Thoman: Springfield, Illinois; 1974.

[2] Lewis, Michael: "The Automation of a Practical Reasoning System Based on Concepts in Deontic Logic", ACMC Research Report 01-0014; Advanced Computational Methods Center, University of Georgia; 1986.

[3] von Wright, Georg Henrik: "On the Logic of Norms and Actions", New Studies in Deontic Logic (editor: Risto Hilpinin); Reidel: Dordrecht, Holland; 1981; 3-35.

Michael Lewis
Department of Philosophy
University Center at Binghamton
State University of New York
Binghamton, New York 13901
U.S.A.

DAISY-DAMP
A Distributed AI System
for the
Dynamic Allocation and Management of Power

Steven B. Hall, Ph.D
Peter C. Ohler
Lockheed Missiles and Space Company
O/59-50 B/580
Sunnyvale, CA 94089-3504

Abstract

One of the critical parameters that must be addressed when designing a loosely-coupled distributed AI system has to do with the degree to which authority is centralized or decentralized. The decision to implement the DAMP system as a network of cooperating agents mandated that we address this issue. The DAISY-DAMP problem is described; the component agents of the system are characterized; and the communication protocols system elucidated. The motivations and advantages in designing the system with authority decentralized is discussed. Progress in the area of Speech Act theory is proposed as playing a role in constructing decentralized systems.

This paper describes the DAISY-DAMP project currently under development at LMSC's Astronautics Division. This paper focuses on the distributed AI issues that are being addressed in this project. The utility of decentralizing authority in a distributed system of this class is discussed. A more complete description of the system's architecture and the role of a prototyping testbed can be found elsewhere.[1] The paper below is organized as follows:

Section 1 is a general introduction to the DAISY-DAMP system.

Section 2 describes the system's design architecture.

Section 3 addresses the communication protocol issues.

Section 4 characterizes lessons learned and future developments.

1   INTRODUCTION

DAISY-DAMP is a prototype version of an avant-garde electrical power management system. The project was initially funded for its potential utilization as an embedded control system on the NASA International Space Station. The global intent of the project is to design a generic power

---

[1]See the Published Proceedings of the 1986 SPIE Symposium on Advances in Intelligent Robotics Systems.

control system that offers considerable adaptivity in responding to evolving environmental demands.

The system is designed as a decentralized network of cooperating expert (or knowledge based) systems. Three of the four major agents of DAISY-DAMP are planning systems which are designed to work in parallel. The fourth agent is a diagnostic system responsible for fault identification and isolation. As such the system is a prototypical loosely coupled distributed AI system.

The design of this system as a DAISY offers the opportunity to explore some of the critical questions regarding the high level communication protocols that need to be utilized within loosed coupled systems. This paper will address some of these issues in the context of an engineering task.

A critical feature of the prototyping of this system involves the construction of a testbed that facilitates the construction and performance testing of a wide variety of DAI protocol systems. Although the testbed design and development will not be discussed in any significant detail here, a note should be made of its critical role in exploring many of the issues addressed within this paper. A good testbed for developing a DAI system should minimally support the expeditious investigation of the following issues:

- Mixes of 'data' and 'goal-directed' control in the system;

- Distributions of uncertainty and error in the input data;

- Types of communication policies used;

- Communication channel characteristics;

- The problem solving and communication responsibilities of each node;

- The authority relationships among nodes;
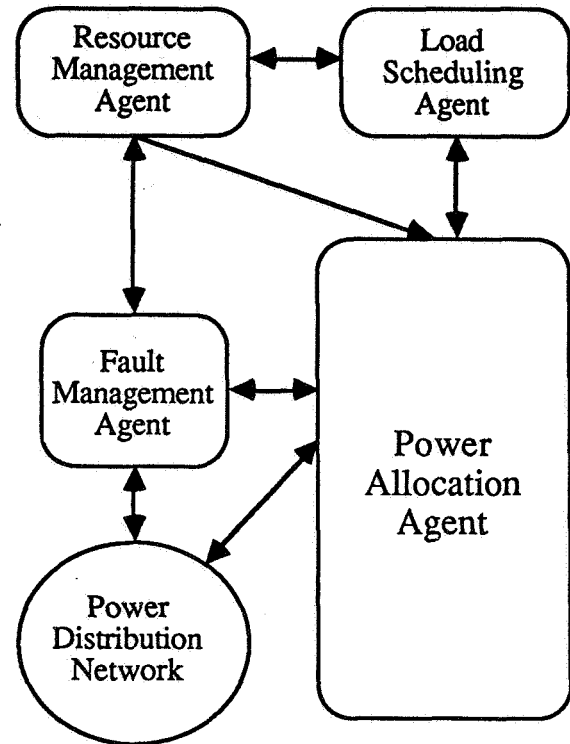
Figure 1: SYSTEM ARCHITECTURE

2 SYSTEM STRUCTURE

A first significant step in the design of DAISY-DAMP has been involved with the definition of the various system agents along with their roles and responsibilities. This section characterizes the current best definition of the four primary agents that together constitute the DAISY-DAMP. See figure 1.

2.1 POWER ALLOCATION AGENT

The power allocation agent is responsible for deciding how power ought to be routed from the various sources of power to the various loads requiring power. As such the problem is a system configuration task with an inherent need to per-

form dynamic reconfiguration.

The goal of this agent is to generate a configuration plan specifying what switches are to be thrown at what times in order to optimize the distribution of available power and minimize the frequency of switch throwing.

Part of the complexity in this domain comes about because of the intrinsic trade-off between these two objectives. In general as the commitment to optimizing the configuration increases, the frequency of switch throwing increases as well.

Additionally, the notion of an optimized configuration is composed of two competing objectives as well: redundant systems should be maintained on distinct circuits while, simultaneously, the unclaimed power in the system should be as evenly (intelligently) distributed as possible to insure power availability for the activation of unscheduled loads (e.g., the activation of the tracking motors).

Finally it should be noted that the notion of an even or fair distribution of excess power is a complex concept that should minimally include references to the power demands of the unschedulable loads on each of the configured circuits and their probability of being 'on' during the period under consideration.

Two interesting trends (laws?) should be noted here:

- As the power margin (i.e., the difference between the power available and the power required) decreases the switch throw frequency increases.

- As the number of powered redundant systems increase the even distribution of available power decreases.

One final consideration should be noted: the low vulnerability loads (i.e., those whose power supply can be interrupted without significant cost) should be distributed throughout the circuit net-

work such that in the context of the need to shed a load because of insufficient power on a particular circuit, a critical load need not be shed. It should be noted that the satisfaction of this constraint (although generally not highly prioritized) is also potentially in conflict with both the redundancy and even distribution constraints.

The power allocation agent module was the first module of DAISY-DAMP to begin development and is currently serving as a prototype for further developments.

2.2 LOAD SCHEDULING AGENT

The load scheduling agent's primary role is to determine when each of the various loads that require power ought to be energized. The load scheduling agent serves a central negotiating role in DAISY-DAMP. The agent is currently assigned the following responsibilities:

- Communicate with the OMS scheduler to determine global scheduling projections;

- Communicate with the Resource Management Agent to determine power available projections;

- Communicate with the Power Allocation Agent to determine the feasibility/advisability of implementing the current load schedule.

In a static or non-modifiable world the Load Scheduling Agent's task is a relatively straight forward one. The agent takes the power available plan from the Resource Manager and the task plan from the OMS scheduler and generates a plan/schedule for energizing the required loads within the bounds of the available power. Such a scenario also assumes that any produced load schedule can be implemented safely and efficiently by the Power Allocation Agent.

The situation is however both more complex and flexible than the scenario specified above suggests. There are satisfaction constraints that

203

guide the generation of the system configuration plan, the resource management plan, as well as the task plan that primarily respond to the structure of the load schedule plan. If these constraints are to be satisfied then the global planning process must involve either a degree of parallelism or be iterative in nature.

A major goal in the development of DAISY-DAMP is to generate a system of cooperating agents that can readily recognize problems that impact the development of the evolving plans of the agents to which they interface.

A few scattered exemplars of the problems in various evolving plans that need to be monitored include the following:

- Planned high frequency switching rates on high power switches reflects a need to modify the planned power margin (i.e., more power or less demand during 'critical' periods).

- The inability to generate load plans that are in concert with the estimated unit estimates derived from the task plan reflects a need to modify the OMS task plan.

- An estimate that the long term health of the electrical power system is being jeopardized is a reflection that the current short term negotiation policy is too aggressive.

- The inability to generate a plan that is stable for a temporal period of time sufficient to dedicate the computational resources to emergency replanning without negative side effects reflects a level of resolution in the global planning process that is excessive.

2.3 RESOURCE MANAGEMENT AGENT

The Resource Management Agent's primary responsibility is to insure the availability of electrical power and the health of the power system. The agent is responsible for the following functions:

- monitoring and projecting the power availability;

- generating plans which specify how much power is to be made available per unit time for powering loads;

- recharging and reconditioning the power system batteries;

- assuring the long term health of the system.

The complexity of the resource management task is a function of the inherent flexibility in how the resource management task is achieved. While the batteries have a limited capacity to provide power, the timing of their utilization is quite flexible within the given constraints. They are under demand during the dark cycle, must be discharged periodically for reconditioning purposes and have some health maintenance requirements but otherwise the resource management agent has a considerable degree of freedom to choose when the best time to 'borrow' power is and when the borrowed power ought to be 'repaid'. As a consequence of this flexibility, multicycle projections and utilization commitments are likely to play a major role in the generation of the resource utilization plan. If, for example, an expressed short-term need for additional power can be balanced with an expressed willingness to limit power usage for a specified period in the future and such an arrangement is judged to not endanger the health of the power system than the request may be granted.

The effect of this arrangement is to produce an ongoing negotiation between the resource management agent and the load scheduling as to how much power is to be provided during the foreseeable future.

2.4 FAULT MANAGEMENT AGENT

The fault management agent is primarily responsible for identifying and isolating power hardware

failures. The agent is also responsible for tracking failures and maintaining statistics regarding the MTBFs (Mean Time Between Failures) for the various components.

The fault identification function actively monitors power system sensors for both failure trends and positive component failures.

The fault isolation function is an active component that utilizes both sensor readings and active configuration manipulation to localize the fault. The active configuration manipulation requires an interface to the power allocation agent.

3 COMMUNICATION PROTOCOL STRUCTURE

The global behavior of Daisy-Damp is largely a function of the extent to which the individual components can be induced to function in concert. This mutual dependency highlights the role of the communication protocols. The protocols of particular interest are at the seventh level. The function of these high level communication rules is to determine not only what types of messages should be transmitted and when they should be transmitted but what the content of those messages should be.

One of the important questions in formulating the structure of these communication protocols concerns the assignment of authority to initiate messages and thereby control the processing of the associated agents. The Daisy-Damp system has been designed with the authority decentralized. This means that each of the various agents can independently initiate actions in its associates.

The primary motivation for this decentralization lies in the uncertainty with which agents that are simultaneously solving inter-related subproblems can know the current state of their associated agents and therefore how and where specific problems ought to be handled.

For example; the optimal solution to the power scheduler's recognition that it is being forced to throw its switches at an unacceptably high rate during some temporal interval, is not deducible by the power scheduler in isolation. The best solution may involve the resource manager increasing the available power for that interval or it may involve the load scheduler dropping some of the power demands for that period or it may involve simply doing nothing. Modification to the plans of either the load scheduler or the resource manager is likely to require coordination with the other.

This problem can be solved by centralizing authority in a single agent which maintains accurate models of the various agents and therefore can assign tasks and solve problems appropriately. Unfortunately the consequent overwhelming communication requirements invalidate many of the motivations for initially designing the system in a distributed fashion and result in a superfluous redundancy.

The better solution to this problem lies not in maintaining complete models of each agent at either each node or at a centralized node, but rather in exploiting the uncertainty by generating messages that are intended by the initiating agent to be interpreted by the receiving agent in terms of its current state. In the example above, a message that simply announces the problem should be sufficient, with an appropriate protocol system, to initiate the communication and problem-solving process that produces the optimal outcome.

The key to such a strategy lies in each agent's possession of a body of knowledge that governs the production and interpretation of messages along with the knowledge and capacity to exploit the fact that this knowledge is shared among the various agents. Since this body of knowledge is largely application independent the perpetual problem of maintaining knowledge base concurrency in distributed AI systems is largely replaced by the one-time engineering task associated with the construction of this 'communication module'.

205
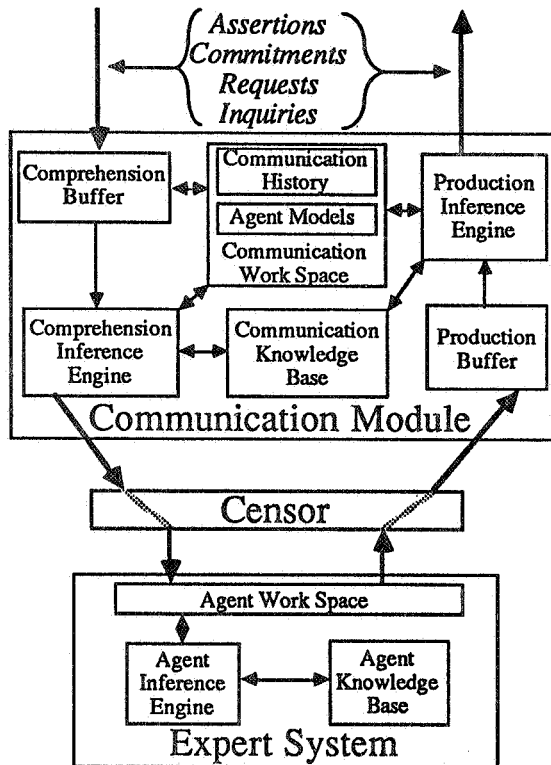
Figure 2: Communication Module Architecture

This communication module is in itself a standard knowledge based system whose function is largely independent of the agent within which it resides or the problem domain being addressed. Like any knowledge based system it is subject to incremental development and can serve a valuable role within a developing distributed AI system at a reasonably early stage in its development. The global structure of a complete system is shown diagrammatically in figure 2. The primary components of this module are summarized below:

3.1 THE CENSOR

The Censor is a relatively simple element whose responsibility involves determining when specific facts or goals in the agent's workspace have reached a level of certitude to justify consider-

ing for communication. The critical confidence value can be modified as a function of a system developer's manipulation or by real-time message interpretation. Enhancements to this function include the ability to censor subclasses of the active facts and goals for different levels of certitude. Some flexibility exists in where the censor function can be reasonably situated; some communication module designs will encode similar functions into the module itself.

3.2 COMMUNICATION WORKSPACE

The communication workspace contains the facts and goals that are of relevance to the communication module. The workspace is segmented into three types of knowledge: agent models, communication history and intermediate results.

The agent models section includes two kinds of information: declarative information about the functional responsibilities of each agent and current state knowledge that is expressed or implied by the transmitted messages.

The communication history section encodes information about the active 'topics', including: what messages have been exchanged between which agents; what topic/problem each message addressed; what the current state of each topic is and the source(s) and message class(es) of the expected next message in each topic area.

The intermediate results section includes both partial conclusions regarding the appropriate interpretation of a received message and the appropriate message class and message content for a transmitted message.

3.3 COMMUNICATION KNOWLEDGE BASE

The Communication Knowledge Base is the most universal of the communication module components and the keystone of the module. The

communication knowledge base encodes a body of knowledge about the communication process itself that enables both information/knowledge that is to be communicated to be encoded in such a way as to exploit the flexibility in the system and messages that are received to be properly interpreted in terms of the receiving agent's current state. The declarative knowledge that must be encoded within this module falls into two broad categories:

1. The conditions under which a particular message type can justifiably be transmitted;

2. The conditions under which the content of a particular message type can justifiably be transmitted;

The structure and representation of this knowledge is a significant task that draws heavily on previous work in both Distributed Artificial Intelligence theory and Speech Act theory.[2] [3]

Communication knowledge base rules are responsible for utilizing this knowledge to both produce and comprehend messages as a function of the current state of the work space and the contents of the message buffers. One subset of these rules are specifically dedicated to determining the implications of the fact that the structure of the messages being received is a product of a system that is utilizing this same body of knowledge. Another subset of rules is specifically dedicated to responding to messages that violate the agent's expected messages set.

The optimal structure and function of these rules has not yet been determined for the general case. Some of the open question include:

- To what extent/depth should the production and comprehension of a message exploit the

fact that the protocols are shared. For example, should the structure of a constructed message exploit the knowledge that the receiving agent will be examining the message for what it implies about the transmitting agent. Should the extent of this consideration be allowed to vary over time.

- To what extent should the maintained agent models be allowed to become a communication focus in itself. What are the best vehicles for assuring pragmatic agent models.

3.4 COMMUNICATION MODULE GLOBAL BEHAVIOR

The 'power' of such a system is a function of many factors including those referenced above; but the following high level capabilities characterize, in general, the consequence of implementing such a module. The generic communication module is expected:

- To determine what state-information possessed by the home agent is of potential relevance to each associated agent;

- To determine what state-information possessed by the associated agents is of potential relevance to the home agent;

- To determine what actions planned by the home agent are of consequence to each associated agent;

- To determine what actions planned by the associated agents are of consequence to the home agent;

The communication module is also responsible for determining the justifications and/or conditions that support and/or determine the state-information and planned actions of the associated agents. This capability plays a critical role in determining the optimal negotiation foci.

---

[2]c.f., Searle, J. 1969. Speech Acts: An Essay in the Philosophy of Language. Cambridge, England: Cambridge University Press.

[3]Linguistic Communication and Speech Acts. Kent Bach and Robert Harnish. MIT PRESS, 1979.

3.5 MESSAGE TYPES

One of the important determinants of the effectiveness of the communication module in generating coordinated system behavior has to do with the number of uniquely defined message types. A message type is an explicitly represented wrap for the message contents. Message types can be roughly mapped onto a class of words that in the world of Speech Act Theory are referenced as illocutionary verbs. Generally, these verbs are grouped into four classes: constatives, directives, commissives and acknowledgements. Instances within these classes include, for example: concessives, retractives, ascriptives, prohibitives, advisories, offers, accepts and bids.

A minimal set of message types for the DAISY-DAMP system has been defined to include: assertions, commands, requests and queries. These four message types correspond roughly to a prototypical instance of each of the four classes referenced above. As prototypical instances they carry the presuppositions and implications of the class with few of the attributes that would distinguish them from other members of the set. As additional message type distinctions are made within the each of the represented types, additional presuppositional and implicational attributes are included. Thus the power of the communication module can be enhanced by the refinement of the message type hierarchy.

4 SUMMARY

Our work on DAISY-DAMP has given us the opportunity to consider some the qualities that characterize a particular problem as a good candidate for implementation as a distributed AI system.

A primary feature of this application that suggests a DAI approach, is the simultaneous planning aspects of the task. An obvious way to approach this problem, if it were to be solved by human ground operators, would be to divide the problem into several simultaneous planning problems. A tremendous gain in system comprehensibility is gained if the system simply models this intuitive approach to solving the problem.

Having committed to a distributed design we have been obligated to confront the high level network control issues that are inherent in such a design. Approaches that depended on knowledge base concurrency whether in a centralized control agent or dispersed throughout the network were seen in this domain to be inefficient and unnecessarily redundant. A solution to the control problem is recognized in a decentralized system that makes robust use of mutually possessed intelligent communication modules. The work in Speech Act Theory seems to offer promising insights into the structure of the knowledge base of this component.

4.1 REFERENCES

1. Searle, J. 1969. Speech Acts: An Essay in the Philosophy of Language. Cambridge, England: Cambridge University Press.

2. Bach, K. and R. Harnish. 1979. Linguistic Communication and Speech Acts. Cambridge, Massachusetts: MIT PRESS.

3. Brady, M. and R. Berwick, eds. 1983. Computational Models of Discourse. Cambridge, Massachusetts: MIT PRESS.

4. Shimanoff S. B. 1980. Communication Rules: Theory and Research. Beverly Hills, California: Sage Publications.

5. Lessor, V. R. and D. D. Corkill. 1983. "The Distributed Vehicle Monitoring Testbed: A Tool Fore Investigation Distributed Problem Solving Networks", The AI Magazine, Fall, pp. 15-33.

6. Davis, R. and R. G. Smith. 1981. "Negotiation as a Metaphor for Distributed Problem Solving", Artificial Intelligence vol. 20, No. 1, pp. 63-109.

KNOWLEDGE-BASED FAULT DIAGNOSIS ON FUTURE SPACE VEHICLES

Robert L. Schroeder
General Dynamics Space Systems Division
San Diego, CA

ABSTRACT

There is a need for knowledge-based fault detection and diagnostics on the future generation of space vehicles. These vehicles have a more demanding set of requirements than today's vehicles. A tank pressurization system was chosen to explore the potential for an on-board knowledge-based fault detection and diagnostic system. Specific tasks were to investigate sensor-based diagnostics, appropriate levels of knowledge representation, the generation of further diagnostic tests, the diagnosis of two failures, and effective forms of graphical representation.

INTRODUCTION

Future space vehicles such as the Orbital Transfer Vehicle (OTV) will experience a change in the way Integration, Checkout, and Launch Processing will be performed. There are several reasons for this. Space basing will require more autonomous vehicles. Vehicles may be reuseable up to 100 missions. The space-based vehicle must be checked out, serviced, and integrated with payloads through the part-time efforts of an astronaut whose primary expertise is in material or life sciences. The Space Station cannot accomodate the same Ground Support Equipment we are used to seeing. Total life-cycle costs must be reduced. A shift to on-board checkout and fault detection would help satisfy some of these requirements.

The General Dynamics Computer Controlled Launch System (CCLS) is a good example of the present state-of-the-art in Launch Processing for unmanned vehicles. The CCLS provides problem isolation capability; flexible operator control; historical data records; identical, safe, rigorous testing; lessened chance of human error; and software development capability. It performs checkout of the major system elements, controls the cryogenic tanking and does the prelaunch sequencing. And finally, it allows automatic redline monitoring during launch.

The CCLS is a mature system originating some 20 years ago. It is also a modern system, having undergone 3 major revisions. It consists of 4 Harris

computers, extensive operator interface including color graphics, communication links, and front-end interfacing in a dual redundant configuration.

CCLS is a real-time system with a response time to many events of seven milliseconds. The extent of CCLS is best realized by considering that it incorporates over a million lines of code. It operates with little operator intervention, does a great deal of data analysis, and gives GO/NOGO indications with explanations. This sets the standard for future Launch Support Systems: they must be smaller, cheaper, even more autonomous, easier to develop, and just as capable.

The purpose of this IRAD is to investigate techniques to develop these future Launch Support Systems. Such techniques might include improved Fault Detection/Isolation and better ways of graphically representing data. Artificial Intelligence (AI) is of particular importance. AI is a term assigned to a field of research attempting to emulate the intelligence of the human mind. A number of methods for classifying and using knowledge are being developed as a byproduct of this research. These methods, regardless of their fidelity to natural intelligence, are useful in their own right. This work considers only the sub-field of Expert Systems as distinguished from robotics, natural language interpretation, and artificial neural nets. Expert Systems have already been applied to a number of off-line diagnostic type problems. On-line diagnostic systems include NAVEX (Maletz) and LES (Scarl).

SYSTEM DESCRIPTION

Symbolics was the selected hardware because it was available and because of its reputation as the standard in AI research. Both ART and KEE were considered for the software. ART was chosen primarily because of a perceived speed advantage. We also wanted to explore the ART viewpoint feature. Although not required by this problem, more complex problems would require both non-monotonic and hypothetical reasoning. ART permits levels of viewpoints which seem designed for this problem. ART will be discussed more later.

Different areas were considered for a prototype Expert System. Electronic systems were ruled out because the future bus-based systems are not yet well defined. Fluid systems are complex and critical and yet have a much smaller number of basic components. A problem of tank pressure control was eventually chosen from the fluids area. The next step was to determine the strategy of reasoning about errors. It was decided to rely on reasoning from what is known as first principles because of the following: (Davis)

1) it provides a strong degree of machine independence;
2) it makes the system easier to construct and maintain;
3) it facilitates defining the programs scope of competence;

210

4) it makes the system capable of dealing with novel symptoms.

What we want to avoid is an ad hoc collection of *if* symptom, *then* fault rules. We will not have the luxury of observing the vehicle in operation for a long period of time to compile fault trees and fault dictionaries. More importantly, it quickly became evident that the biggest obstruction to the acceptance of Expert Systems in this environment is more than a narrow technical issue. There are additional automation features which could be implemented on CCLS today if it were not for safety concerns (it is difficult to prove these concerns are not valid.) An Expert System which reasons from first principles will permit a more rigorous analysis of its operation than one consisting of a collection of shallow rules. This may be the only way to verify and validate an Expert System.

We first describe the system in the following way:

a) the structure is described by a set of relations linking the various components.
b) constraint rules describe the function of each type of component taking care not to intermix any knowledge of the structure of the system.

These relationships are the key to a valid system. Faults which modify the structure in non-obvious ways must be considered. The classical case is that of an electrical short.

The program proceeds by interaction of the following rules:

a) state-change rules are controlling the process. In a strictly monitoring function they would be absent. If the program were also controlling the process, these rules would be the heart of the control logic. In our prototype they are simulating the control function.
b) propagation rules use the constraint rules and propagate pressures throughout the system.
c) dependency rules set up dependency nets which are used to determine which components could be involved in a fault symptom.
d) fault detection rules test expected and actual values of sensors and indicators.
e) fault testing rules suspend the constraints for each item in the dependency net and thereby isolate to a component or group of components.
f) maintenance rules are artifacts of the programming language and style of the programmer.
g) graphics rules allow operator input, control and explanation.

There is another group of rules which provide the simulated fault entry of

the demonstration program. They are completely separate from the control and monitor rules.

The use of an Expert System shell is a excellent way to get a program running quickly. ART is particularly powerful in the organization, classification, and manipulation of the data base. Graphics are easy to incorporate and easy to use. A rich variety of programming constructs allows a wide variety of pattern matching operations to be easily performed. Nevertheless, certain details of implementation have become apparent:

1) The first problem is speed of execution and memory requirements. Even our simple prototype became very cumbersome to work with. This seems to be related more to the number of schemata than to the number of rules. Obviously there are always ways to speed any program up, but this involves what some would consider kludges. Another reason for slowness is discussed in item 5 below.

2) It is not convenient (although it is possible) to save the current state of facts and schema. In other words you cannot stop in the middle of a run and pickup where you left off the next day. Each run starts from the initial state. This curtailed certain ideas we were going to explore concerning graphical manipulation of data.

3) The name of a slot cannot be a variable. This necessitated large case statements, extra rules, and needless repetition of data.

4) The compiled load cannot be saved. Every load is a new compile. Although the incremental compile feature usually negates this problem, it appears that it doesn't always work as expected when .lisp files are involved.

5) Graphics can be created in two ways--with the Icon Editor or with programmed functions. The editor is easy and the programmed functions are efficient. That is the problem. One would like to have easy and efficent together. Of course, it could always be argued that with practice comes ease.

6) When logical dependencies become false they remove slot values from the data base. This is probably what some applications desire. Unfortunately, we would have preferred the slot value be set to an initial or default state.

7) We experienced a problem with backward chaining when several rules created the same goal patterns. Since backward chaining was not necessary, we didn't investigate this further. It could have been a problem with our implementation.

8) While viewpoints did appear to be a very useful tool, they were also extremely demanding in their use of machine resources. So much so as to be virtually unusable in any further extension of this application.

CONCLUSIONS AND RECOMMENDATIONS

A shell requires too much overhead for any embedded application. An application of this type probably doesn't require all the frills the shell provides; but, in turn, has unique requirements of its own. A user written inference engine tailored to the task is necessary.

One impression we gained from this effort was the essentially deterministic and associative nature of our problem solution. A colloquial way to express this is "where is the intelligence." A more rigorous explanation is given by quoting from a treatise on this subject (Sutherland). "AI constructs cannot endogenously perform the sorts of inference operations that stochastic decision exercises demand." "Lacking such a capability, AI constructs cannot transcend, but rather must compete with instruments of the type available from traditional decision disciplines." In many instances, certain AI properties might place them at an efficiency disadvantage." "AI facilities are vastly more sophisticated for ordering and manipulating knowledge than for either employing it or generating it."

With these limitations of current AI techniques in mind it should not be surprising to learn that the fundamentals of a rule-based production system have been reduced to IC chips (Helley) (Togai and Watanabe). Further research in this area should concentrate on a consistent representation of vehicle subsystems at the appropriate level and the incorporation of an appropriate inference engine in a combined hardware/software representation.

REFERENCES

1. Malitz, M. C., "NAVEX:Space Shuttle Navigation Expert System," Copyright 1985 Inference Corp.
2. Scarl, E. A., Jamieson, J. R., Delaune, C. I., "Process Monitoring and Fault Location at the Kennedy Space Center," Sigart Newsletter, No. 93, July 1985.
3. Davis, R., "Diagnostic Reasoning Based on Structure and Behavior," Qualitative Reasoning about Physical Systems, MIT Press, 1985.
4. Sutherland, J. W., "Assessing the Artificial Intelligence Contribution to Decision Technology," IEEE Trans on Systems, Man, and Cybernetics, Vol. SMC-16, No. 1, Jan/Feb 1986.
5. Helly, J. J., Bates, W. V., Cutler, M., "A Representation Basis for the Development of a Distributed Expert System for Space Shuttle Flight Control," NASA Technical Memorandum 58258, May 1984.
6. Togai, M., Hiroyuki, W., "Expert System on a Chip:An Engine for Real-Time Approximate Reasoning," IEEE Expert, Fall 1986.

EXPERT SYSTEMS FOR MSFC POWER SYSTEMS

David J. Weeks

Information and Electronic Systems Laboratory
NASA/Marshall Space Flight Center
Marshall Space Flight Center, Alabama 35812

Abstract

Future space vehicles and platforms including Space Station will possess complex power systems. These systems will require a high level of autonomous operation to allow the crew to concentrate on mission activites and to limit the number of ground support personnel to a reasonable number.
The Electrical Power Branch at NASA's Marshall Space Flight Center is developing advanced automation approaches which will enable the necessary levels of autonomy. These approaches include the utilization of knowledge-based or expert systems.

Introduction

As the electrical power requirements for spacecraft increase, the complexity of managing such systems also increases. A key lesson learned from the first American space station, Skylab, was that future spacecraft with larger power systems must minimize ground support and crew involvement.

Since 1982 Marshall Space Flight Center has been involved with the development of expert or knowledge-based systems to facilitate the automation of electrical power systems. These expert systems focus on the problems of fault diagnosis and payload scheduling. Future plans involve the development of expert systems for intelligent data reduction, payload scheduling including implementation of the revised schedule, fault recovery, battery management, trends analysis, and component failure forecasting.

215

Several expert system prototypes have been developed. This paper will give an overview of several of these systems including the Fault Isolation Expert System (FIES II), the Space Station Experiment Scheduler (SSES), Space Telescope Electrical Power System Diagnoser named NICBES (Nickel-Cadmium Battery Expert System), and the Loads Priority List Maintenance System (LPLMS). Other current and proposed activities will also be briefly discussed.

Fault Isolation Expert System

The second generation of the Fault Isolation Expert System (FIES II) was completed in 1985. This system diagnoses faults in a electrical power system breadboard. The breadboard is a complete end-to-end electrical power system with power generation simulation, energy storage subsystem (NiCd batteries), and a power management and distribution system. Employing three power channels, the breadboard distributes 350 watts to the load simulators.

A block diagram of the system architecture is given in Figure 1. This knowledge-based system is implemented in ART, the Automated Reasoning Tool, and is executed on a Symbolics 3670 workstation. As ART was observed to be cumbersome and slow for demonstrations of this application, much of the code was stripped and implemented directly in LISP.

An electrical fault can be injected into the breadboard power system by keyboard command or by manually setting a toggle switch on the fault insertion panel. Whenever a fault is injected into the breadboard, the expert system reasons about the cause of the anomaly and diagnoses the faulty component or most likely failure candidates.

Developed by Martin Marietta Denver Aerospace, this project was originally funded by the Office of Aeronautics and Space Technology (OAST) before sponsorship was transferred to the Office of Space Station (OSS) and directed by the Software and Data Management Division and the Electrical Power Branch at Marshall Space Flight Center.

FIES II SYSTEM ARCHITECTURE

SYMBOLICS 3670

BREAD-BOARD POWER SYSTEM	MICRO-PROC. SYSTEM	INTERFACE SOFTWARE	EXPERT SYSTEM FIES II	USER INTERFACE
HARDWARE SIMULATION OF FAULTED POWER SYSTEM	HANDLES COMMUNICA-TIONS BETWEEN BREADBOARD AND SYMBOLICS ERROR BAND CHECKING	FAULTED SENSOR DATA SENT TO FIES II RECONFIG. REQUESTS SENT TO MICRO	LOCATES FAULT AND SPECIFIES FAULT TYPE	ALLOWS USER TO CONFIGURE SYSTEM AND CHOOSE FAULT (SENT TO MICRO) REPORTS TO USER

5-122-6-1

FIGURE 1. FIES II

Space Station Experiment Scheduler

The Space Station Experiment Scheduler (SSES) is a proof-of-concept demonstration prototype expert system which performs scheduling/rescheduling activities for payloads much faster than conventional approaches. Though a relatively simple model, this expert system demonstrates that a dynamic rescheduler embedded in the power management system can handle perturbations to the available power to help ensure that power is utilized as it becomes available and critical loads are never shed unless absolutely necessary.

Space power cannot be wasted due to the historical cost of $1000 per kilowatt versus about 5 cents for terrestial use. It is imperative that the 'wrong' load is never shed unless absolutely necessary. A DOD or ESA 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 its importance as an expensive crystal nears completion and cannot be interrupted without flaws occurring.

SSES can reschedule about 50 payloads for a two week period in a couple of minutes. This model employs only a fraction of the 200 scheduling constraints that the Marshall Space Flight Center Experiment Planning System (EPS) utilizes, but does consider power consumption, payload duration, intermittent usage, crew attendance required, and priority class. SSES was developed by Technology Applications Inc. of Jacksonville, Florida under the direction of the Electrical Power Branch for the Office of Space Station under the Space Station Advanced Development program.

Hubble Space Telescope Electrical Power System Diagnoser

The Nickel-Cadmium Battery Expert System (NICBES) will be interfaced with the Hubble Space Telescope Electrical Power System breadboard at Marshall Space Flight Center in November, 1986. A block diagram of the configuration is shown in Figure 2. As presently configured, this breadboard is operated

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

HUBBLE SPACE TELESCOPE ELECTRICAL POWER

SYSTEMS BREADBOARD
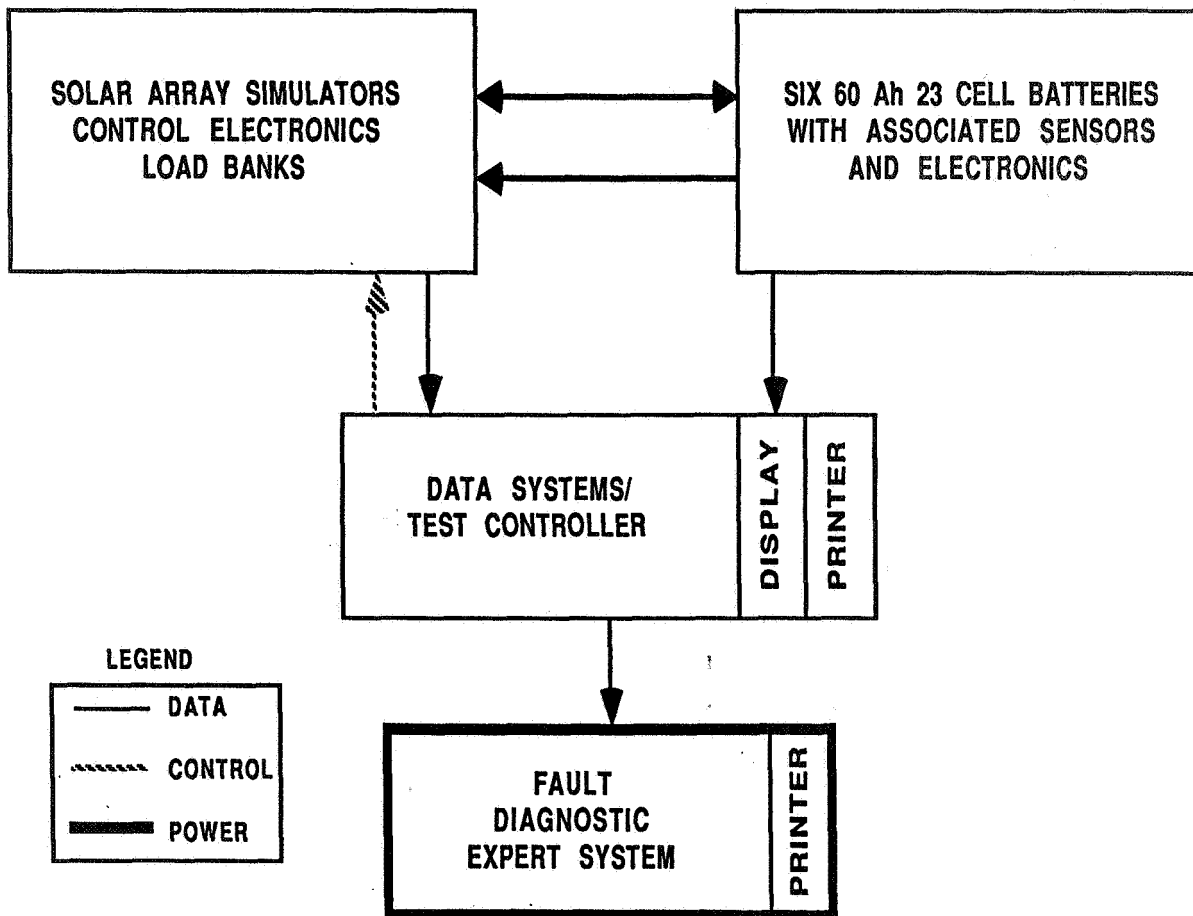
FIGURE 2. NICBES AND HUBBLE SPACE TELESCOPE BREADBOARD

219

NICBES will independently verify the occurrence of an anomaly, diagnose the malfunction, and print out recommended appropriate actions for the test bed personnel to take once they arrive. The expert system may also be interrogated for battery status information including plots and histograms of recent battery information. NICBES is implemented in Arity Prolog on an IBM PC/AT. It was sponsored by the Office of Aeronautics and Space Technology and developed by Martin Marietta Denver Aerospace.

Load Priority List Maintenance System

The Load Priority List Maintenance System (LPLMS) is an expert system prototype model that has been developed for the Space Station Module power management and distribution system. LPLMS maintains a real time dynamic representation of all the module loads and relevant facts such that applicable rules can fire to reorder portions of the list as situations change. LPLMS will be employed with a dynamic scheduler/ reschuler, intended to ultimately cooperate with an enhanced SSES-type expert system.

The loads in a laboratory module may have dynamic priorities. A critical materials processing experiment involving crystal growth may have a different priority as it nears completions if growth cannot be interrupted. Other factors may change priorities such as equipment malfunctions. Such an expert system 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 'wrong' loads are not shed unnecessarily.

The LPLMS is currently implemented in the production language HAPS on the Symbolics 3645 workstation. Current plans call for porting this system to the Xerox 1186 in LOOPS. The prototype was developed under contract to Martin Marietta Denver Aerospace as part of the automation approach to the Space Station Core Module power system under Advanced

Development for Space Station.

Integrated Electrical Power System Scheduler/Fault Isolation

Currently, an effort is underway to evaluate scheduling and fault diagnosis expert systems previously developed for Marshall Space Flight Center and at Kennedy Space Center. Candidate systems will be upgraded to be interfaced with the FIES II electrical power system breadboard to provide cooperative autonomous scheduling, schedule implementation, fault diagnosis, and fault recovery functions. The University of Alabama in Huntsville is assisting in this evaluation/enhancement effort.

The candidate scheduling systems under review are the SSES and a payload scheduler developed by the University of Alabama in Huntsville. From these candidates, a fast payload scheduler that can be triggered by breadboard faults will be developed to actually implement the new schedule by controlling loads and switchgear.

Candidate systems for the fault diagnosis and fault recovery expert system include FIES II and KATE. KATE is the Knowledge-based Automatic Test Equipment expert system, a follow-on effort to LES, the LOX Expert System developed at Kennedy Space Center. KATE was implemented in large memory Golden Common LISP on the PC/AT and is being ported to the Symbolics 3670 at Marshall Space Flight Center. KATE not only diagnoses faults in a breadboard but also performs corrective actions to effect fault recovery.

Future Directions

Proposed plans call for research and development efforts in the area of intelligent data reduction. Efforts would be centered on reducing battery telemetry data from the Hubble Space Telescope electrical power system breadboard. At any

221

given time, perhaps 85 to 95 per cent of such data is insignificant. As such data reduction is extremely labor intensive, an expert system could prove extremely beneficial. A goal is to reduce the telemetry data by an order of magnitude.

If battery telemetry data can be reduced by an expert system, other system and subsystem telemetry data may also be handled by such expert systems. This could lead to better health monitoring, fault prediction, trends analysis, and overall fault management performed autonomously on-board spacecraft in the future.

Another proposed effort involves the development of a large expert scheduler/controller for the science payloads in the Core Module Integration Simulator (CMIS) at Marshall Space Flight Center. Such a system would be useful to personnel who will plan, schedule, and coordinate science experiments for Space Station and could be an extension of the power management system on-board. This system would initiate new schedules dynamically when sensor inputs indicate payload, power system, or other system anomalies that could perturbate the current baselined payload schedule.

Other expert system developments are planned for the Power System Development Facility at Marshall Space Flight Center. These expert systems would perform fault analysis, data reduction, trend analysis, loads scheduling, and battery management. Breadboards in this facility include the Autonomously Managed Power System (AMPS), a 20 kilohertz system, and the core module power management and distribution system. A layout of this facility is shown in Figure 3.

The AMPS breadboard is an agency test facility for the verification and characterization of power components and power management automation approaches. Its embedded processors are networked via Ethernet which provides a good interface for adding advisory expert systems as shown in Figure 4.

POWER SYSTEM DEVELOPMENT FACILITY
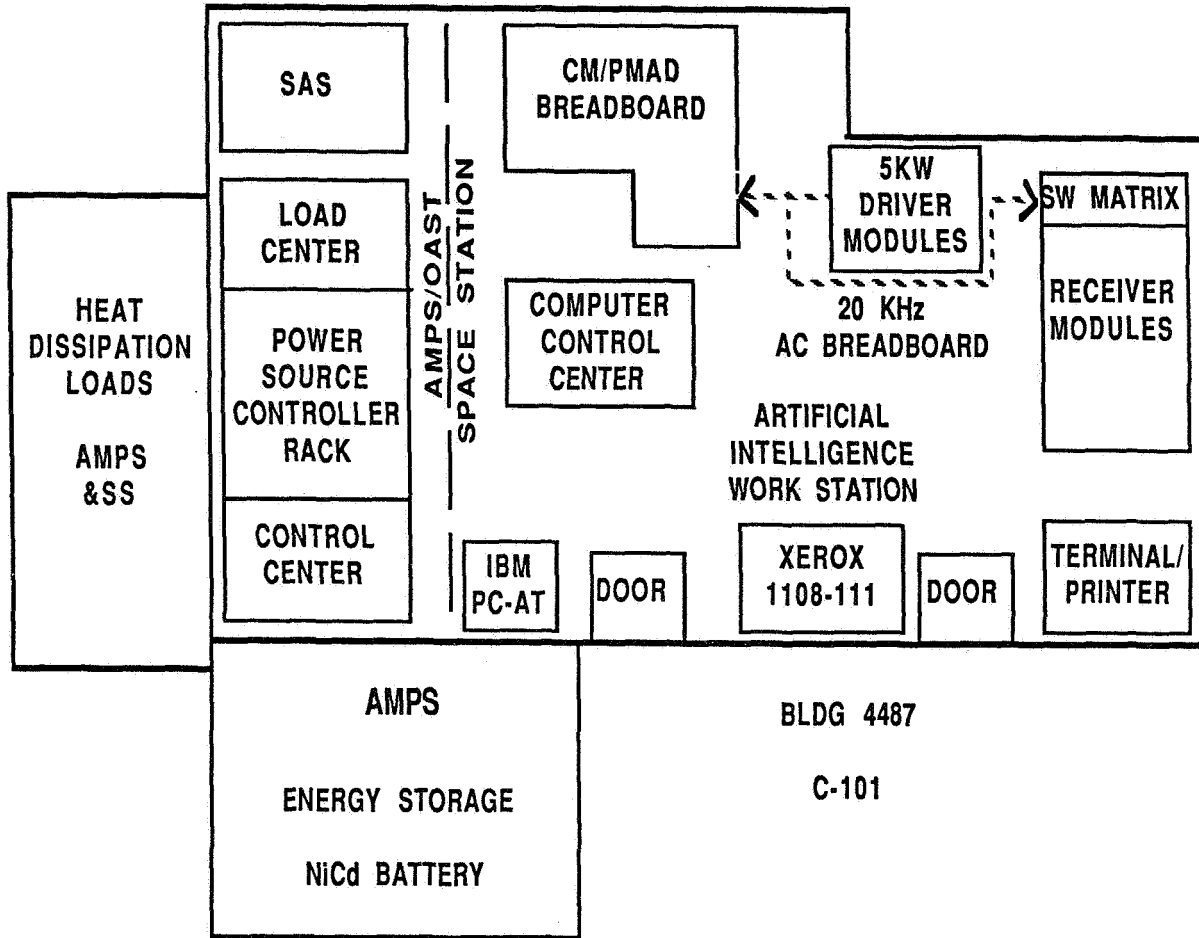
FIGURE 3. MSFC POWER SYSTEM DEVELOPMENT FACILITY LAYOUT

5-124-6-1

AUTONOMOUSLY MANAGED POWER SYSTEM
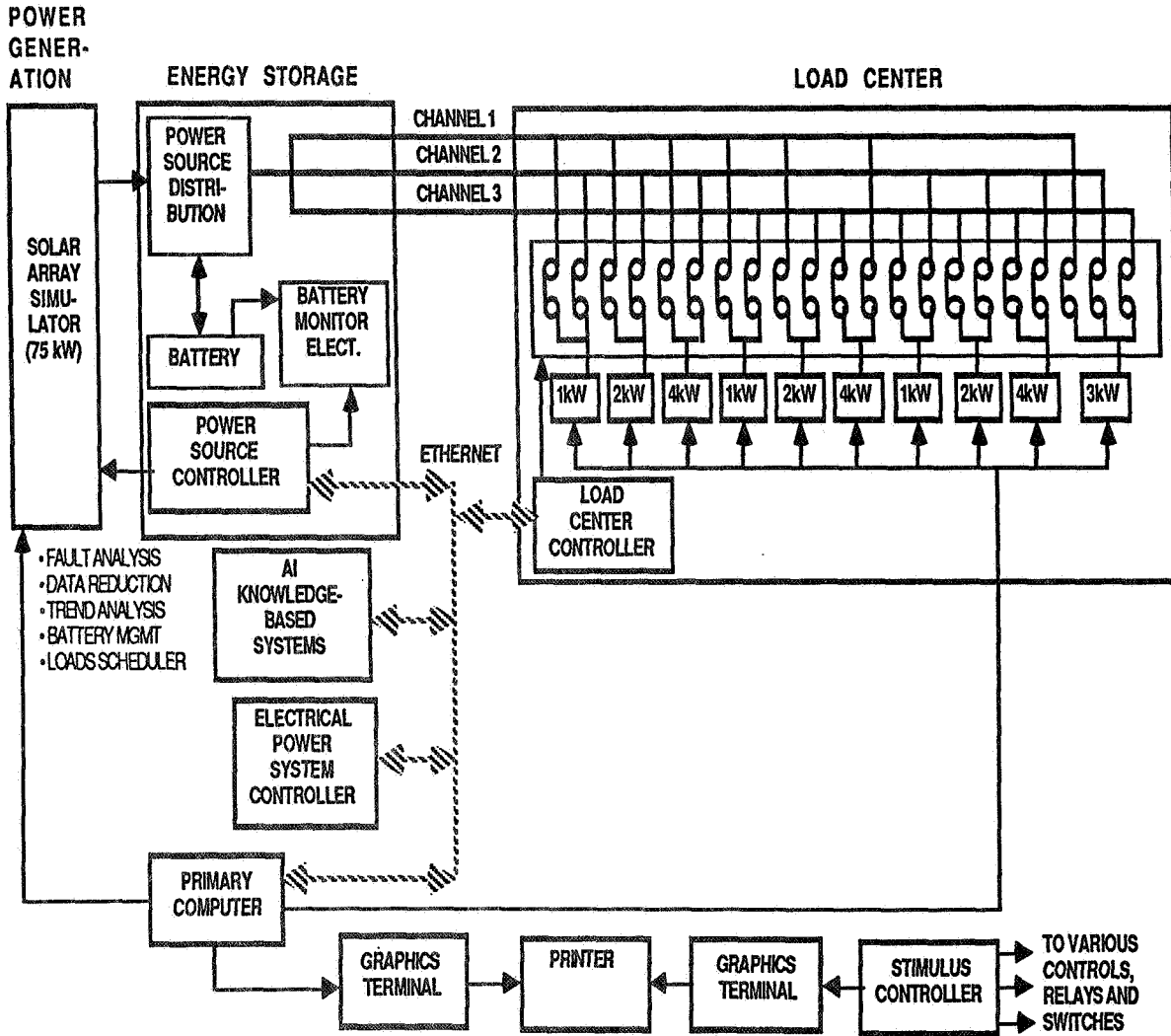
FIGURE 4. CONFIGURATION OF AMPS

5-125-6-1

Conclusions

Expert systems activities relating to power systems at Marshall Space Flight Center dates back to 1982. To date, most of these activities have been performed under contract support. Emphasis is currently being placed on developing more in-house expertise in this technology as it relates to the automation of electrical power systems.

Current efforts are centered around the FIES II breadboard, the Hubble Space Telescope electrical power system breadboard, and the Power System Development Facility. Efforts surrounding the FIES II breadboard are focused on the Symbolics 3670 with ART and KEE installed. The Hubble Space Telescope breadboard efforts are concentrated on the IBM PC/AT computer while the power System Development Facility employs the PC/AT and the Xerox 1109. Additional Xerox 1186's will likely be added to this facility.

The electrical power system has been identified by several groups as an ideal system for the application of expert and knowledge-based systems. These groups include SRI and ATAC, the Advanced Technology Advisory Committee established in response to public law regarding advanced automation and robotics for the Space Station. The Electrical Power Branch is earnestly dedicated to advancing spacecraft power system automation approaches and techniques including the utilization of expert and knowledge-based systems.

Acknowledgements

Many of these expert system development efforts are coordinated with Audie Anderson and Caroline Wang of the Software and Data Management Division in the Information and Electronic Systems Laboratory at the Marshall Space Flight Center.

References

1.   D.J. Weeks, "Application of Expert Systems in the Common Module Electrical Power System," _SPIE Vol. 580 Space Station Automation,_ Cambridge, 1985.

2.   D.J. Weeks and R.T. Bechtel, "Autonomously Managed High Power Systems," _Proceedings of the 20th IECEC,_ Miami, 1985.

3.   N. Kirkwood and D.J. Weeks, "Diagnosing Battery Behavior with an Expert System in PROLOG," _Proceedings of the 21st IECEC,_ San Diego, 1986.

4.   T. Hester, "A Real Time Fault Isolation Expert System," _Proceedings of the 21st IECEC,_ San Diego, 1986.

5.   R.A. Touchton, "Common Module Dynamic Payload Scheduler Expert System," _Proceedings of the 21st IECEC,_ San Diego, 1986.

6.   S. Floyd and D. Ford, "An Expert System for Dynamic Scheduling," _Proceedings of the 21st IECEC,_ San Diego, 1986.

A GENERIC EXPERT SYSTEM
FOR MATERIALS PROCESSING IN SPACE

Kristinn Andersen, George E. Cook, Alvin M. Strauss

Vanderbilt University,
P.O. Box 1824, Station B,
Nashville, Tennessee 37235

Abstract

A generic expert system is described for inspecting materials processed in space (MPS). The system may be applied, with the appropriate knowledge base, to any of the nondestructive testing methods (NDT) which are appropriate to MPS. Regardless of the method being used, the inspection process consists of three tasks: (a) signal or image processing of the NDT output and feature extraction, (b) interpretation of features in terms of MPS discontinuities, and (c) evaluation of the quality of the MPS based upon industry standards. In contrast to rule-based systems this system represents its knowledge as multidimensional vectors and appropriate functions on them.

Currently the expert system accepts manual input of observed features. Once the expert system has been tested and compared with human expert inspectors, a vision front-end will be developed to complete automation of the expert MPS inspection system, based on visual discontinuities. Then the database will be extended to include a variety of other NDT methods. In addition to functional performance, the following objectives were established: ease of use through menu/window-driven input and flexibility in building, using and modifying databases for different applications.

1. Introduction

Materials processing in space [1] offers numerous advantages over processing on earth, particularly due to the absence of gravity. The highest possible level of automation is very desirable in space based manufacturing plants and the automation of one important component of the manufacturing process; quality control (QC) will be discussed here.

Automation of materials QC is generally of significant interest, particularly for economic and ergonomic reasons. Bochove and Somers [2] have described a system which performs specific tests on materials and determines if the results are acceptable or not. A database system which stores test criteria and test results has been presented by Burgade [3] and Svinning [4] uses computers to maintain data on production parameters and corresponding product properties for manual reference. The approach proposed here is an attempt to improve on these systems.

The operation of a typical materials processing plant and its QC components are sketched in figure 1.
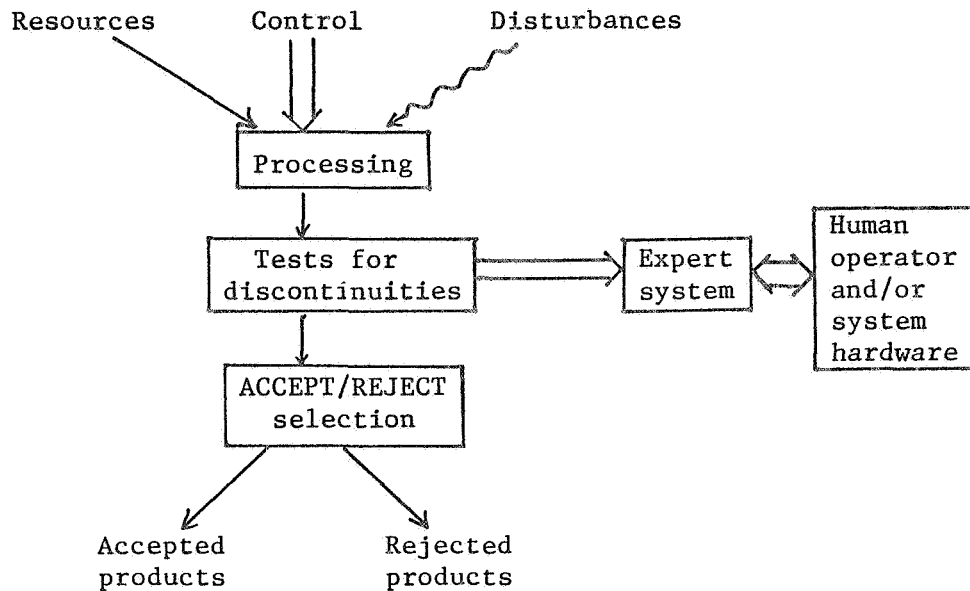
Figure 1. Materials processing.

The required resources (materials, energy, etc.) and appropriate control signals enter the processing unit which in turn delivers the manufactured items. In addition to the required inputs the processing unit may be affected by various "disturbances", i.e. generally unwanted effects which possibly degrade the manufactured items. These disturbances are typically electrical noise, chemical impurities, gravitational jitter, etc. Each produced item is subjected to various tests and the results are submitted to a QC expert system such as the one proposed in this paper. The role of the expert system is to

(i) determine if the item tested should be accepted or rejected;
(ii) in the case of rejection, estimate which disturbances
     contributed to the manufacturing failure.

The estimates of disturbances may be transmitted to human operators or to appropriate control units in the plant for corrections.

Many problems encountered in QC are inherently hard to solve and some even without any final solutions. We suggest certain approximations and heuristic methods to arrive at results where necessary -a common approach for many "artificial intelligence" (AI) methods. Instead of coding the accumulated information of the expert system in terms of "if-then rules" we store it as multidimensional vectors which are retrieved and processed by ad-hoc functions. This is done for the sake of efficiency and furthermore this approach is more in line with the inherent nature of our task.

In the following discussion we will refer to the production of

228

lead-aluminum (Pb-Al) cylinders as an example of materials processing [1],[5]. This particular process is well suited for space operation as gravity causes separation of lead and aluminum when molten on earth.

2. Quality Control Fundamentals

All products are designed so as to have certain properties suitable for their functional performance. Examples for the Pb-Al cylinders include yield strength, fatigue life and external dimensions. Some of these properties can be measured by nondestructive tests (NDT) while others require destructive tests (DT) which generally degrade or destroy the tested item [6]. Degradation of the required properties is usually caused by discontinuities in the items and these discontinuities are in turn generally caused by the disturbance factors. As most discontinuities can be assessed by NDT methods (visual inspection, radiography, acoustic tests, etc.) examination of the discontinuities are frequently preferred over tests of the relevant properties which we want to achieve. The quality control criteria are therefore expressed in terms of discontinuites rather than the desired physical properties.

2.1. Required Properties

Let us assume that the specifications for the Pb-Al cylinders call for certain characteristics for some N property types. For example we would want "the fracture strength to be at least 150 ksi", "the endurance for some cyclic tension test to be at least 10^6 cycles", etc. As these are our unconditional requirements we want the tested item to fulfill all N of these conditions. As a matter of notation let us label the N property types by 1,2,..,N and use $p_{i,t}$ for the measured value (by DT or NDT) of property i. We denote the boundary value for acceptable property i by $p_{i,c}$ ("critical"). If, for example, "fracture strength" is labelled by i=1 then we state the corresponding test by $(p_{1,t} \geq p_{1,c})$ where $p_{1,c}$ is 150ksi or some other required value. For the properties which we prefer to have large numerical values we define $p_i = p_{i,c}/p_{i,t}$ and for preferably low valued properties we set $p_i = p_{i,t}/p_{i,c}$. Thus we can define our criterion for acceptance as

$(p_i \leq 1)$ for i=1,2,..,N.

As the required properties are an ordered set of distinct property values we will represent them as a vector

$\underline{p} = (p_1,p_2,...,p_N)$

composed of the N nonnegative components representing the measured properties.

2.2. Quality Functions

The quality analysis, as set forth above, was based on a strict accept/reject evaluation. We now introduce a gradual measure of the "overall quality" of a tested item. Defining such a measure is not trivial. We

will define it as some scalar function $Q(\underline{p})$ where increased Q in some sense indicates increased quality. By selecting a threhold value Q_0 we can rephrase the acceptance criterion as $Q(\underline{p}) \geq Q_0$. The function $Q(\underline{p})$ should be defined so as to be consistent with commonly accepted quality criteria, such as

$Q(\underline{p})$ has a maximum at $\underline{p}=0$;
$Q(\underline{p}) < Q_0$ iff any component $p_i > 1$;
$Q(\underline{p}) = Q_0$ for boundary cases (where at least some $p_i = 1$
   while all other $p_i < 1$);
$Q(\underline{p}) > Q_0$ iff all $p_i < 1$;
$dQ(\underline{p})/dp_i \leq 0$ for all i.

A simple candidate for such a function is

$$Q(\underline{p}) = 1/\max(p_i) \text{ for } i=1,2,..,N.$$

Figure 2 illustrates this function for $N=2$.
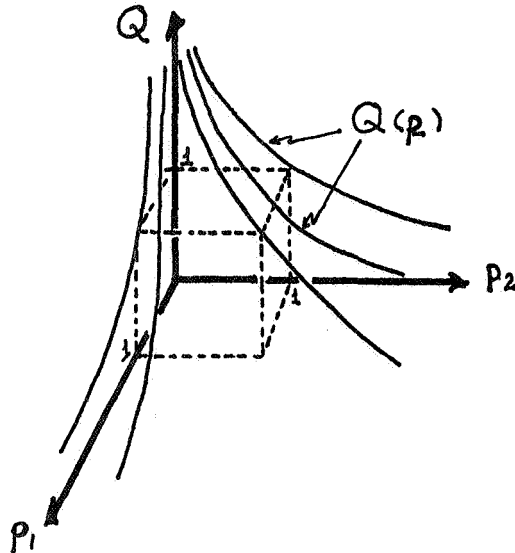
Figure 2. The quality function $Q(\underline{p})$.

Any item will be accepted as long as its $\underline{p}$-vector is within the rectangular volume enclosed by the planes $p_i=0$ and $p_i=1$.

2.3. Discontinuities and Disturbances

Discontinuities (porosity, cracks, etc.) can be measured and assigned values $d_{i,t}$ according to their severities. Specifications for NDT-QC, based on discontinuities, generally state the minimum rejection value $d_{i,c}$ for each discontinuity type i. For M distinct types we are therefore interested in the values $d_i = d_{i,t}/d_{i,c}$ for $i=1,2,..,M$. Each d_i is nonnegative and defined so that decreasing d_i indicates decreasing severity of discontinuity i. As for the property vector $\underline{p}$ we similarly define

$$\underline{d} = (d_1,d_2,...,d_M).$$

230

Likewise we represent the process disturbances (i.e. the causes for discontinuities) by the vector

$$\underline{c} = (c_1, c_2, .., c_L)$$

where the c_i values indicate amounts of chemical impurities, temperature deviations, etc. Unlike the $\underline{p}$- and $\underline{d}$-vectors we do not normalize the components of $\underline{c}$.

2.4. Relations Between The $\underline{c}$, $\underline{d}$ and $\underline{p}$-vectors

By some underlying physical processes (which are generally unknown) any given set of disturbances $\underline{c}$ causes a corresponding set of discontinuities $\underline{d}$ to appear in the produced item. Each $\underline{d}$ in turn affects the final properties of the item and results in a given $\underline{p}$-vector. As these relations are governed by physical laws we expect them to be deterministic. Therefore e.g. repeated occurrences of a given $\underline{c}$ should always result in the same $\underline{d}$. If different $\underline{d}$-vectors are observed for the same $\underline{c}$ we have to conclude that errors have been made somewhere in the observations.

It is fair to assume that these mapping functions are reasonably well behaved in the sense that they and at least their first derivatives are continuous. Although they are generally unknown as mathematical expressions we will approximate these nonlinear mappings at isolated points, based on known points in their neighborhoods.

3. QC - The Traditional Approach

The ususal method of QC based on observed discontinuities is to examine the $d_{i,t}$ values and reject the tested item if any $d_{i,t} > d_{i,c}$ (or equivalently if any $d_i > 1$). In other words, we check if $\underline{d}$ is within the rectangular acceptance boundaries delimited by $d_i = 1$ ($i = 1, 2, .., M$). Clearly this is valid for single isolated discontinuities but we may question this method when applied to concurrently occurring discontinuities. For example we could be tempted to reject an item where several discontinuities occur, if each one just barely remains within the acceptance boundaries. This rarely becomes a problem, however, as real acceptance criteria are usually defined with some factors of safety (this in turn often results in unnecessarily stringent acceptance criteria). In the following section we will see that this concern does not arise in the approach presented there, as it evaluates joint discontinuities in the same way as isolated ones.

4. QC - The Vector Based Approach

A database of measured $\underline{c}$-, $\underline{d}$- and $\underline{p}$-vectors is maintained as triples of vectors for a large number of tested items. It can be updated by inserting or deleting any triple. For a new item to be evaluated we measure its $\underline{d}$-vector by NDT or other appropriate means. Then we select those $\underline{d}$-vectors from the database which most closely resemble the new $\underline{d}$, say a total of K vectors. The $\underline{p}$ for each of the K database items is known and based on that information we estimate $\underline{p}$ for the currently examined item.

Once that $\underline{p}$ has been estimated we evaluate $Q(\underline{p})$ and make the appropriate accept/reject decision.

Optionally, we may also seek to estimate which process disturbances caused the observed $\underline{d}$, i.e. we would want an estimate of $\underline{c}$ for the tested specimen. This can be derived similarly to the $\underline{p}$ estimate. Finally, we may wish to add the current subject to the database. In that case we perform careful tests (NDT and/or DT) to determine the actual $\underline{p}$, we record the observed $\underline{c}$ and insert these vectors with $\underline{d}$ into the database. The entire process is illustrated by the flowchart of figure 3. The estimation of $\underline{p}$ will be discussed below.

4.1. Selecting Database Vectors Closest to $\underline{d}$

The starting point for estimating $\underline{p}$ for the currently examined $\underline{d}$ is to select from the database a set of some K vectors, which adequately resemble $\underline{d}$. Thus we face two questions: what is an appropriate measure of resemblance or similarity, and what should K be?

A measure of similarity could be any scalar function $S(\underline{d}_1,\underline{d}_2)$ which exhibits a reasonable behavior for our sense of similarity. E.g. we would want S to assume a maximum for $\underline{d}_1=\underline{d}_2$ and minimum for all $\underline{d}_1$ and $\underline{d}_2$ which have no common nonzero components. An example of such a function is

$$S(\underline{d}_1,\underline{d}_2) = \underline{d}_1 \cdot \underline{d}_2 \;/\; \max(\|\underline{d}_1\|^2, \|\underline{d}_2\|^2)$$

which ranges from 0 to 1.

For a given $\underline{d}$ we search the database for the $\underline{d}_i$ which yields the highest $S(\underline{d},\underline{d}_i)$, then we find the one giving the second highest S, an so on for up to some K vectors. These K neighbors to $\underline{d}$ will be used to estimate the $\underline{p}$ of the examined item. A simple approximation for $\underline{p}$ is defining it so that it is related to $\underline{p}_1,...,\underline{p}_K$ in the same way as $\underline{d}$ relates to $\underline{d}_1,...,\underline{d}_K$. Specifically, expressing $\underline{d}$ in terms of its database neighbors as $\underline{d} = a_1\underline{d}_1+..+a_K\underline{d}_K$ yields an estimate of $\underline{p}$ as $\underline{p} \simeq a_1\underline{p}_1+..+a_K\underline{p}_K$. Generally we may require a large K to span all nonzero dimensions of $\underline{d}$. In that case it may be wiser to truncate K to a lower value and evaluate the constants $a_1,...,a_K$ so that $\underline{d}' = a_1\underline{d}_1+..+a_K\underline{d}_K$ is at a minimum distance from $\underline{d}$. The truncation could, for example, be set at the $\underline{d}_i$ which yields the first S value below some specified threshold. Finally, as discussed in next section, we must always keep $K \leq M$ where M is the dimension of the $\underline{d}$-space.

4.2. Estimating $\underline{p}$ and $\underline{c}$

The outline of a simple estimation of $\underline{p}$ was introduced above. To maintain, for the estimation of $\underline{p}$, only the $\underline{d}_i$-vectors most similar to $\underline{d}$ and to keep the computational efforts reasonable we choose to keep K within some modest value. Then we evaluate $\underline{A} = (a_1,a_2,...,a_K)'$ so that $\underline{d}' = a_1\underline{d}_1+..+a_K\underline{d}_K$ approximates $\underline{d}$ as closely as possible (actually $\underline{d}' = \underline{d}$ when the selected set of $\underline{d}_i$ spans all nonzero dimensions of $\underline{d}$). This problem has a unique solution for $\underline{A}$ as long as $K \leq M$ and none of the $\underline{d}_i$ are linearly dependent. By differentiating the expression for $\|\underline{d}'-\underline{d}\|$ with respect to each a_i we get K expressions and equate each one to zero. The eventual result is
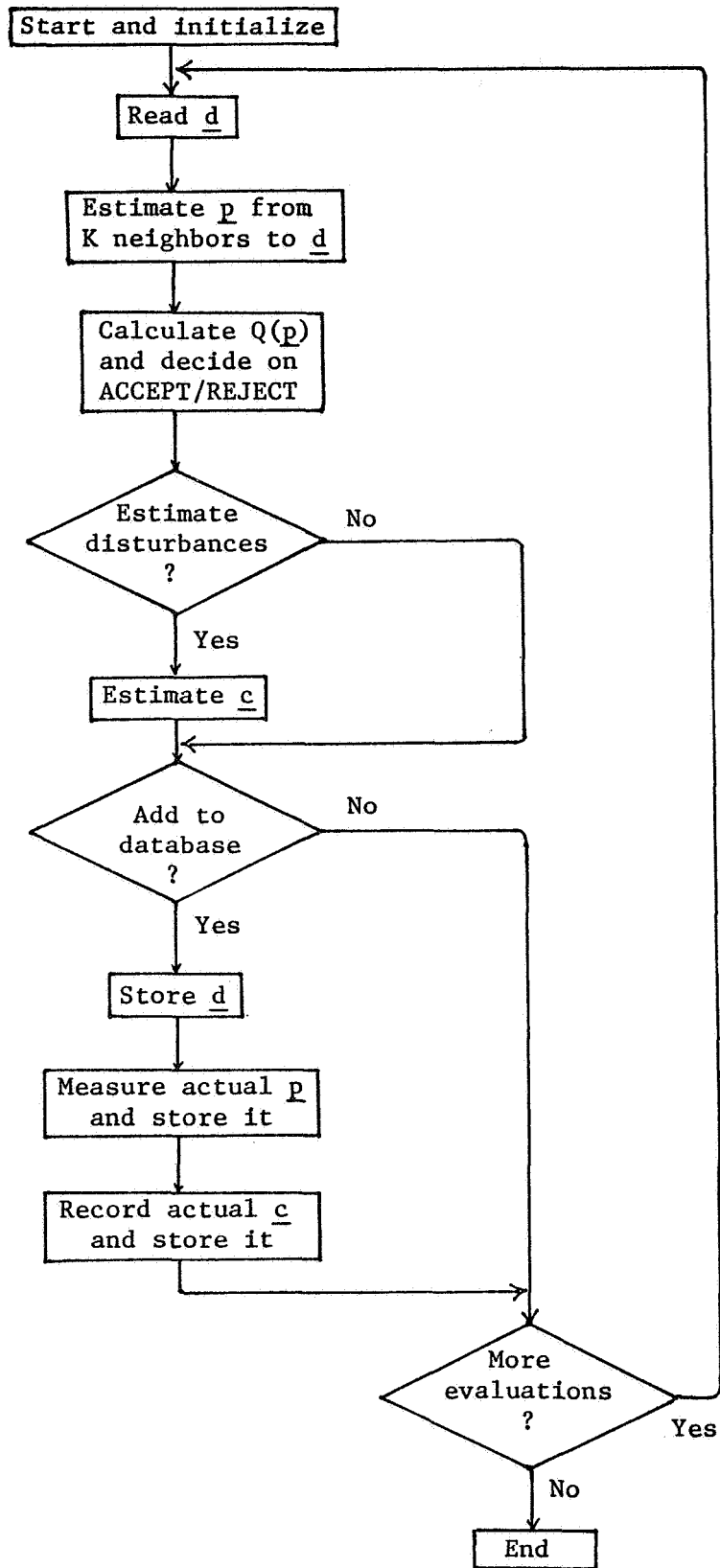
Figure 3. An outline of the expert system.

$$\underline{A} = [\underline{D}\underline{D}']^{-1}\underline{D}\underline{d} \qquad \text{for } K \leq M, \text{ and}$$

$$\underline{A} = [\underline{D}^T]^{-1}\underline{d} \qquad \text{for the special case of } K=M$$

where $\underline{A}$ is the K by 1 vector of the a_i constants and $\underline{d}$ is the (M by 1) column vector of the observed discontinuities. $\underline{D}$ is a (K by M) matrix where the 1st row is $\underline{d}_1$ and so on to the K-th row which is the $d_1, d_2, \ldots, d_M$ values of $\underline{d}_K$. Note that $[\underline{D}\underline{D}']$ is a symmetrical (K by K) matrix, which simplifies the matrix inversion somewhat.

Having obtained $\underline{A} = (a_1, \ldots, a_K)^T$ we estimate $\underline{p}$ accordingly as

$$\underline{p} \simeq a_1\underline{p}_1 + a_2\underline{p}_2 + \ldots + a_K\underline{p}_K.$$

This $\underline{p}$ is then used to evaluate $Q(\underline{p})$ which determines whether the tested item should be accepted or rejected.

Optionally, an estimate for $\underline{c}$ can be obtained too, in the same way as $\underline{p}$, by using the same calculated a_i values:

$$\underline{c} \simeq a_1\underline{c}_1 + \ldots + a_K \underline{c}_K.$$

5. Implementation

At the time of this writing a QC expert system using the vector based methods is in its early test stages. It is written in LISP and runs on IBM-PC compatible computers. Emphasis is placed on ease of use, windows and menus are extensively applied and on-line help is provided at each step. In its current form all information transactions take place via the keyboard and the screen. Future versions may be modified so as to interact with other peripherals, such as discontinuity testing hardware and process controllers.

From the perspective of computational performance the structure of the database is important. Identification of the K "neighbors" to an examined set of discontinuities is a potential time bottleneck for large databases. This problem is basically the "multidimensional nearest neighbor problem" [7] for which a general efficient solution is not known. The current expert system attempts to reduce the search time by keeping a table of the M discontinuity types and pointers from each one to the stored d-vectors where that discontinuity is present. Each $\underline{d}$ has pointers to the corresponding $\underline{p}$ and $\underline{c}$ as well. When the neighbors to a given $\underline{d}$ are to be located, the program uses these pointer to list up only those points of the database which contain any of the nonzero d_i of $\underline{d}$. In other words all discontinuity vectors which have no common nonzero discontinuity types with $\underline{d}$ (i.e. S=0) are left out. This reduces the search space somewhat. The table of discontinuity types serves also as storage for information specific to each discontinuity (allowable numeric ranges, help information, etc.) and we keep similar tables for the property- and disturbance types.

6. Conclusions

In retrospect the methods presented for QC differ from traditional approaches in several ways.

(i) Instead of using QC criteria based on singular discontinuities we attempt to estimate relations between concurrently occurring discontinuities and the corresponding relevant properties. Using these approximate relations we evaluate each collection of discontinuities by estimating and examining how the corresponding properties will be affected.

(ii) The method described provides not only quality evaluation but also estimates of process errors (disturbances) causing degradation of the tested items.

(iii) Building the database of complete individual test results for a large number of items provides a means of storing and retrieving information not otherwise readily accessible.

Although the system has been described here with materials processing in space in mind, it is general enough to handle any QC tasks of nature similar to the materials processing. Thorough tests to evaluate the performance of the system have not yet been undertaken, as it is still in the final construction phase. Only a few simple examples for quality control of welds have been examined and so far they yield satisfactory results. Full evaluation awaits more intensive applications of the system.

References

1. Avduyevski, V.S., <u>Manufacturing in Space</u>, Moscow: MIR, 1985.

2. Bochove, G.F.J. and Somers, P.A.A.M., "Research and Development on
   Automation of Nondestructive Testing Methods".
   <u>Progress in Advanced Materials and Processing</u>.
   Amsterdam: Elsevier Science Publishers B.V., 1985.

3. Burgade, G., "An Application of the Statistics to the Follow-Up of
   the Composite Materials Quality".
   <u>Progress in Advanced Materials and Processing</u>.
   Amsterdam: Elsevier Science Publishers B.V., 1985.

4. Svinning, T., "A Database for Small and Medium-Sized Industries".
   <u>Computers in Materials Technology</u>.
   Oxford, U.K.: Pergamon Press Ltd., 1981.

5. Lichter, B.D., Personal communications.

6. American Society for Testing and Materials (ASTM),
   <u>Annual Book of ASTM Standards, Part 10</u>.
   Philadelphia, Pa.: ASTM, 1974.

7. Bentley, J.L., "Multidimensional Divide-and-Conquer".
   <u>Communications of the ACM</u> (April 1980): 214-229.

A PLAN FOR TIME-PHASED INCORPORATION OF AUTOMATION AND ROBOTICS ON THE U.S. SPACE STATION

R. B. Purves - P.S. Lin - E.M. Fisher Jr.

Boeing Aerospace Co.
Space Station Program
P.O. Box 1470
Huntsville, AL. 35807-3701

ABSTRACT

A plan for the incorporation of Automation and Robotics technology on the space station is presented. The time-phased introduction of twenty-two selected candidates is set forth in accordance with a technology development forecast. Twenty candidates were chosen primarily for their potential to relieve the crew of mundane or dangerous operations and maintenance burdens, thus freeing crew time for mission duties and enhancing safety. Two candidates were chosen based on a potential for increasing the productivity of laboratory experiments and thus directly enhancing the scientific "value" of the space station. A technology assessment for each candidate investigates present state of the art, development timelines including space qualification considerations, and potential for technology transfer to earth applications. Each candidate is evaluated using a crew-workload model driven by crew size, number of pressurized U.S. modules and external payloads, which makes it possible to assess the impact of automation during a growth scenario. Costs for each increment of implementation are estimated and accumulated.

1. Introduction

This paper is a summary of the Boeing Company's input to Data Requirement 17, "Automation and Robotics Plan", which constitutes part of NASA's response to public law 98-371 that mandated the study of advanced automation technology for the space station.

2. Method

The objective of the creation of a comprehensive A&R plan has been accomplished according to the subtask flowchart shown in Figure 1. Initially, 69 A&R candidates were identified for consideration. A set of selection criteria was formulated whose application to the original candidates has produced a reduced set of 22 candidates, one of which, Robot Friday, is

shown in Figure 2. A technology assessment has been performed for each candidate in which expected dates for the availability of the required advancements are estimated. Costs have been estimated using the RCA Price-H model for hardware, and estimates of the number of lines of code and number of rules for software. A crew workload model has been built to evaluate the effects of automation on crew utilization. Based on Skylab experience and driven by crew size, number of pressurized U.S. modules and external payloads, this modelling approach makes it possible to assess the impact of automation during a growth scenario. The on-orbit activity time percentages which drive the workload model are shown in Figure 3 and the workload evaluation methodology is shown in Figure 4. A time-phased implementation plan has been formulated and hook and scar requirement for the IOC station identified.

3. Groundrules and Assumptions

Several key assumptions and groundrules have been set to guide the formation of the implementation plan :

* The IOC station is assumed to be equipped with only "conventional" automation, with Artifical Intelligence and possibly robots to follow.

* Operation and Maintenance Bias - Selection of candidates is biased in favor of Operations and Maintenance over Laboratory Support. This represents our contention that the release of crew time from station-keeping duties in favor of mission duties is the most valuable and urgently required function of A&R.

* Appropriate Goal for the Level of Automation - An appropriate goal for the level and rate of introduction of automation for the space station has been formulated as the approximate amount which would maintain the station-keeping crew workload steady despite growth of the station.

* Autonomous Controller Goal - The plan is formulated with the implementation of an Autonomous Controller as the long term goal.

* Development time curves for non-space specific technologies assume sufficient funding to continue the present rate of development for terrestial use, without the need for additional space-station funding to accelerate development for space use.

* Acceptance of Artificial Intelligence - Because AI programs cannot be rigorously tested due to the non-deterministic nature of their results, we assumed that AI will be accepted rather like human experts are qualified for critical jobs. That is, a skill subset is tested explicitly and then observed over an extended period of increasingly critical correct judgements. Using this approach the implementation sequence for artificial intelligence was governed by the

238

following scenario, with sufficient time at each stage to verify performance and for human acceptance :

A. on the ground, advise only
B. on orbit, observe only
C. on orbit, observe and advise
D. on orbit, observe and control

4. Results

The 22 A&R candidates listed in Table 1 have been recommended for space station implementation. Twenty candidates are in the area of operations and maintenance and two are in laboratory support.

4.1 Operations and Maintenance

The 20 operations and maintenance candidates have been organized into a buildup sequence as shown in Figure 5 in which technology, experience and hardware flow from one candidate to the next. Figure 6 shows the result of inputting the time-phased implementation of the 20 candidates into the crew workload model. It indicates that the station-keeping crewload, which grew unacceptably high assuming a constant Skylab level of automation, could be maintained roughly at five men on the growth station via the given implementation sequence. The band between physical crew size and crew size to operate the station represents the number of people available to contribute to mission activities.

The number of hours and calculated value of the crew-time saved by implementation of the A&R candidates is shown in Figure 7 . According to the figure the operation and maintenance candidates will pay for themselves in terms of crew hours saved in less than 2 years after IOC.

Implementation cost estimates for each increment of A&R are shown in Figure 5 associated with each node on the graph. The figures are the accumulated costs of all increments feeding that node. Yearly cost estimates are shown in Figure 8. The total accumulated cost for the entire plan is $4.44 billion throught the year 2014.

4.2 Laboratory Support

The two selected laboratory support candidates, the experiment monitoring and control system and the laboratory robotic system, are based on pre-existing terrestial hardware and software and are recommended for IOC implementation. Early implementation can cost-effectively increase laboratory productivity with a breakeven point less than two years after IOC, as shown on Figure 9. This assumes launch of a laboratory followed by a one year station buildup (man-tended) phase prior to a continuously manned capability. The increased experiment value provided by A&R has been equated to the cost necessary to achieve the same increased experiment value with additional crew time, assuming a value of $20,000 per on-orbit crew hour.

239

The estimated increment of cost for laboratory support automation above operations and maintenance automation is $55.5 million spread over a 23 year period assuming concurrent implementation of the other candidates. Although the full implementation and operation schedule extends until the year 2014, a significant telescience capability is available at IOC, given six to seven years of prior developement.

4.3 Hardware Scars

Hardware scar requirements in the areas of robotic locomotion, vision and manipulation have been investigated.

For robotic locomotion, the probable scarring of the module interior structure with tracks or rails was identified, assuming that a decision to use this type of locomotion could be made early. Other related items such as attachment fittings, structural strength, recessed switches and navigational beacons were determined to be either adequate at IOC or upgradable.

For robotic vision, items such as bar code labels, shadow defeating lighting schemes and contrasting color treatments were all determined to be adequate at IOC or upgradable; therefore no scars were identified.

For robotic manipulation of connectors, latches, controls, etc., a possible problem area was identified. It was noted that generally a task designed for robotic manipulation is easy for humans also. It is recommended that a "standard" robotic manipulator be defined and that all hardware requiring manipulation tasks be evaluated against this standard to determine the extent of the design changes that might be required to allow robotic manipulation of mundane tasks.

4.4 Software Hooks

Software capability requirements to assure integratability of the A&R candidates should be assured by growth capability built into the IOC station and by proper design methodologies and enforcement of software engineering design discipline.

5.0 Summary

A time-phased program for the implementation of 22 selected A&R candidates has been formulated. A cost and benefit analysis indicates that such an A&R program would pay for itself in terms of crew hours saved and scientific value produced in the lab within 2 years of IOC.
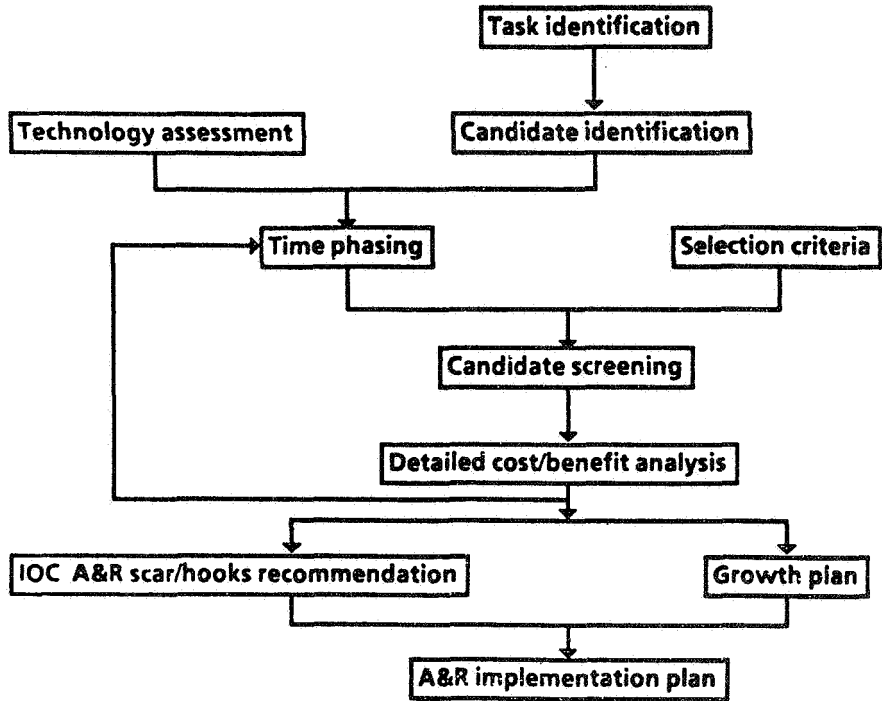
Figure 1. Automation & Robotics Subtask Flowchart
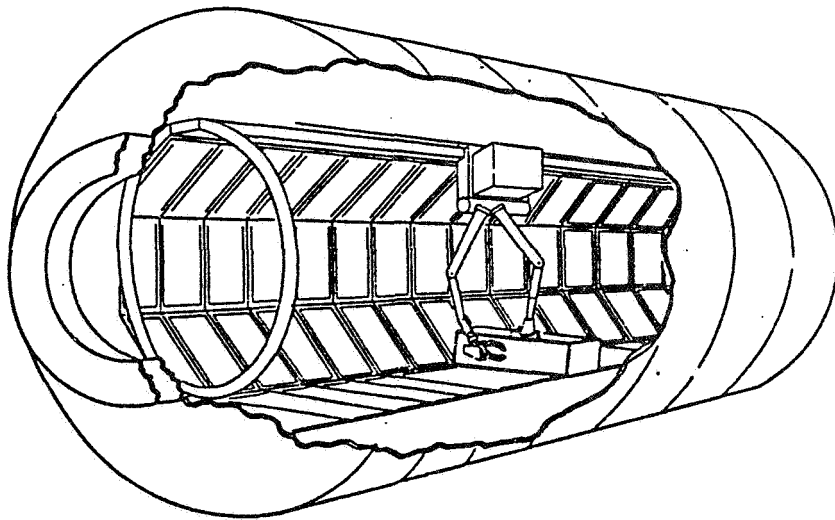
Figure 2. Robot Friday A&R Candidate

Percentages based on total on-orbit time (for one team) 3 crew x24 hours x 90 days = 6,480 hours

Self-sustenance activity
(14 hours per day)
+
days off (12 days)
63.9%

8.8%

17.0%

Mission activity

10.3%

IVA nonmission activity

EVA nonmission activity
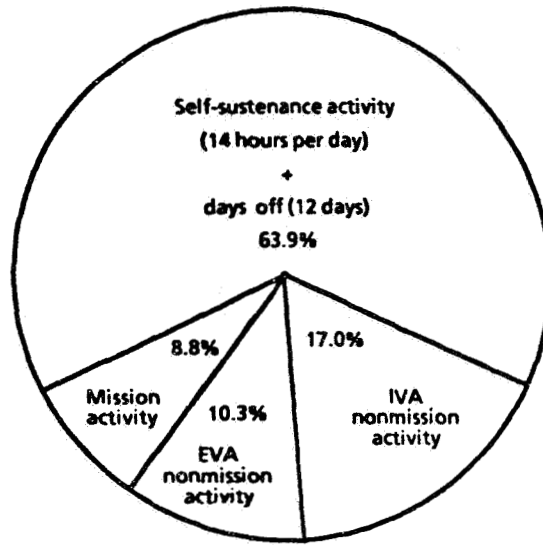
Figure 3. On-Orbit Activity Time Percentages
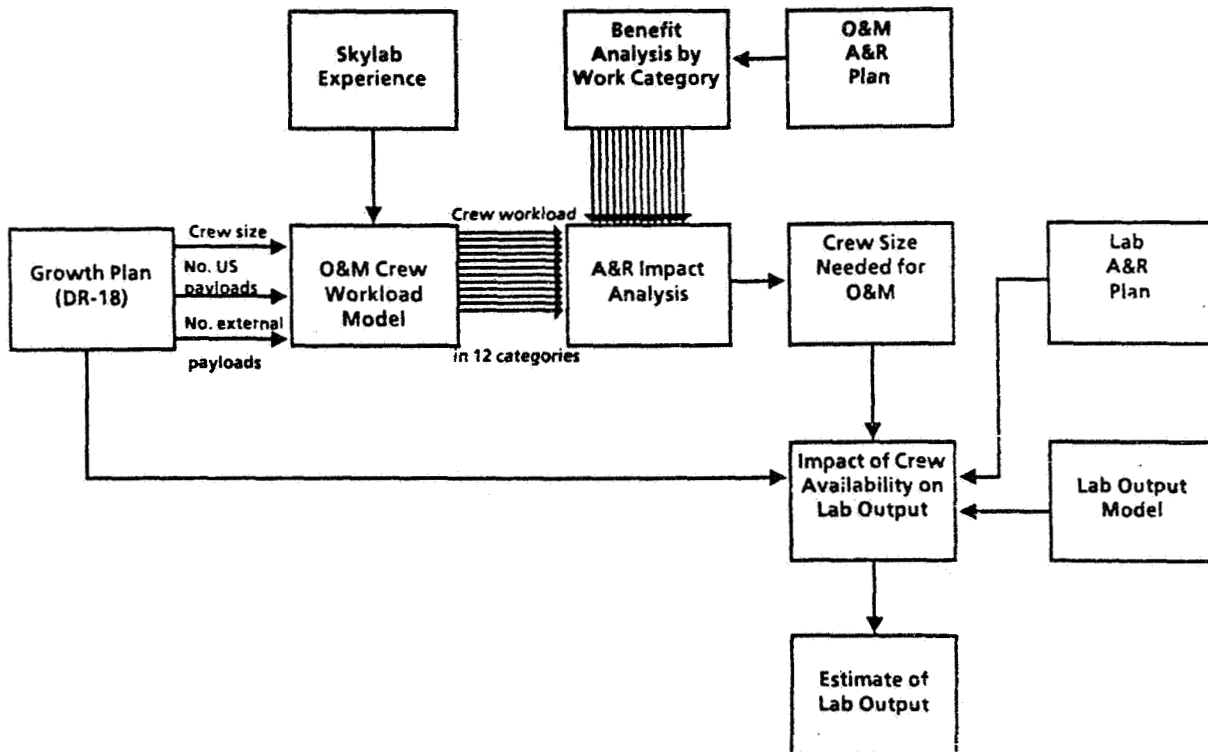
Figure 4. Methodology for Evaluating Crew Utilization of A&R Candidates

Operation and Maintenace

1. Software Application Generator
2. Design Knowledge Database
3. Advanced Conversation Operating System Interface
4. Module Safety Advisor
5. Intelligent Tape Recorder
6. EVA Inspector Robot
7. Manned EVA Teleoperator
8. Smart Front End
9. Integrated Maintenance Trainer
10. Mobile Intelligent Dextrous Robot
11. Autonomous Controller
12. State estimator
13. Subsystem Monitor/Statusing System
14. Fault Prediction/Trend Analysis
15. Fault Manager
16. Intelligent Controller
17. Failure Modes and Effects Analysis Assistant
18. Operation and Maintenance Scheduler
19. Inventory Management System
20. Robot Friday

Laboratory Support

21. MTL Experiment Monitor
22. Laboratory Assistance Robot

Table 1. Selected A&R Candidates
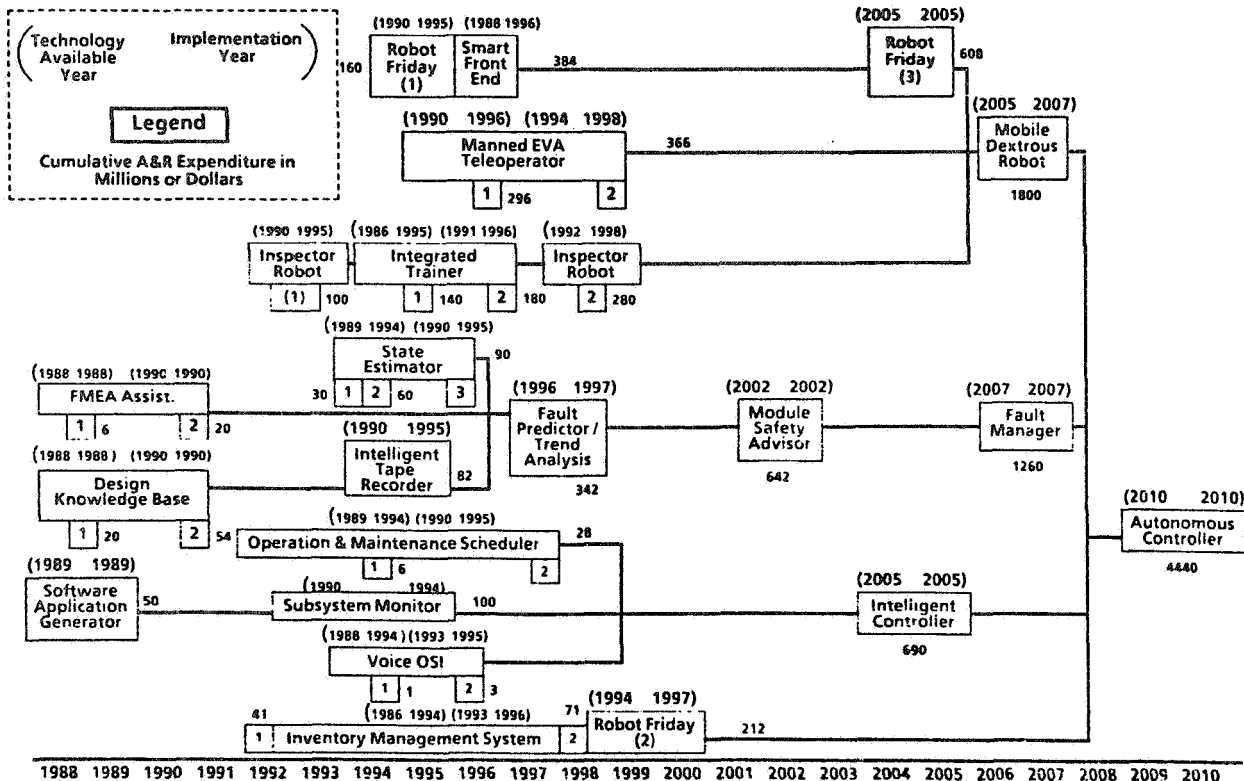
Figure 5. Estimated Life Cycle Cost to Implement O&M Candidates

243
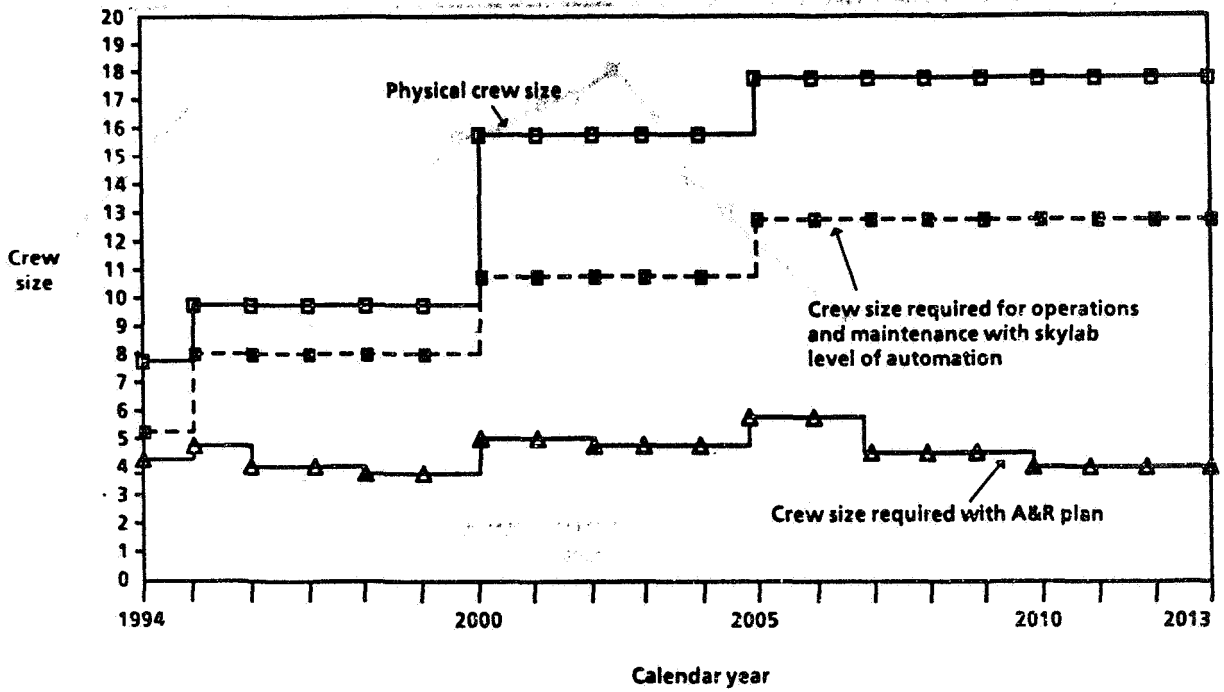
Figure 6. Crew Size Required for Operation and Maintenance with Current A&R Plan.
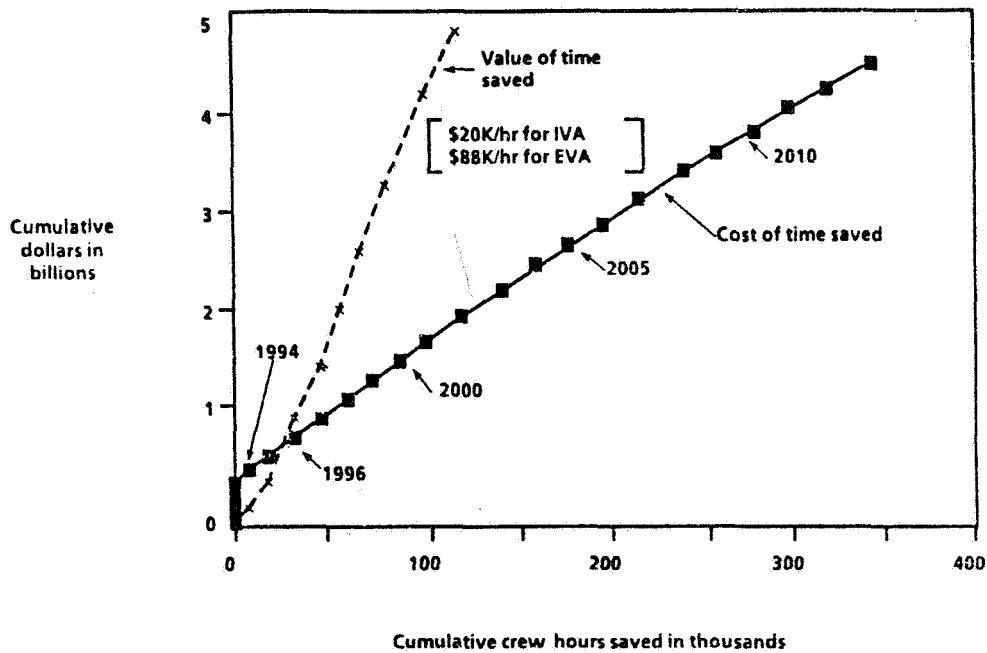
Figure 7. Operations & Maintenance A&R Return on Investment

244
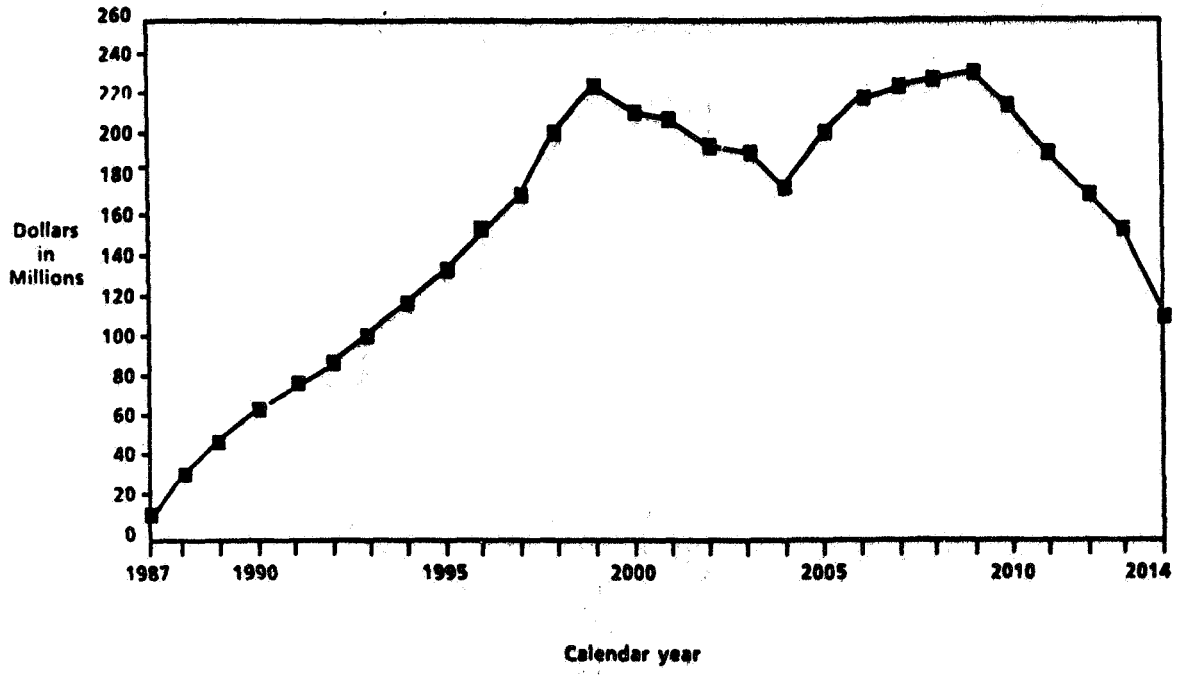
Figure 8. Expenditure by Year for Automation of Operation and Maintenance
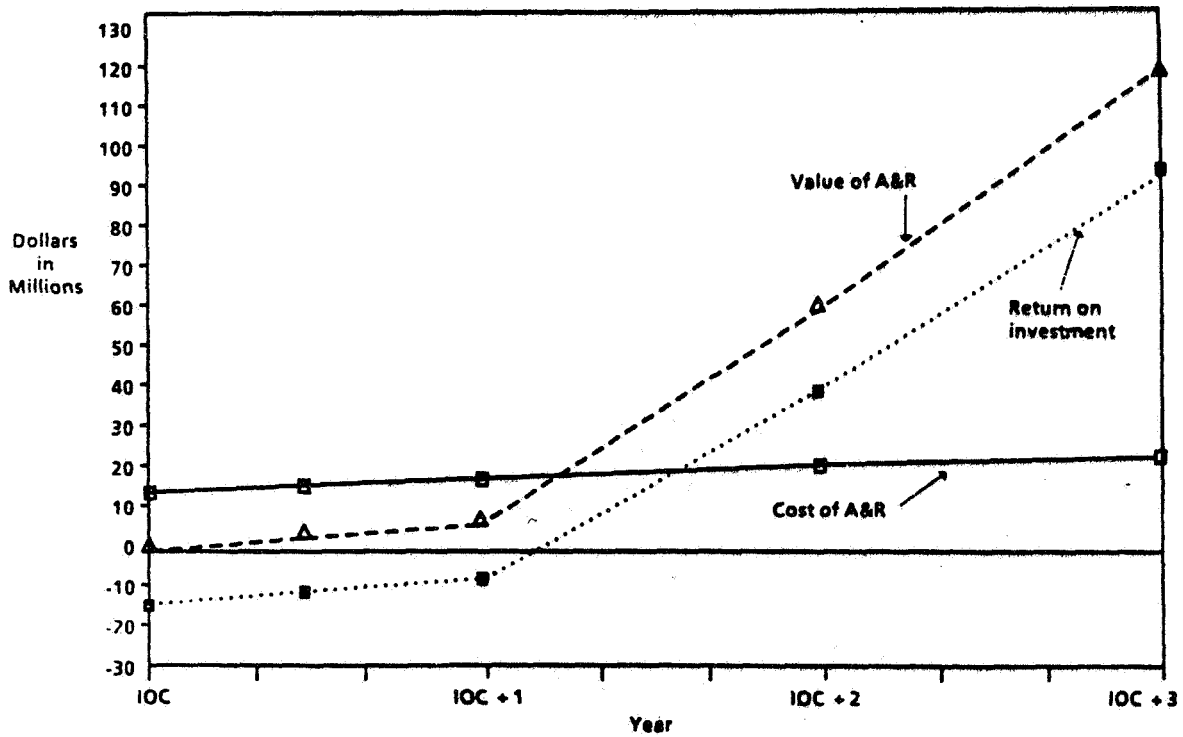
Figure 9. Lab Telescience Return on Investment

MOTION PLANNING FOR A FREE-FLYING ROBOT

Donald Leo Kaiser
Patrick J. Hawkins

Boeing Computer Services
Advanced Technology Center for Computer Sciences
P.O. Box 24346, MS 7L-65
Seattle, WA 98124

ABSTRACT

This paper presents an investigation of motion planning combining low level control and obstacle avoidance for a free-flying robot. This free-flying robot is an outgrowth of the concept of an assistant for astronauts on the U.S. Space Station and Shuttle, suggested by Boeing at a meeting with the NASA Advanced Technology and Automation Committee in 1985 (1). During this meeting, Boeing was encouraged to conduct additional research.

A motion planner based on the Khatib potential field approach is described. Because of the uncluttered environment in space, it generates a path from representation of known obstacles rather than from a representation of free space A global planner supplies the low level controller with interim points between the current position and the desired goal position so that the vehicle does not become trapped by local minima, a phenomenon of the potential field approach. Discussion of the feasibility of this system for space applications is presented.

I. INTRODUCTION

Boeing has been investigating the role semi-autonomous robots will have in space because extra-vehicular activity will increase in frequency and be of longer duration for future space missions. Currently, a simulation (2) exists that allows an operator to vocally guide and teach the robot safe paths around a graphic display of the U.S. Space Shuttle. Figure 1 shows the architecture for the free-flying robot which consists of seven distinct functional blocks: the system blackboard, voice interface, graphics simulation, path planner, command interface, control simulation, and dynamic simulation. The system blackboard provides a central location for all communication, both requests and information deposit/retrieval. The voice interface accepts and coordinates interaction with the user by speech recognition and synthesis. It also validates incoming commands and passes them to the blackboard. A robot viewpoint is given by the graphics simulation, i.e. the computer screen supplies a simulated view from the onboard camera. The path planner determines a path to a previously remembered point. Points along the path in terms of world coordinates are converted to relative coordinates by the command interface. The control simulator determines the necessary thrust commands. Finally, the thrust commands are converted by the dynamics simulator to positions which are sent to the graphics simulation and to the control simulator as feedback. The current operator directed path planner is being changed to a motion planner that incorporates obstacle avoidance and has knowledge of the vehicle's environment. This paper discusses the investigations into the area of motion planning

The motion planning problem involves generating a trajectory from its current to the desired goal position. Several issues for motion planning may be considered:

- The moving vehicle must not collide with stationary or moving obstacles.
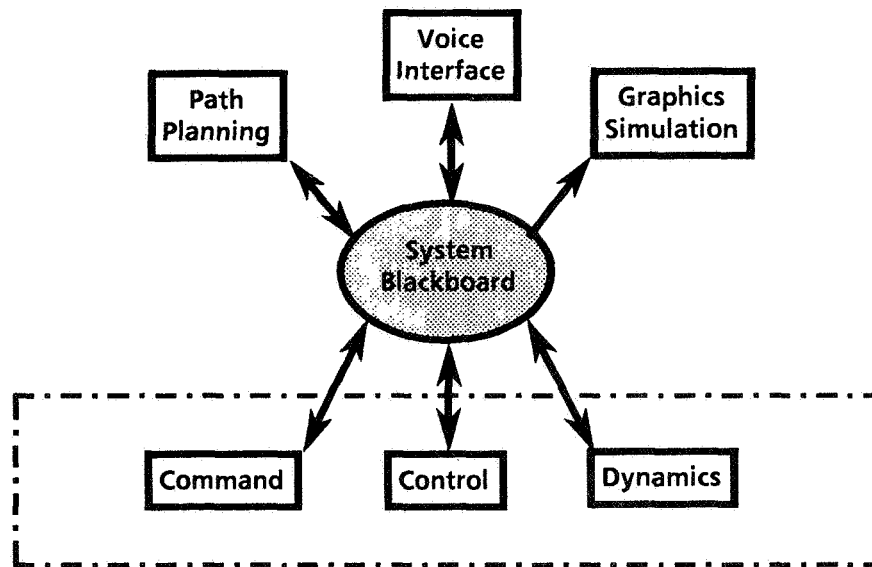
Figure 1. Free-Flying Robot Architecture

- Dimension of environment: 2D versus 3D.
- The geometry of the obstacles, particularly a convex or concave domain.
- The geometric representation of the vehicle.
- Rotation of the vehicle between close obstacles, or around the corners of an obstacle.
- Choosing a path based on a cost function.

Two approaches have been used for motion planning: global and local. A global planner, with information about all known obstacles in the environment, generates an entire collision-free path for a vehicle from its current position to a goal position. If the environment is changed during path execution, the vehicle must stop and the global planner must generate a new path. A local planner focuses on the vehicle's immediate vicinity and generates a path by taking into account observed obstacles as well as the goal position. However, local minima can occur such that the vehicle becomes trapped at certain points in space. Global planners generate entire paths, but typically do so very slowly. Local planners generate, in real time, potentially inefficient paths that are not guaranteed to reach the goal.

The traditional approach to path planning is to divide the continuous free space into adjacent finite states and then search the graph for possible solution states. Whitney (3) is an early example of this combinatorial approach. He used a constant grid mesh to divide a robot manipulator's working space into discrete units. A graph representing the valid states is explicitly stored and searched to generate a path. Whitney experimented with several search techniques, including dynamic programming and the heuristic A* algorithm (4)

Krogh et al. (5) also assigned costs to paths on the grid to determine the "best" path. They commented that typical path planning algorithms evaluate the cost solely on the basis of path length while their method considers how close the path comes to obstacles and penalties for travelling through particular areas, such as unmapped areas. However, Ruff et al. (6) tried several different heuristics, most rather complex, which gave little or no improvement in the number of graph nodes checked and slowed down the A* search, therefore, Ruff et al. returned to a simple distance cost function.

248

Brooks (8) generated a graph using generalized cylinders to represent 2D free space. This representation system was created by Binford for modeling objects for artificial intelligence-based computer vision systems. Brooks also used the A* algorithm with a distance cost function. Brooks (9) later extended the generalized cylinders method to a four degrees of freedom robot manipulation problem with two classes of motion: pick and place, and insertions or fitting.

Ruff et al. (6) extended the grid search method to 3D by decomposing the space into cubes. They showed that deriving the cuboidal representation was the most time consuming part of their algorithm. They gave an example that a 10' x 10' x 6' room represented with 1" resolution would require over 12.5 million cubes.

Khatib (10) presented a fresh approach to 3D motion planning by having the low level controller perform obstacle avoidance. In his approach, obstacles are described by the composition of *primitives* such as a cylinder, cone, or ellipsoid. The obstacles generate artificial potential fields that repel the vehicle, and the goal position generates one that attracts the vehicle. A path is generated in real time by the low level controller, but local minima can occur causing the vehicle to stop before reaching its goal. Local minima are positions where the attractive force of the goal and the repulsive force of the obstacle are equal and opposite.

Krogh (11) introduced *generalized* potential fields, i.e. potential fields which are both position and velocity dependent, and subgoals that are positioned to eliminate the minima problem. However, no method was given for determining those subgoals.

Krogh et al. also used a low level controller in conjunction with the grid method to do motion planning. The grid search generates subgoals that are an optimum path for the low level controller. The local minima are removed by addition of intermediate subgoals at the edge of obstructing obstacles. Determining the *best* set of subgoals was left for future research.

We use a modified version of Khatib's low level controller in conjunction with a global planner. The global planner uses its knowledge of the obstacles to generate a sequence of interim points to the goal that circumvents the local minima problem. We assume the following:

- Obstacles are represented by convex polygons and the free-flying robot by a point. The size of the free-flying robot is taken into account when the potential field of each obstacle is constructed.
- The dimension of the environment is 2D. The free space surrounding the obstacles are referred to as corridors and there are no dead end corridors.
- Obstacles do not touch one another. Potential fields from two separate objects do not overlap. If the range of the potential field is x_0, then the corridor width must be twice x_0.
- The goal point is within an accessible corridor. This is a point-inclusion problem which can be easily determined in 2-space (12).

II. LOW LEVEL CONTROLLER

The low level controller utilizes Khatib's (10) obstacle avoidance techniques to generate actuator signals that direct the vehicle away from obstacles and towards the goal. To accomplish this task, the low level control law must have knowledge of the obstacles to be avoided. Geometric models and locations of the obstacles must be stored in the low level control law memory if no sensors are used; otherwise sensors are used to confirm or determine their shapes and location. The low level control law computes the actuator signal based on the shortest distance to each obstacle. This distance is either computed from stored obstacle models or found with a search routine on the sensed distances.

249

From the control law perspective, the vehicle moves in a field of force with the goal state as an attractive force and the obstacles as repelling forces These forces are generated from artificial potential fields. The potential field of the goal in the x direction is

$$U_{gx} = \tfrac{1}{2}K(x_g - x)^2$$

where K is the position gain, x_g is the distance to the goal from the initial position in the x direction, and x is the distance travelled in the x direction The attractive force is

$$F_{gx} = - \text{grad}(U_{gx}) = K(x_g - x)$$

There are similar equations in the y direction.

This is simply the proportional position control law. The potential field of each obstacle is

$$
\begin{aligned}
U_0 &= \tfrac{1}{2}[(1/(p-p_0)) \cdot (1/p_0)]^2 && \text{if } p \le p_0 \\
&= 0 && \text{if } p < 0
\end{aligned}
$$

where p_0 represents the range of the potential field influence, $(p-p_0)$ is the shortest distance to the obstacles from the vehicle's position p, and N is the potential field strength The repulsive force due to the obstacle's potential field is

$$
\begin{aligned}
F_0 &= - \text{grad}(U_0) \\
&= N(\tfrac{1}{2}[(1/(p-p_0)) \cdot (1/p_0)](1/(p-p_0))^2(dp/dx)) && \text{if } p \le p_0 \\
&= 0 && \text{if } p < 0
\end{aligned}
$$

where **dp/dx** is the partial derivative of the distance from the vehicle to the obstacle,

$$\mathbf{dp/dx} = [dp/dx, dp/dy, dp/dz]^T.$$

In addition, the control law contains a damping term, $K_v(dx/dt)$ where K_v is the velocity gain and (dx/dt) is the velocity vector. The servomechanism is velocity limited such that the position error is set to K_vV_{max}, when $K_p(x_g-x)$ is greater than K_vV_{max} where V_{max} is the commanded maximum velocity .

III. SOLUTION OF THE LOW LEVEL CONTROLLER'S LOCAL MINIMA PROBLEM

An obstacle's potential field is limited to a given region surrounding the obstacle to avoid undesirable perturbing forces beyond its vicinity (10) Therefore, the only time that a minimum point can occur is when the vehicle is *within the range* of the obstacle's potential field and the goal's attractive force is equal and opposite to the obstacle's repelling force. Essentially, the obstacle is between the vehicle and the goal The two cases when local minima occur are (Figure 2):

- The obstacle exerts a repulsive force (F_0) on the vehicle in *one* direction which balances the attractive force (F_g) of the goal in the *opposite* direction.
- The obstacle exerts repulsive forces (F_{0x}, F_{0y}) on the vehicle in *more than one* direction which balance the goal's attractive forces (F_{gx}, F_{gy})

Both cases can occur in a concave domain, but only the first pertains to a convex domain under our assumption that potential fields from obstacles cannot overlap.

Two approaches to solving the local minima problem are:

- Locate the minima points and avoid them

250

$$F_g = -F_0$$

$$F_{gx} = -F_{0x}$$
$$F_{gy} = -F_{0y}$$
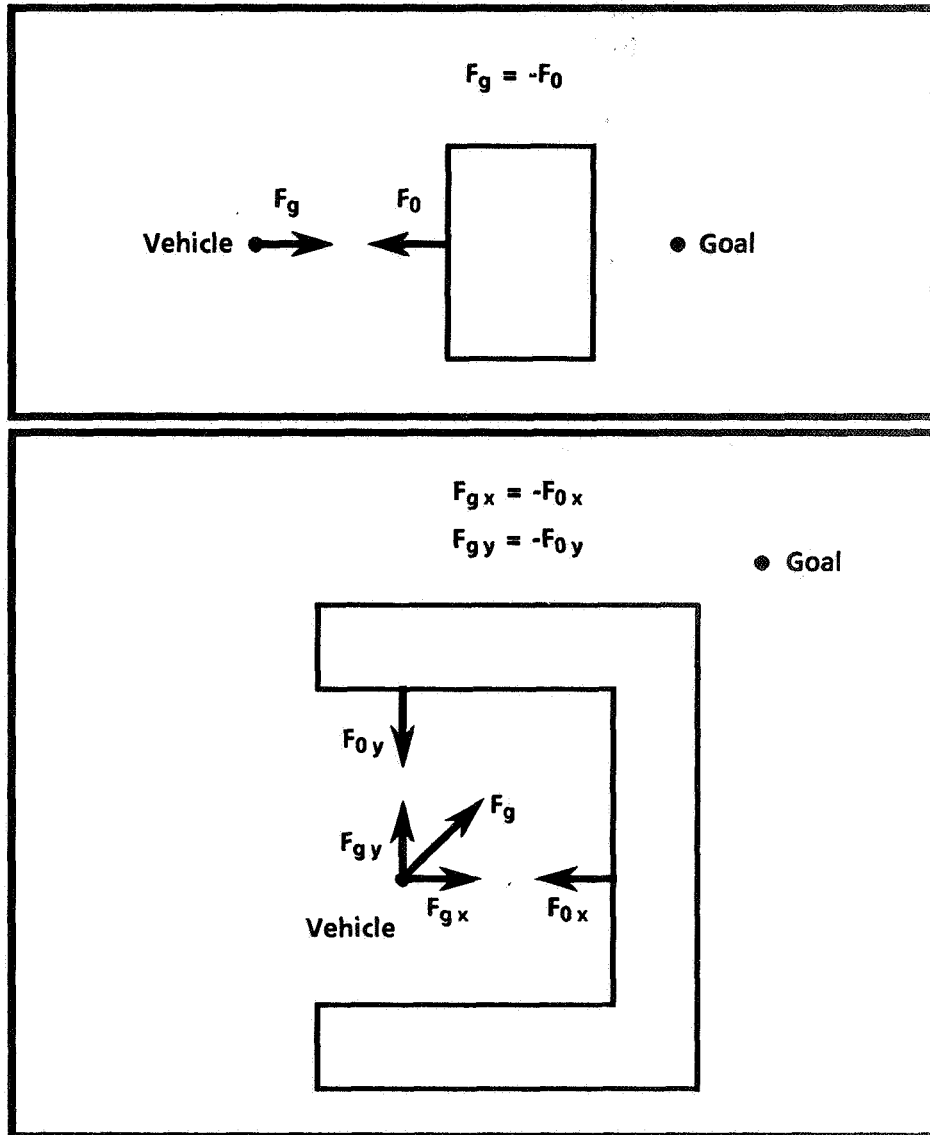
Figure 2. Two Cases When Local Minima Occur for Potential Field Approach

● Provide a sequence of interim goal points to the desired goal point that prevents creation of local minima.

A wide range of geometric models can be selected for the obstacles. Because we represent the obstacles by convex polygons, simple equations can be used for the potential field forces. However, due to the uncertainty of the vehicle's trajectory while avoiding obstacles, local minima cannot be easily predicted. Therefore, the first approach is too computationally expensive for a real time control system (10) and becomes intractable in real time if the obstacles are moving. We implemented the second approach combining the low level controller with a global planner.

IV. DESCRIPTION OF GLOBAL PLANNER

This implementation generates a sequence of interim points such that at any given interim point both the previous point and the next are visible. Figure 3 shows a sample path created by the global
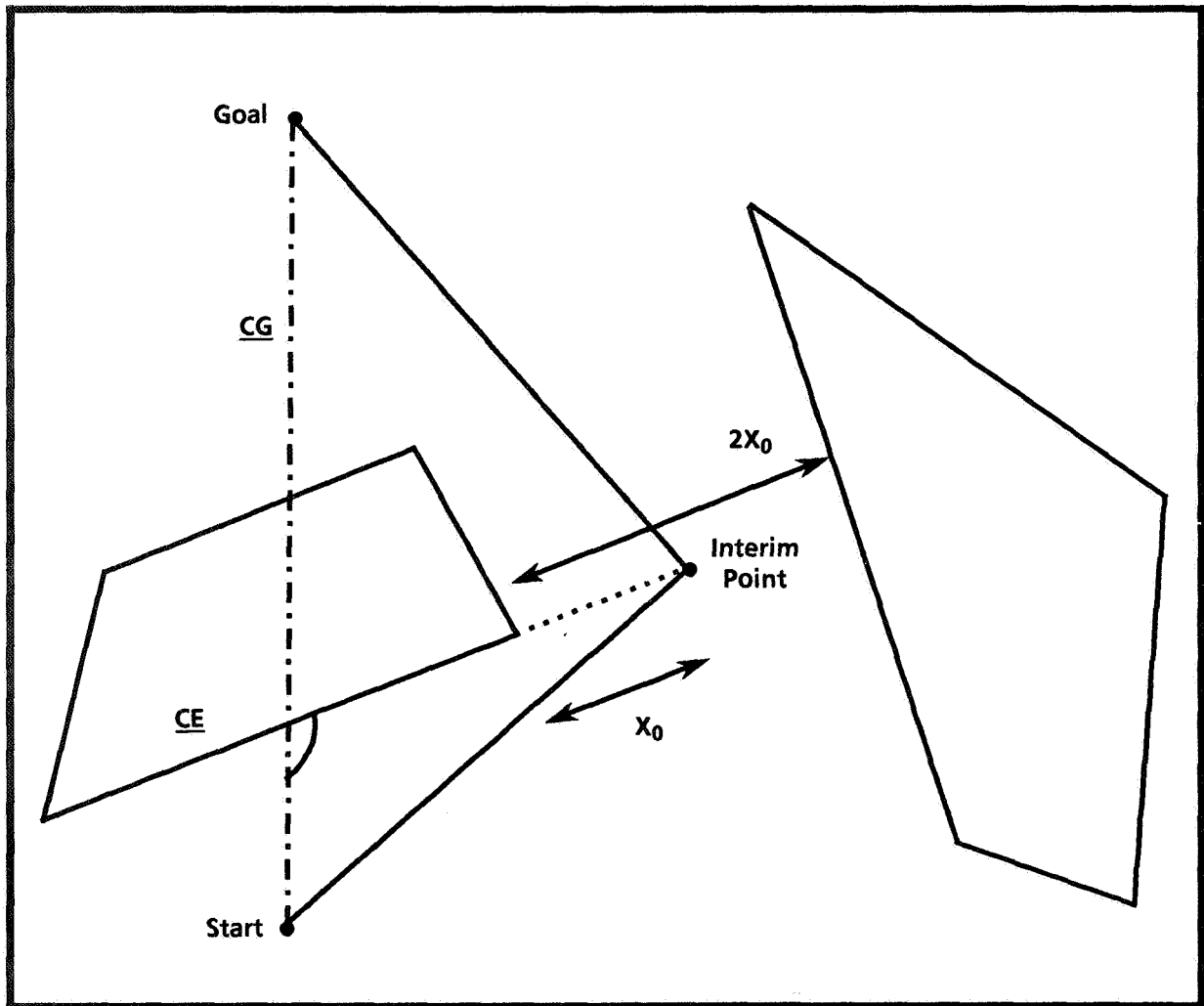
Figure 3. Global Planner Generated Path

planner. The recursive algorithm below is called initially using the current position, the desired goal position, and the empty path list as parameters. It produces a list of interim points in reverse order needed to go from the start to the goal. Note that if there are no obstacle edges that intersect $\underline{CG}$, the goal is visible and the current level of the algorithm ends.

Planner (current-start, current-goal, path-list)
- Find equation of line segment (CG) from current-start to current-goal.
- If there are any obstacle edges that intersect CG then:
  - Find the closest edge (CE) to current-start and the intersection point
  - Identify the obtuse angle that CG makes with CE from the current-start and identify CE 's corresponding endpoint
  - Place an interim point on the line containing CE a distance x_0 away from the endpoint
  - Planner (current-start, interim-point, path-list)
  - Push interim-point onto path-list
  - Planner (interim-point, current-goal, path-list)

where D_{max} is the largest width of the vehicle, i.e. the largest distance from the vehicle's center of mass to its perimeter.

V. APPLICABILITY OF METHOD TO A SPACE ENVIRONMENT

To consider the vehicle a point mass, the range of influence of the potential field is increased by the largest width, D_{max}, of the vehicle. In addition to this distance, the region of influence must contain the distance the vehicle will penetrate the potential field before reversing and moving away from the obstacle. This distance is a function of the maximum velocity of the vehicle, as shown in Appendix I. Typical design values for a free-flying robot are shown in Table 1. From these values the

Table 1. Free-Flying Robot Parameters

Given		
Mass of Robot (M)	250	kilograms
Maximum width of Robot (D_{max})	1.1	meters
Maximum Velocity of Robot (V_{max})	0.5	meters/second
Computed		
Range of Potential Field (x_0)	1.51	meters
Strength of Potential Field (N)	1000	watt-seconds meters^2

range of the potential field is 1.51 meters using a potential field strength of 1000 watt-seconds meters^2 based on the equation developed in Appendix I. The maximum actuator force of 208 newtons is developed using the potential field strength.

The space environment is well suited to this approach of motion planning The vehicle's environment is known and uncluttered, lending itself to a path planning system whose representation focuses on the obstacles rather than on the entire free space. By surrounding the actual obstacle shapes with convex polygonal envelopes for the purpose of working with a simpler representation, an adequate margin of safety will be built into the system. Furthermore, the vehicle will be closest to the polygonal envelope only when it is approaching an edge perpendicularly.

A severe limitation with the current global planner is that no cost function is implemented to allow comparison of different paths. It may position the interim points quite inefficiently depending on the configuration of the obstacles. The space environment would necessitate incorporating a cost function into the approach; fuel limitations could then be considered and sensitive areas where experiments were being conducted or photographs being taken could be avoided. Currently, the choice of the side of the obstacle on which the interim point is placed is arbitrary. Future work is

suggested toward considering the obstacle size and geometry in this choice. Use of a visibility graph may provide a tractable approach. Use of interim points to avoid local minima in a concave domain is also under investigation. Future work will also include examining motion planning in a 3D environment.

VI. ACKNOWLEDGEMENTS

REFERENCES

(1) Advanced Technology Advisory Committee (ATAC), April 1, 1985, Executive Overview, NASA TM 87566.

(2) Hammond, R.A. and Skillman, T.L., "Simulation of a Free-Flying Inspection Robot," Second Annual Workshop on Robotics and Expert Systems, pp. 37-44, June, 1986.

(3) Whitney, D.E., "State Space Models of Remote Manipulation Tasks", IEEE Transactions on Automatic Control, Vol. AC-14, No 6, pp 617-623, December, 1969.

(4) Hart, P., Nilsson, N.J., and Raphael, B., "A Formal Basis for the Heuristic Determination of Minimum Cost Paths", IEEE Transactions of System Science and Cybernetics, Vol. SSC-4, No. 2, pp. 100-107, July, 1968.

(5) Krogh, B.H. and Thorpe, C.E., "Integrated Path Planning and Dynamic Steering Control for Autonomous Vehicles," IEEE International Conference on Robotics and Automation, Vol. 3, pp. 1664-1669, April, 1986.

(6) Ruff, R. and Ahuja, N., "Path Planning in a Three Dimensional Environment", Seventh International Conference on Pattern Recognition, Vol. 1, pp. 188-191, 1984

(7) Lozano-Perez, T. and Wesley, M.A., "An Algorithm for Planning Collision-FreePaths Among Polyhedral Obstacles", Communications of the ACM, Vol. 22, No. 10,pp. 560-570, October 1979.

(8) Brooks, R.A., "Solving the Find-Path Problem by Good Representation of Free Space", IEEE Transactions of System Science and Cybernetics,Vol. SMC-13, No. 2, pp. 190-197, March/April 1983.

(9) Brooks, R.A., "Planning Collision Free Motions for Pick and Place Operations", M.I.T. A.I. Memo. 725, May 1983.

(10) Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," 1985 IEEE Conference on Robotics and Automation, pp. 500-505, March 1985.

(11) Krogh, B.H., "A Generalized Potential Field Approach to Obstacle Avoidance Control," Robotics International Robotics Research Conference, August 1984.

(12) Preparata, F.P. and Shamos, M.I., *Computational Geometry*, Springer-Verlag Inc., New York, New York, 1985.

APPENDIX I. DERIVATION OF THE RANGE OF THE POTENTIAL FIELD

The range (x_0) is the maximum distance of the potential field. Beyond this distance the potential field is zero; hence the low level controller does not generate obstacle avoidance commands. We assume the following:

- Worst case conditions for developing x_0
- The vehicle is moving perpendicularly towards a surface at a maximum velocity, V_{max}.
- The actuator system is linear

The potential field has the form

$$\text{P.E} = \tfrac{1}{2}N[(1/(x_0 - x)) - (1/x_0)]^2 \qquad \text{for } 0 < x < x_0$$
$$= 0 \qquad \text{for } 0 < x$$

where N is the potential field strength constant and x is the distance traveled.

Since the system is conservative, x_0 can be determined based on conservation of energy. The vehicle's kinetic energy as it enters the potential field is

$$K.E. = \tfrac{1}{2}MV^2_{max}$$

where M is the vehicle's mass.

All of the kinetic energy must be converted to potential energy at a safe distance from the obstacle. Hence,

$$x = x_0 - D_{max}$$

where D_{max} is the maximum width of the vehicle. Substitution of the previous equation into the potential energy equation gives

$$P.E. = \tfrac{1}{2}N[(1/D_{max}) - (1/x_0)]^2$$

Application of the conservation of energy gives

$$K.E. = P.E.$$

and

$$\tfrac{1}{2}MV^2 = \tfrac{1}{2}N[(1/D_{max}) - (1/x_0)]^2$$

There are two solutions to this equation:

$$x_0 = D/(1 \pm [M/N]^{\frac{1}{2}}V_{max}D_{max})$$

The solution with the positive sign is discarded because the penetration distance x of this solution is negative. For negative x, the potential field is defined as zero. Hence,

$$x_0 = D/(1 - [M/N]^{\frac{1}{2}}V_{max}D_{max})$$

OMV Docking Simulator

by

W. Teoh, Ph.D.
University of Alabama in Huntsville
Huntsville, Alabama   35899

and

J. Hawkins
Boeing Aerospace Company
Huntsville, Alabama   35805

ABSTRACT

The Boeing OMV Docking and Proximity Operation System (DAPOS) has been completed. The system constructed involves the use of four separate processors. Appropriate software is developed that drives each of these four processors.

The hand controller logic coordinates all the activities in the control station, and communicates with the OMV mathematical model. The state vector generated by the model is in turn transmitted to the control station as well as the POLY 2000 (via the ALCYON host computer) for real time graphics generation.

The OMV characteristics are stored in a data file which may be easily updated and modified without disturbing the software, thereby making the system very flexible.

The current system supports two types of hand controllers. A programmable touch panel (PTP) is being integrated to the system as a follow-up work to the current contract.

The system has been flown by several volunteers some of whom are airplane pilots. A user manual is also enclosed. Data is collected and will be presented.

INTRODUCTION

The paper summarizes the results of a joint effort between UAH and Boeing Aerospace to develop an OMV simulator to investigate docking of the OMV with the Space Station. The OMV Docking and Proximity Operation Simulator (DAPOS) developed by UAH encompasses the following elements:

a) A mathematical model of the OMV

b) A graphical model simulating the scene as recorded by an on-board video camera, and

c) Control station logic.

A special purpose process (POLY 2000) is used for graphics generation and display, but conventional, general purpose computers are used for both the mathematical model (microVAX) and the hand controller logic (IBM AT).

DAPOS -- SYSTEM ORGANIZATION

DAPOS is made up of several hardware and software components. A POLY 2000 (SCM) is used for the generating and updating of the graphics. The software is a graphical model that simulates the scene as seen by the OMV on-board, forward looking camera.

The scene must be constantly updated, depending on the state of the OMV. Thus, a mathematical model of the OMV is required. The model resides in a computer (microVAX). Lastly, the mathematical model accepts input from a hand control station which is itself made up of several hardware and software components. To coordinate these activities in the control station, a third computer (IMB AT microcomputer) is used to process the signals and communicate with the mathematical model.

The intended system architecture is as Shown in Figure 1. Here, the intended host computer is the ALCYON that is attached to the SCM via a modified Q-bus. After the hand controller logic was installed, plans were made to implement the OMV model to the ALCYON computer. A quick benchmark

was carried out, and it was found that this computer is not fast enough the execute the model at the desired updating rate. A suggestion was forwarded to implement both the geometric model and the OMV mathematical model in the SCM, but this suggestion was rejected on the grounds that:

a) There are no performance figures to support the fact that the SCM can handle both tasks and still be able to update at the desired rate,

b) The SCM may not have sufficient memory to accommodate both tasks,

c) The SCM is not equipped with a floating point accelerator and the mathematical model uses almost exclusively double precision floating point arithmetic, and

d) It is the desire of Boeing Aerospace to implement the system quickly so that there is sufficient time to further investigate the numberical processing capability of the POLYT 2000.

Based on these reasons, a microVAX equipped with a floating point accelerator is used to host the mathematical mode. To date, the system architecture is shown in Figure 2. In the present implementation, four separate processors are involved, and the processors are interconnected by means of data links. As can be seen in this diagram, the joysticks (used as hand controllers) communicate with the IBM AT via a hard-wired Q-bus. The AT communicates with the microVAX bi-directionally using an RS232 link.

The state vector of the OMV computed by the microVAX is transmitted to the ALCYON communicates with the SCM over the modified Q-bus. Figure 3 summarizes the logical flow of information between these processors.

260

CONTROL STATION

The control station is made up of several pieces of hardware and software components. An IBM AT is used to coordinate all its activities and to communicate with the OMV model in the microVAX.

Two types of hand controllers are attached to the IBM AT; a pair of 3 DOF joysticks and a single 6 DOF hand controller made by CAE of Canada. The latter will be discussed in a separate report, and in the balance of the present report, the term ˜hand controller˜ will be used to mean the pair of 3 DOF joysticks. For clarity, the two joysticks will be called joystick-0 and joystick-1 respectively. Each joystick provides three analog and two digital outputs, and each joystick is connected to the appropriate connectors of a TECMAR LAB TENDER data acquisition board installed on the mother-board of the IBM AT, and this board can accomodate both analog and digital signals. 8-bit ADC˜s are used for the present application. This board is also equipped with three parallel ports named A, B and C providing a total of 24 lines of digital I/O. In the present work, all three ports are programmed via software for input, and only four out of the 24 channels are used.

As pointed out earlier, each joystick has two switches (digital output) in addition to its three potentiometers (analog output). These are normally open swithces, and in the present work are programmed as toggle switches. Each switch has been assigned a unique function as shown in Figure 4. It is important to remember that these are toggle switches, and that is is necessary to press the switch again to restore it to the original state.

The hand controller logic is implemented in software called BBHC. This software is physically found in a sub-directory called BOEING on the hard disk (drive C:). Figure 5 shows the high level flow diagram of BBHC.

An auto-pilot logic is also implanted in the hard controller software. When the auto-pilot is engaged, the analog signals from the hand controller are read but not processed. Instead, the auto-pilot algorithm employs a scheme using the state vector of the OMV at that given mement to determine the first 12 bits of the control word. The four switches are, however, still active. The strategy employed by the auto-pilot algorithm is one that tends to minimize flight time by using a number of short straight path segments to approximate the flight path of the OMV. As implemented, the auto-pilot may be engaged and disengaged at will be toggling the AUTO switch located on the joystick. The hand controller logic is written in TURBO PASCAL.

OMV MATHEMATICAL MODEL

In order to construct the model of the Orbital Maneuvering Vehicle, the following assumption are made:

1. The OMV is assumed to be a circular disk of uniform mass distribution. In the present application, the detail shape of the OMV is unimportant as long as the mass and propulsion characteristics of the Orbital Maneuvering Vehicle are known. In the present model, the mass and thrust characteristics are taken from the MSFC Preliminary Definition Studies.

2. The OMV is manipulated using signals derived from the hand controller. These signals can be classified into two groups. The first group is used to simulate a force acting through the center of mass of the OMV. In other words, one can, from this group of signals, generate an acceleration vector $\underline{a} = [a_1, a_2, a_3]$ in the body frame. The other group of signals simulates rotations about 1, 2 and 3 axes, namely, a vector $\underline{w} = [w_1, w_2, w_3]$. Assumptions 1 and 2 mean than detailed

262

knowledge of the shape, thrust level and placement of the thruster and so forth are not really needed.

THE GEOMETRIC MODEL

The geometric model emulates the scene as seen by the on-board, forward looking video camera. This model must satisfy the following three requirements:

1) It must be able to generate realistic graphics display in the vicinity of the space station, including the docking probe

2) The scene sequence must be dictated by the motion of the OMV, so that a trained operator can "sense" the OMV motion, and

3) The updating rate must be sufficiently high so that a smooth animation effect can be obtained.

In the present implementation, the entire system is updated at a rate of eight times per second (real time), even though the designed update rate is ten times per second. At the current updating rate, smooth animation is obtained. In order that the position and orientation data may be used by the geometric model, both the SCM graphics processor and its host (the ALCYON) must be used. These two processors are linked together by means of a modified Q-bus that permits information exhange between the two processors using the file server.

A short program called COP3.C written in C exists in the ALCYON which repeatedly reads these data from the mathematical model, scales them suitable and places them in the Q-bus. This is necessary since these quantities are real numbers, and must be scaled and converted to integers before being transmitted over the Q-bus. Specifically, all angular quantities must be expressed in BAM´s.

To generate the necessary animated graphics, an existing program called FLY.C provided by GTI is modified and used. This software is

compiled, and uploaded to the SCM for execution after it is suitably modified to:

- include a short algorithm to intercept data from the Q-bus, and use this data to suitably update the scene sequence, and

- incorporate the geometric model of the space station with a docking probe developed by John Holmes at Boeing Huntsville. This model yields a sufficiently realistic display of the Space Station.

This software loads the appropriate set of picture files from disk and these files constitute the visual data base. These are the files used to generate the graphics, and as the visual model is improved, the picutre files in this data base may be replaced by the lastest version without having to disturb the source code. This represents a very efficient method of implementing the visual model.

The picture files are developed by John Holmes at Boeing Aerospace Company.

PRELIMINARY RESULTS

DAPOS is now on-line and operational. Since four separate processors are used in this system, the start-up procedure is necessarily somewhat complicated, and all efforts have been made to make the process easier.

At present, both the 3 DOF joystick pair and the single 6 DOF hand controller are connected and operational, although only one set of hand controllers may be used at a time. The axes of both hand controllers are chosen in an appropriate manner.

The auto-pilot, though not an important part of the current scope of work, functions well unless the OMV is tumbling very badly, (with the attitude in rate control) due to severe cross-coupling. In general, the auto-pilot can successfully dock the OMV much faster than a human operator,

at the expense of more fuel consumed.

HUMAN FUTURE STUDIES

The primary objective in this project is to determine whether there is any objectively measurable superiority of either a single six degree-of-freedom hand controller or a pair of three degree-of-freedom hand controllers for the task of OMV control. There is no published data that has been identified that addresses this objective comparison, although there is some anecdotal evident that suggests the superiority of the 6 DOF controller.

During the information-gathering phase of this work, personal discussions were held with Tom Bryan, who supervises the operation of the telerobotics lab facility at MSFC. He related an experience in which they had used both a single 6 DOF controller (an engineering prototype on loan from CAE) and a pair of 3 DOF controllers for control of a robotic arm by a human operator. The arm being used had a small "drift" which caused it to appear not to be following the operator's commands exactly. When the heman operator tried to overcome the drift using the pair of 3 DOF controllers, the task was a very difficult one which some operators were never able to perform satisfactorily. However, when using the single 6 ODF controller, correction for the drift was an easy task which was almost intuitive and could be accomplished with practically no training required.

Discussions were also held with Mike Kink of CAE of Canada, the firm which manufactures the 6 DOF controller. He related experience with the multi-axis hand controller in which the task of flying a high-performance helicopter had been placed under the control of a CAE hand controller. The controller replaced the three controls routinely used in the flight control of rotary wing aircraft (cyclic, collective ,and the pedlas). The comments from the test pilots and their experience in the flight test indicted that

the new controller was far superior to the more traditional configuration of flight controls, and that the training required to satisfactorily fly the helicopter with the multi-axis hand controller was substantially reduced from more traditional approaches.

The apporach taken to evalutate this question was to have human operators perform a selected segment of an OMV mission, using both kinds of controllers, and to compare their perofrmance. It was anticipated that the dependent variable selected would allow the identification of any superiority that may exist for either control configuration.

TEST SUBJECTS.

The subjects used for this test were all inexperienced in the actual task of controlling an OMV. Two Skylab asstronauts were used as subjects, as well as six non-austronaut personnel chosen randomly to participate in the test. It is believed that eight subject provide sufficient data for a determination to be made regarding the hypothesis of no difference between the control treatments, since the experimental design is within subjects design and since there are only two levels of the independent variable and no interaction effects to be delt with.

TASK

The task performed by the subjects in this study is a simple one of flying the OMV visual model with the controls of the MPAC Element Demonstrator. Rather than duplicate a lengthy section of descriptive text, the subject is asked to refer to the instructions to the subject. A visual representation of the task is included as Figure 6.

EVALUATION APPROACH

Since there is a single independent variable, the type of hand controller(s), this is a simple two level, within subjects experimental design. The null hypothesis or hypothesis of no difference, identified as

266

HO, is that there is no difference between the two levels of hand controllers. The research hypothersis, indentified as Hl, is that there is a significant difference in the subjects' performance of the experimental task, and that the difference will favor the six degree-of-freedom controller.

It is likely that there is a learning effect on the subjects, and it is also likely that there would be positive transfer of this learning from whichever controller is used first to the controller used second. Therefore, it was necessary to use a counterbalanced order of presentation to the eight subjects to control for this order effect. Thus, four subjects used the single controller first and used the pair of controllers second, while the other four subjects used the pair of controllers first and then switched to the single controller. The instructions to every subject were the same.

DEPENDENT MEASURES

Evaluation measures are known as dependent variable in the design of experiments on the performance of man-machine interface combinations. The dependent variable selected to be measured for this research project are whether or not the docking is successful, the time it takes for the subject to successfully or unsuccessfully dock the OMV, and the total fuel expended while making the attempt.

DATA ANALYSIS

The raw data for all the subjects is shown in Table I. The data collapsed across all eight subjects is shown in a successful docking, so that dependent variable was dropped from further consideration.

The analysis of variance yielded F ratios that were not significant at the .05 level.

RESULTS

The results indicate that, within the limited conditions of this study, there is no significant difference between a single 6 DOF controller or a pair of 3 DOF controllers for the task of OMV control. Comments from the participants, however, clearly showed a majority subjective preference (7 of 8) for the 6 DOF controller.

The objective of this effort was to determine whether there is any objectively measurable superiority of either 3 DOF of 6 DOF controllers for the task of piloting the OMV. The results of our data collection seem to indicte that there is not any such difference, at least not under the conditions of this experiment. Therefore, it is possible that we will have to look further for reasons to choose between controller configurations for this vitally important task.

The intital bias of this investigator was that the single 6 DOF controller would prove to be superior to the pair of 3 DOF controllers for the OMV control task. However, this did not turn out to be true. It is likely that one of themain reasons is that the task of flying the OMV is, in reality, not very similar to the task of flying an airplane or helicopter. The control inputs are typically small and infrequent, and the response of the vehicle to those inputs is not rapid, but actually rather languid in most axes of motion. The precision of the control seems to be of more importance than whether of not control inputs to all the exes of motion can be input with one hand. Further study is planned along the lines of clearly describing the key parameters for controls for OMV and other free-flyer proximity operations.

268

CONTROLLERS

	6 DOF			3 DOF	
SUBJECT	TIME USED	FUEL (% LEFT)		TIME USED	FUEL (% LEFT)
S1	934.4	95.39		1211.8	92.92
S2	408.9	97.00		313.0	97.03
S3	267.7	94.21		313.8	94.36
S4	238.6	94.95		247.6	95.09
S5	267.8	90.84		311.3	90.28
S6	419.8	92.26		386.3	90.58
S7	359.2	97.16		365.2	96.38
S8	298.1	97.99		306.1	97.92
Mean	399.3	94.98		431.9	94.32
SD	211.9	2.32		297.3	2.68

TABLE I. RAW DATA FOR ALL SUBJECTS

269

AVERAGE FUEL CONSUMPTION COMPARISON

6 DOF CONTROLLER   3 DOF CONTROLLER
    5.02%   5.68%

AVERAGE DOCKING TIME COMPARISON

6 DOF CONTROLLER   3 DOF CONTROLLER
    399.3 sec.   431.9 sec

[Comparison with one anomalous data set (S1) deleted]

6 DOF CONTROLLER   3 DOF CONTROLLER

FUEL   5.08%   FUEL   5.48%
TIME   322.9 sec   TIME   320.5 sec

TABLE II.   COMPARISON OF RESULTS ON DEPENDENT VARIABLES

C-4.
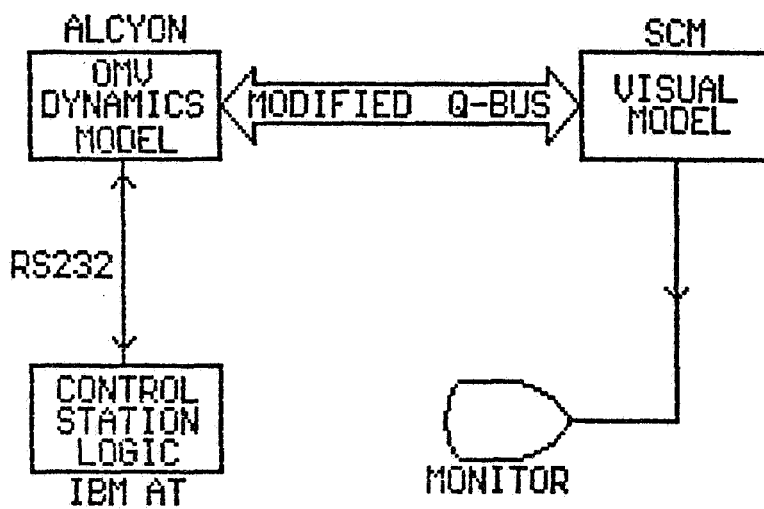
Figure 1

Proposed System Architecture
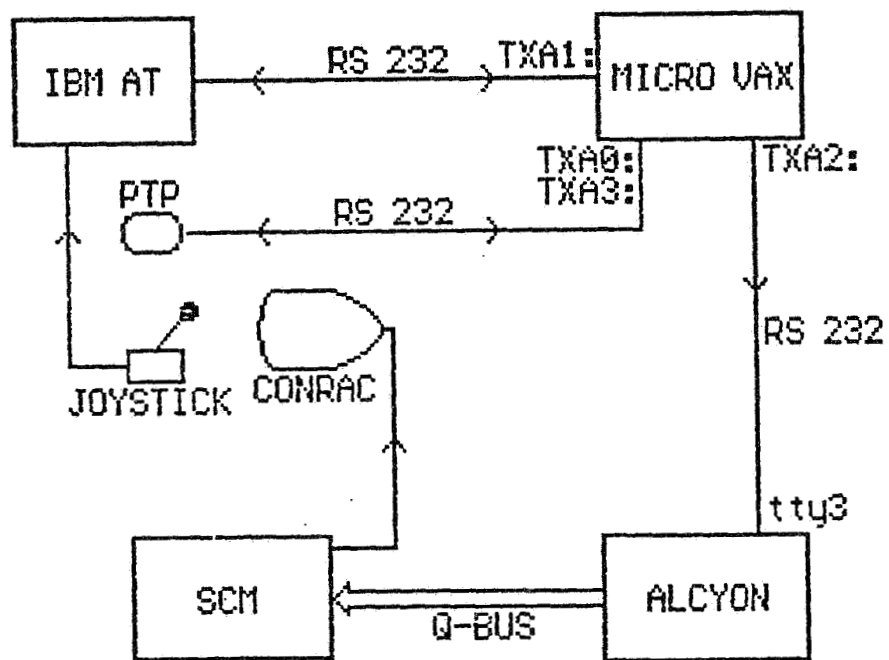
Figure 2

DAPOS -- System Architecture
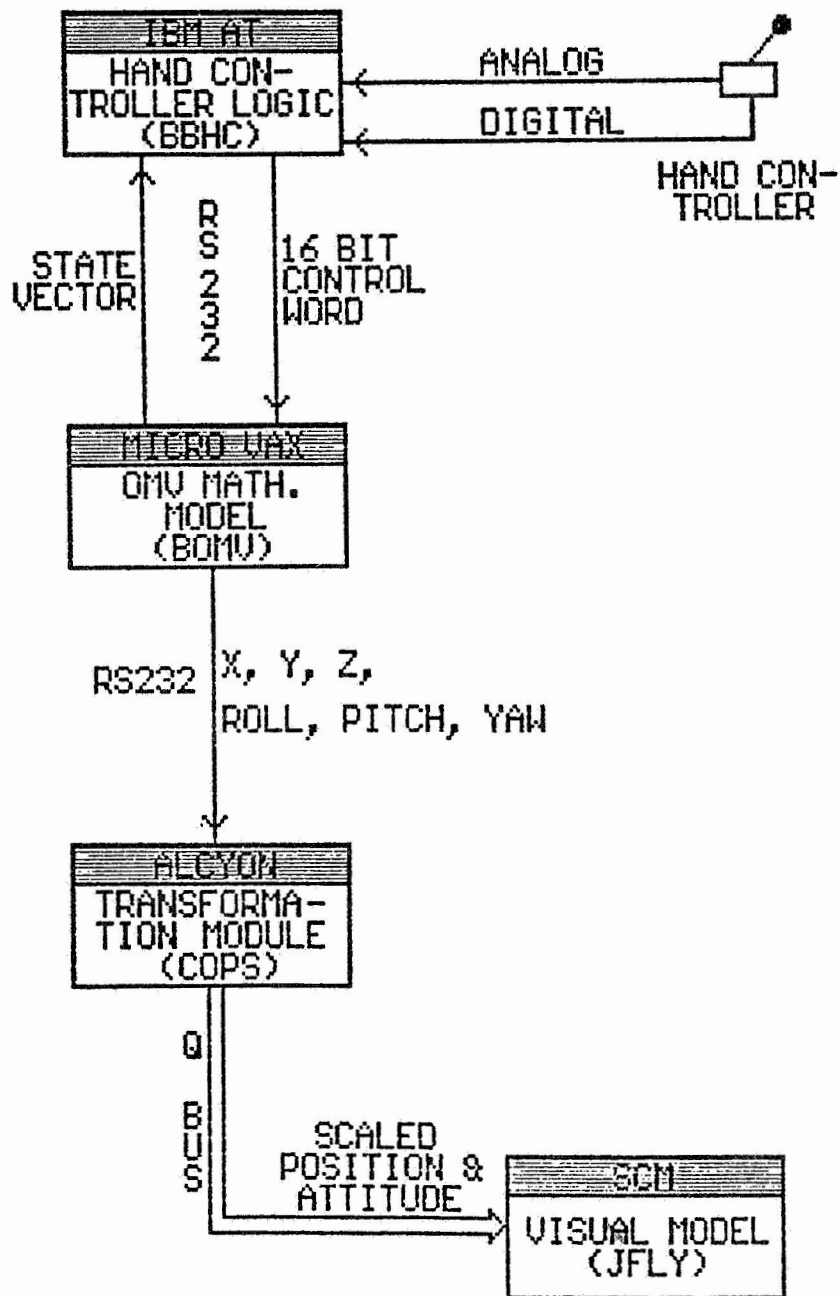
Figure 3

DAPOS -- Information Flow

switch	channel number	functional assignment	meaning
0	0	ABORT	terminate current run
1	1	AUTO	enable/disable auto-pilot
2	2	BRAKE	toggle brake, causing all motion to cease
3	3	MAIN	selects main engines or cold gas thruster for pro-pulsion.

Figure 4

Switch / Function Assignment
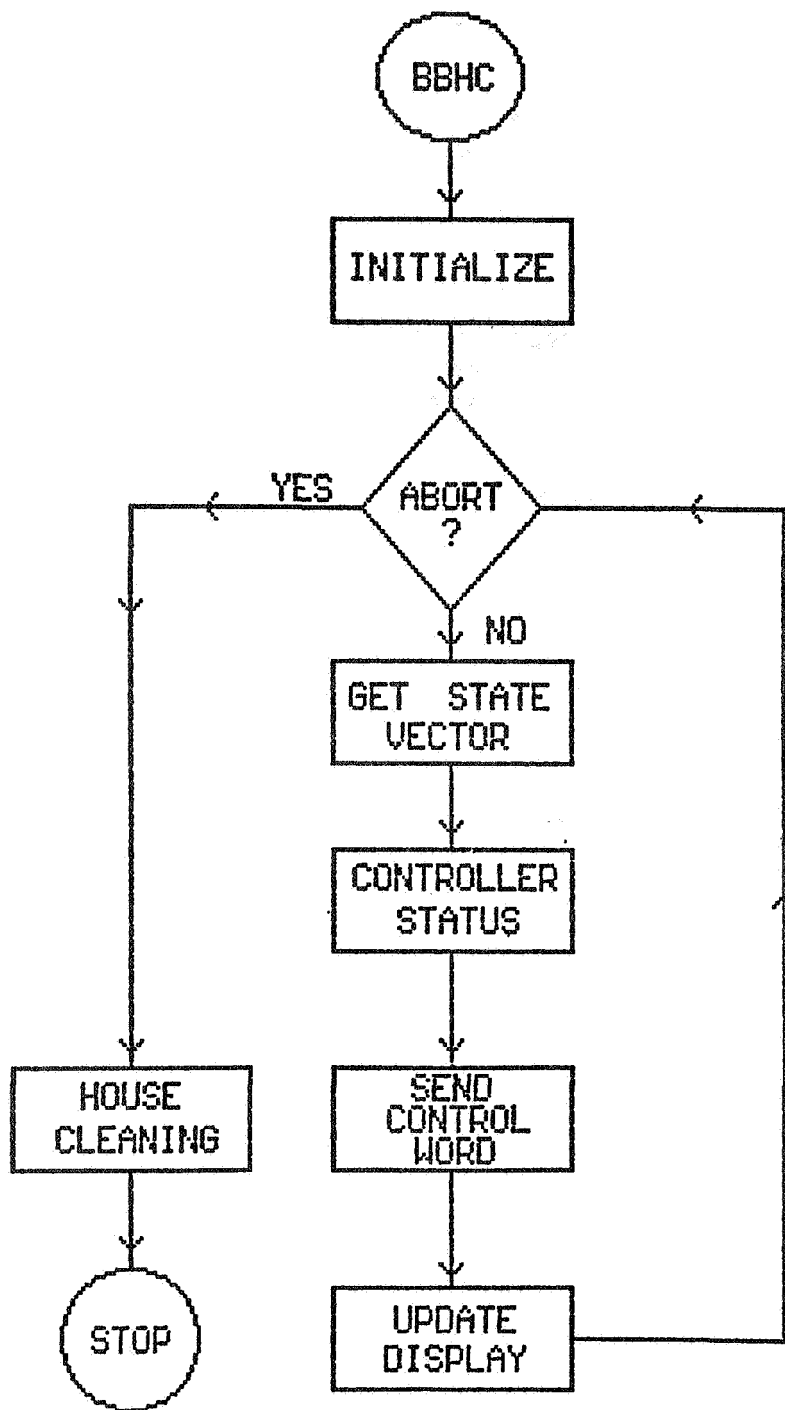
Figure 5

BBHC -- Hand Controller Logic

275

+Z

+Y

TRY TO FLY
"BACK" TO STATION

+X

DIRECTION
OF FLIGHT
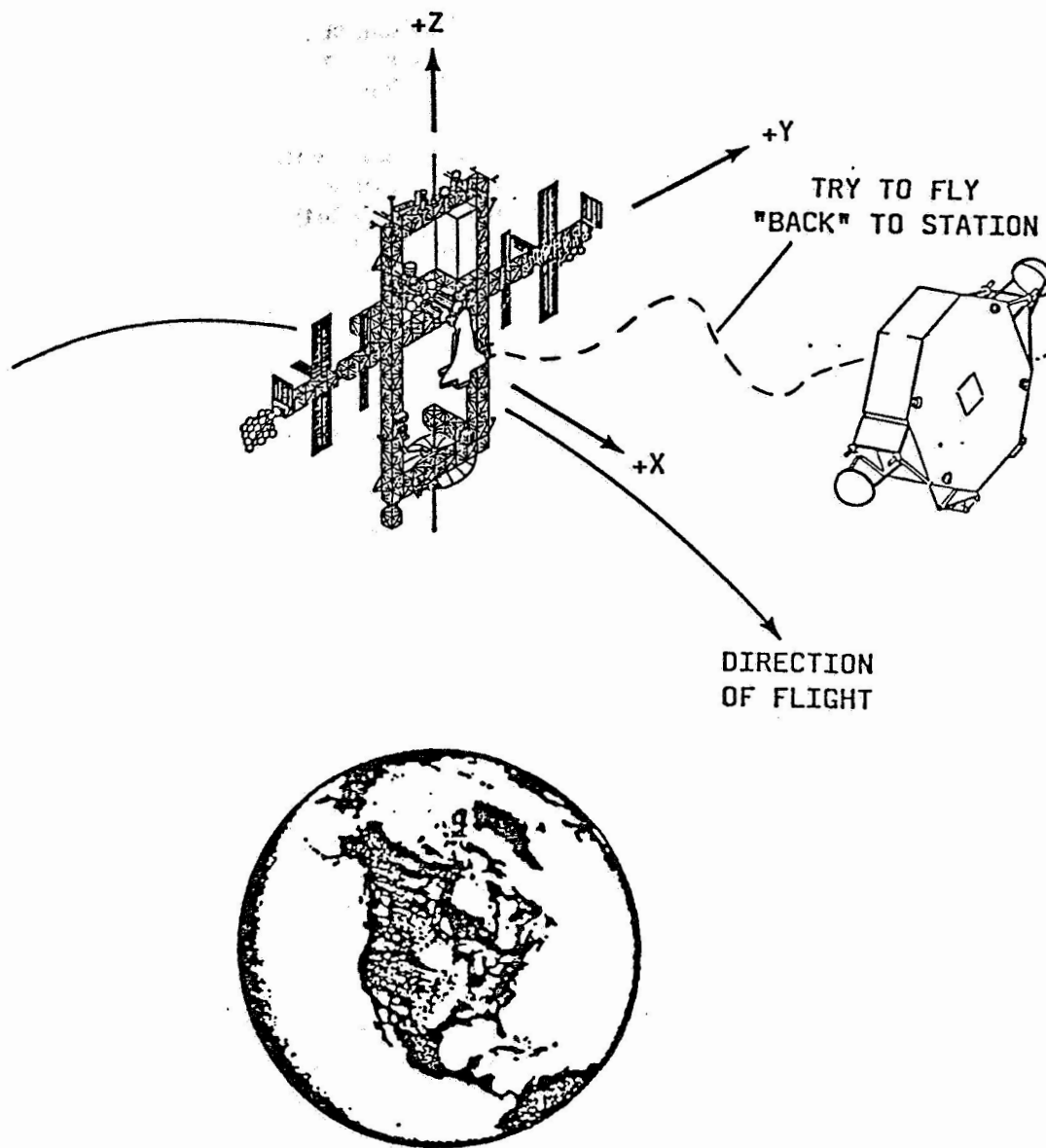
Figure 6

Task Visual Representation

ARGES: An Expert System for Fault Diagnosis within Space-Based ECLS Systems

David W. Pachura
Salem A. Suleiman
Andrew P. Mendler

Martin Marietta Denver Aerospace
Space Station Program
P. O. Box 179 (MS D1744)
Denver, CO 80201

Abstract

ARGES (the Atmosphere Revitalization Group Expert System) is a demonstration prototype expert system for fault management for the Solid Amine, Water Desorbed (SAWD) CO_2 removal assembly, aassociated with the Environmental Control and Life Support (ECLS) System. ARGES monitors and reduces data in real time from either the SAWD controller or a simulation of the SAWD assembly. It can detect gradual degradations or predict failures. This allows graceful shutdown and scheduled maintenance, which reduces crew maintenance overhead. Status and fault information is presented in a user interface that simulates what would be seen by a crewperson. The user interface employs animated color graphics and an object-oriented approach to provide detailed status information, fault identification, and explanation of reasoning in a rapidly assimilated manner. In addition, ARGES recommends possible courses of action for predicted and actual faults. ARGES is seen as a forerunner of AI-based fault managment systems for manned space systems .

Introduction

ARGES (the Atmosphere Revitalization Group Expert System) is the result of an independent research and development project (D-47s) at Martin Marietta to demonstrate the application of artificial intelligence to fault diagnosis and management in space-based Environmental Control and Life Support ECLS systems. The work was performed in conjuction with Hamilton Standard, Inc ,Windsor Locks ,Conn., who provided expert engineering and design knowledge regarding operations of the ECLS system assembly hardware/software. The goal was to show an increased flexibility and function within this task, providing greater assistance to the crew and reducing the need for ground-based support. The first phase of the development was the design and implementation of a prototype that performs fault detection and isolation and demonstrates the user interaction and interface with the overall management software. In this paper, we discuss some of the significant features of ARGES, the architecture and current state of the system, and conclude with some areas of future work.

Approach

ARGES is an expert system for fault diagnosis of the Solid Amine Water Desorbed (SAWD) CO2 Removal Assembly. It is a prototype for demonstrating the applicability of AI/Expert Systems technology to space-based ECLS systems and its function as part of the overall management software system. With that goal in mind, the resulting system departs from other work previously done in the ECLS system area [Dickey84, Lance85]. Its most important features are:

1. It is a prototype of an expert system which functions as part of the control and management software for the ECLSS; not as an isolated system which communicates with a user, but as an embedded program, which uses data received directly from the hardware. It interprets the data, detects a fault, and reaches a conclusion without human intervention. Any interaction between the user and the program occurs at the user's option and convenience, as a means of verifying the conclusion, not as an aid to the program's diagnosis.

2. The user interface is designed to simulate what would be seen by an on-board crewmember — we have simulated enough of interface to see how an expert system could interact with other components of the manager, and to see how a sophisticated user interface can support a crewmember (see [Greitzer86]).

3. A major goal of ARGES is to recognize potential faults before they cause shutdown of the hardware. Currently, the alternatives to using an expert system are either to run the system until it fails and then diagnose the problem, or employ a ground-based human operator to monitor the hardware telemetry downlink for degradations. By adding a forecasting function which is not currently performed by the controller, we can improve crew utilization and reduce the "fire-fighting" mode of operation.

4. ARGES generates recommendations for action based on a simulation of the space environment. Thus , we are beginning to automate the massive operating procedures manuals currently in use, which detail all known contingencies and procedures for dealing with them. By providing this as part of the overall fault handling, crew training for contingencies can be reduced.

Description of the ARGES Problem Domain

The domain chosen for the prototype implementation was the Atmosphere Revitalization Subsystem within ECLSS. In particular, we focused on the atmosphere revitalization "group" of assemblies which remove CO_2 from cabin air and replenish it with O_2. In this group, there is a CO_2 Removal Assembly, which removes CO_2 from cabin air, a CO_2 Reduction Assembly, which combines the CO_2 with H_2 to yield water plus either carbon or methane, and an O_2 Generation Assembly which takes the water from the CO_2 Reduction Assembly and generates pure O_2 to be added to cabin air. For the prototype expert system, we focused on the CO_2 Removal Assembly because it is the most complex of the three. We chose the SAWD system because we had access to the experts (see [Bailey86]). Although ARGES is designed to perform fault diagnosis for the entire atmosphere revitalization group, only the SAWD data are considered because access to data from the other components was not initially provided.

The use of a SAWD CO_2 removal system for manned space platforms has been discussed in detail ([Boehm82]). The following is a brief description of the operation of the SAWD system to familiarize the reader with the technology.

The system consists of two canisters, or beds, of solid diethylenetriamine — "amine" — in a polystyrene substrate. During normal operation, one bed adsorbs CO_2 from cabin air, while the other desorbs CO_2. A fan draws air through the moist adsorbing amine bed. This cools and dries the bed, collecting CO_2 on the amine particles in the bed. Steam is driven through the other bed to desorb it. This initially pushes the remainder of the purified cabin air, or ullage air, out of the bed and concentrates CO_2 at one end of the bed. A sharp increase of flow out of the amine bed signals that CO_2 is now being driven out of the bed. This causes a valve to switch and the CO_2 to be directed to an accumulator and the CO_2 reduction unit. An increase in the bed outlet temperature signals that all the CO_2 has been driven off and the flow is now almost all steam. At this point, valves are switched to connect the output of the desorbing bed to the input of the adsorbing bed, so that the steam in the bed just desorbed can be used to heat the bed that was adsorbing. After this, the roles of the beds are reversed; ie., the desorbed bed now adsorbs CO_2 and vice versa. Operation of the system is performed by a microprocessor that communicates via an RS-232 port or a MIL-STD 1553 network to an external controller/display unit. Fault handling in the controller is confined to limit checking on critical temperatures and times in the system, with out-of-bounds conditions resulting in automatic shutdown and an error indication being sent to the controller/display unit.

The ARGES Architecture

ARGES is organized into four major components (see Figure 1): the input processor, the expert system, the display manager, and the user interface. ARGES can read data from several sources — a software simulation of the SAWD hardware, an RS-232 link to the SAWD controller, or a file (consisting of stored RS-232 data). These data

are passed to the input processor (which performs data transformation for the expert system) and to the display system
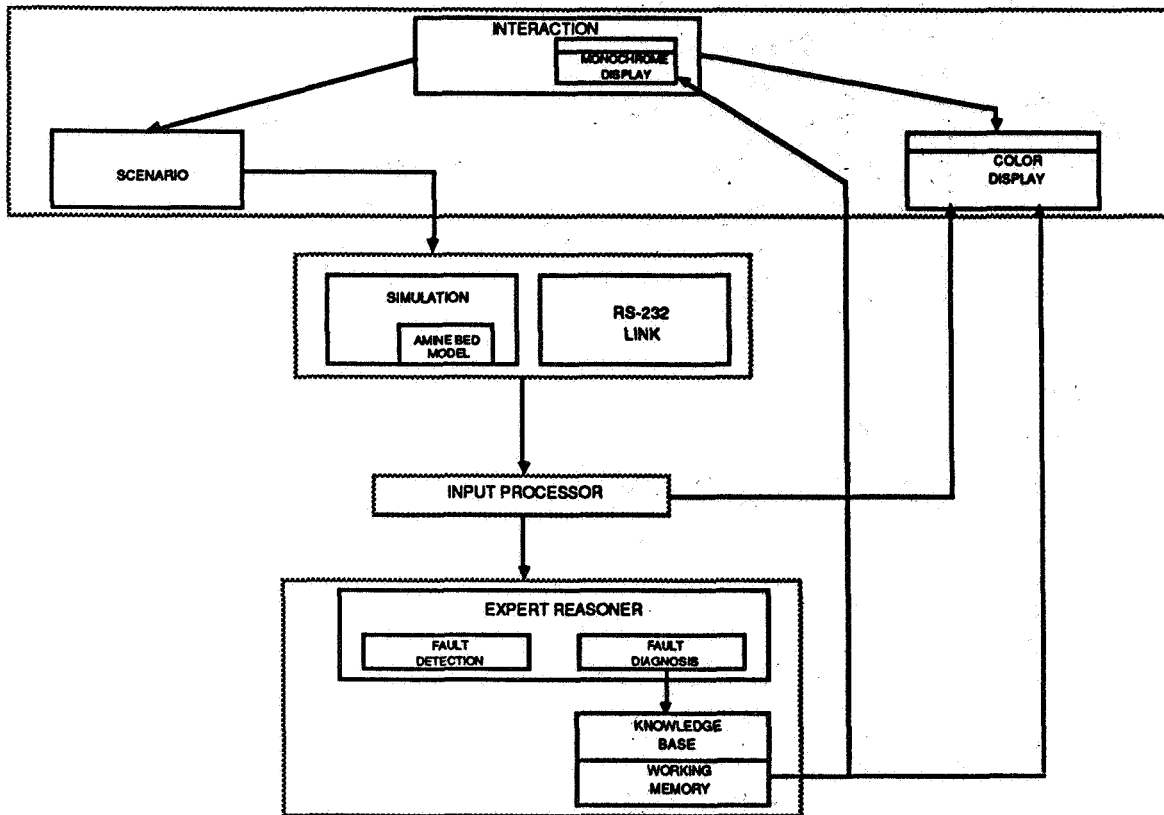
Figure 1. ARGES Structure

for updating the schematic display. After accumulating sufficient data, the expert system makes a diagnosis and sends the conclusion to the display system and user interface. Below, we discuss each of these components in more detail.

The Data Input Sources

The Simulation. The central part of the simulation is a simple model of the SAWD amine bed provided by Hamilton-Standard, Inc. This is a one node version of the computer model they use for hardware design, development and testing of the SAWD [Yanosy85]. The ARGES simulation of the SAWD system allows the graphics display and expert reasoner to be driven with reasonable accuracy in the absence of the actual hardware. Since the trending that the expert system performs may take several days, the use of the simulation at multiples of real-time allows testing more quickly than the actual hardware allows, even if we could induce faults in the hardware to test the system.

SAWD hardware via RS-232 link to controller. Because the SAWD operation cycle is on the order of two hours, and drifts and changes usually occur on the order of days,the only truly time-critical portion of ARGES is the link to the SAWD controller. Once a link is established, the input processor begins to receive information every several seconds. In order to handle these incoming data without lagging behind, the input reader and processor were implemented as processes to separate the real-time data processing from the more time-consuming, but less time-critical operations. Each process can be assigned a separate priority that determines the amount of processor time it receives. Currently in ARGES, the input processing does not operate at high priority, but if the amount of computation changes (through representation changes, or extensions to the expert system or interface), we can force the input processor to run before the other components.

Files. In order to facilitate testing, we recorded several hours of SAWD controller output in a disk file, which can be replayed at varying speeds. Since this data is from the hardware, it enabled us to confirm some operation before we had access to the hardware.

279

Input Processing

The input data from the SAWD controller via the RS-232 link are a series of floating-point numbers and one-byte flag values that reflect the current state of the hardware. These data are converted into the representation used by ARGES and a checksum is validated. If invalid, or if the display and expert system fall behind the input processor, the current data are discarded because the loss of a single data point is insignificant. If either the display or expert reasoner request data when the current value is invalid, they "sleep" until valid data becomes available.

The Expert Reasoner

The expert system operates in two modes: fault detection and fault diagnosis. In the fault detection mode, which occurs during normal SAWD operation, it monitors a few key parameters, or "health indicators", and the status and error-code values received from the controller. When these parameters indicate a potential fault in SAWD, the expert system enters fault diagnosis mode and begins to record data to determine the long-term data trends (if the system has not already shut down).

In the fault diagnosis mode incoming data are collected and stored, then analyzed and trended using linear least-squares fit for several SAWD cycles. Whenever sufficient data are collected, the result ing trend is inserted into the production system working memory and the rule base is invoked to diagnose and verify whether a fault has occurred.

When a fault is diagnosed, the rules invoke procedures to notify the user about the fault detected and provide recommendations. Display functions are also invoked to highlight the faulty component on the SAWD schematic. If the SAWD has not yet shut down, the data trends are extrapolated to find when they will cross thresholds and cause system shutdown. If no fault is recognized, a "default" rule is invoked to notify the operator that an undiagnosed fault exists. A tree of conclusions and antecedents, or inference net, is built as diagnosis proceeds to provide a means of constructing an explanation, discussed further below.

The Knowledge Representation

In ARGES, the expert knowledge is represented using HAPS (the Hierarchical, Augmentable Production System), an in-house developed production system similar to OPS5[Brownston84]. We chose HAPS because it was available at low cost, we had access to the source code so we could enhance it to meet any special requirements, and HAPS provided a reasonably flexible representation — it supports lists, frames, and FLAVOR instances as working memory elements; access to LISP functions on both the left and right hand sides of a rule; user-definable conflict resolution; and access to HAPS structures from the LISP environment.

Initially we used only simple lists for representations of working memory elements (domain knowledge facts), but this has been replaced with a frame hierarchy providing a fault taxonomy and description of the ECLSS. The diagnosis and fault recognition is performed by forward-chaining rules, with recommendations generated from the fault type using the frame hierarchy.

The User Interface

The user interface for ARGES performs several functions: a) it depicts graphically the operation of SAWD system to facilitate user understanding, b) it displays the conclusions reached by the expert system, c) it allows the user to examine the chain of reasoning used by the expert system. In order to study how an expert system could be integrated with the rest of a space station management system, the user interface simulates a more complete interface between a crew member and a more generalized fault management system. In particular, the fault management interface simulates other components of the overall space station management function and the status & warning display for the space station, although only that component dealing with the CO2 removal assembly is actually functional. By simulating these functions, we can demonstrate how an AI-based fault management system can enable a crewmember to detect and diagnose faults, perform temporary work-arounds, and take corrective actions with reduced knowledge of the Space Station systems and reduced ground support.

The interface uses two monitors, one color and one monchrome, and a mouse. It is based on the direct manipulation, object-oriented approach found in the Lisp machine and Smalltalk environment[Goldberg80, Symbolics85]. Text and objects displayed on both the monchrome and color screens are, to the extent possible, "mouse-sensitive". Inquiries concerning the represented systems, components, sensors, etc., are made by interacting with the displayed objects directly. In an operational system, the mouse would be replaced by a suitable zero-G device, such as a track ball.

The monchrome screen primarily displays textual information, such as error messages, and menus of possible corrective actions. Notifications of errors happen as highly-visible "pop-up" text boxes with an indication of the response time required by the user. A set of possible responses recommended by the expert system is an associated "pop-up" menu. In addition, significant events are recorded in a scrollable log. When the user requests, an additional window appears to display an explanation of the current conclusions.

To provide explanation, we make use of the inference net built during the fault diagnosis process. Associated with each working memory element is the description of the conclusion, which is easily converted to English text and displayed. Each displayed conclusion is mouse-sensitive, and clicking on it results in the addition (or removal) of its antecedent symptoms to the display. The user can traverse the inference net from the final conclusion back to the leaves of the net (i.e., trended data).

The color screen is the primary means of displaying information to the user. Under normal circumstances (before a fault has been recognized) a simple, hierarchical diagram of the space station systems is displayed. This "map" graphically provides the current status of various systems (off, on, warning, alarm, etc.). The user can click on a component box to see an expansion down to its subsystems with the status of each subsystem displayed. At the bottom of the hierarchy, the schematics of the assemblies are available, with the diagram dynamically updating to show the current status of each assembly. When a fault occurs, the appropriate subsystem boxes change state. In the assembly schematic, the icon of the faulted component also changes state, to graphically display the fault location isolated by the expert reasoner. As an aid to the user to help visualize the relationship of the faulty subsystem, the subsystem hierarchy is reproduced in the lower right corner, complete with status and possible other fault indications. This enables the user to always have an indication of overall status (or other problems) while dealing with a fault in one subsystem.

Current Status of System

ARGES has been implemented with a small set of faults on a Symbolics 3675 and an LMI Lambda. We have concentrated primarily on predictable faults, since this gives the additional capability of predicting a failure before it happens, as well as diagnosing a fault to a single component. We have tested it against the SAWD prototype at Marshall Space Flight Center for monitoring normal operations, but limitations on the testing of the hardware prevent us from introducing faults into the hardware to test the expert system. Currently, the system is undergoing an evaluation study of the expert system and user interface (see [Greitzer86]).

Conclusion and Future Directions

ARGES is a demonstration of AI technology applied to the problem of fault diagnosis and handling for manned space platforms. By applying AI/expert system technology, we can perform functions not possible with conventional controller software, such as: detection of degradations before they result in failure, providing greater flexibility in handling faults and modifying the fault software, and providing a higher level of interaction between the Space Station fault software and crewmember. The ultimate goal is to enable the crew to function more as managers and decision-makers, and less as interpreters of data and procedures manuals.

We are currently working to expand the simulation of the crew interface, so that more of the characteristics of an integrated "fault management" system can be evaluated. In addition, we would like to expand to a full set of faults for the entire atmosphere revitalization group. One of the limitations of the approach we have taken is that rule-based systems can only encode previously conceived faults. We would like to build a model-based reasoner,

281

either quantitative, based on the simulation we use currently, or qualitative, to provide the deeper, causal reasoning necessary when the shallow rule-based approach is inadquate.

References

[Bailey86]   Bailey, Pat, and Doehr, Brett, "Knowledge Acquisition and Rapid Prototyping of an Expert System: Dealing with "Real World" Problems", paper presented at the Conference on Artificial Intelligence for Space Applications, Nov. 13-14, 1986, Huntsville, Alabama.

[Boehm82]   Boehm, Albert M., and Cusick, Robert J. "A Regenerable Solid Amine CO2 Concentrator for Space Station" Twelfth Intersociety Conference on Enviromental Systems, July 19-21, 1982, San Diego, CA.

[Browston84]   Brownston, Lee and Robert Farrell, Elaine Kant, Nancy Martin, Programming Expert Systems in OPS5, Addison-Wesley, Reading, MA, 1984.

[Dickey84]   Dickey, Frederick J., and Toussaint, Amy L. "ECECIS: An Application of Expert Systems to Manned Space Stations", Conference on Artificial Intelligence Applications Proceedings, IEEE, 1984.

[Goldberg80]   Goldberg, Adele, Smalltalk-80, Addison-Wesley, Reading MA, 1980

[Greitzer86]   Greitzer, Frank, "Intelligent Interface Design and Evaluation", paper presented at the Conference on Artificial Intelligence for Space Applications, Nov. 13-14, 1986, Huntsville, Alabama.

[Lance85]   Lance, Nick and Malin, Jane T., "Development of a Prototype Expert System for Fault Diagnosis of a Regenerative Electrochemical CO2 Removal Subsystem", Ref. No. EC3/SR111, NASA/Johnson Space Center, Houston TX, May 13, 1985.

[Symbolics85]   Symbolics, Inc., Symbolics Reference Manual, Vol. 7, Programming the User Interface, Cambridge, MA 1985.

[Yanosy85]   Yanosy, James L., and Rowell, Lawrence F. "Utility of an Emulation and Simulation Computer Model for Air Revitalization System Hardware Design, Development, and Test" Fifteenth Intersociety Conference on Enviromental Systems, July 15-17, 1985, San Francisco, CA.

SCARES

A Spacecraft Control Anomaly Resolution Expert System

Marc Hamilton

TRW Inc.

1 Space Park

Redondo Beach, CA 90278

Abstract

The current pace of our technological development is reflected in the increased mission lifetime of each new generation of satellite. Coupled with this has come a reduced availability of experts to provide technical assistance in satellite operation on a day to day basis. Given such an environment, this paper discusses an expert system based architecture for spacecraft anomaly resolution. By capturing "deep knowledge" about a spacecraft, the system is able to detect and diagnose faults better than previous conventional approaches.

A prototype expert system named SCARES (applied only to a spacecraft attitude control system) is discussed. This prototype was developed in the Artificial Intelligence Technology Center of TRW's System Development Division, under contract F04701-83-C-0079 for the U.S. Air Force Space Division. Extension of the prototype to handle anomalies in other systems of the satellite is also discussed.

Keywords: Expert Systems, Artificial Intelligence, Diagnosis, Satellite Control

Introduction

Command and control of today's spacecraft is a complicated process which includes monitoring telemetry, determining vehicle health and status, and generating command messages. Currently, these tasks are carried out in ground stations equipped with mainframe computers and staffed with many operators. As newer satellites become more sophisticated, failures become fewer, but the diagnosis of remaining failures becomes more difficult [1]. When spacecraft anomalies do occur dozens of experts are often called in to investigate the problem. As we aim to increase the survivability of future spacecraft it is evident that much of this process must be automated. Working toward the goal of autonomous spacecraft, SCARES attempts to diagnose anomalies in the attitude control system of a spacecraft using an expert system approach.

Background

Some background on the attitude control system to be monitored is needed to understand how SCARES works. The attitude control system's function is to keep the spacecraft correctly pointed toward the earth and to maintain the correct orbit. The attitude control system is composed of four separate assemblies, each with a dual redundant backup. The Sun Sensor and the Earth Sensor sense the presence of the Sun and Earth, respectively. These signals are then used as inputs to the Control Electronics which determines the attitude of the spacecraft. If necessary, the Valve Driver is signaled to activate control jets which keep the spacecraft pointed toward the earth. Various signals are sampled at different points in this process and sent to the ground station as telemetry. Redundant units can be switched in via ground command if necessary. A simplified functional flow of this process is presented in Figure 1.

The earth sensor is a critical component in the earth pointing process, which keeps a satellite correctly pointed toward the earth. A typical scanning type earth sensor includes a mirror drive mechanism which scans back and

forth across the earth's horizon at a high rate, as shown in Figure 2a. Reflection of sunlight from the earth's surface (or the lack of reflection when pointing off the horizon) is directed by the mirror onto a sensor. The earth sensor electronics then generates a raw radiance signal (RAD). The RAD then is used to calculate a horizon crossing signal (HCS) which corresponds to "1" when the earth sensor is pointed toward the earth and "0" when it is pointed off the earth. If the HCS is toggling at the correct rate, the track check (TC) signal will be set to "1". A pointing angle signal (PAS) indicating the pointing angle of the spacecraft is also generated by the earth sensor electronics. A sample graph of these figures is shown in Figure 2b.

System Overview

The current SCARES system is implemented in Lisp on a Symbolics 3670 workstation. It uses a frame based knowledge representation scheme and handles both inductive (forward chaining) and deductive (backward chaining) rules. The SCARES software is implemented using an object orientated programming approach. Among its many advantages, the object orientated implementation allows separate components or sub-components to be implemented as independent asynchronous processes.

The general SCARES architecture is shown in Figure 3. This architecture breaks down the anomaly resolution process into three main steps. First, a real time monitor detects faults at an early stage, while their signatures are still confined to one area. Next, a diagnosis routine attempts to localize the fault down to a line replaceable unit or assembly. In doing so, the diagnosis component may instruct the monitor to perform retry or fault masking operations on itself so that the monitor may continue to operate correctly. Thirdly, a hypothesis generation and test component is invoked to handle faults not easily resolved by the diagnostic process and to suggest possible reconfiguration and recovery actions. These three components were designed to correlate with spacecraft command and control functions which can be classified as:

- routine, on-line (monitoring)
- routine, off-line (diagnosis)
- non-routine, off-line (hypothesis generation and test)

The SCARES Environment controls and coordinates the actions of the above three routines. All user interface functions are managed by the environment. It also maintains a central situation data base for use by the other SCARES modules. This data base stores the telemetry history, the current satellite configuration, performance statistics, an alarm log, the operator log, and all commands issued to the satellite. The environment is also responsible for storing and maintaining the different knowledge bases which make up the rules used by SCARES. Before any of these rules can be triggered, however, the environment needs to receive telemetry data from the telemetry preprocessor.

Telemetry Preprocessor

The Telemetry Preprocessor is responsible for converting the telemetry obtained from different sources into a format suitable for use by the rest of the SCARES system. Different types of telemetry input are accepted by the preprocessor. The first record format consists of mnemonic name, timestamp, and value triplets. This generic satellite independent format is used by the simulation which provides data for the prototype. Alternately, data in the mainframe format, which is satellite specific, can be read from telemetry tapes or directly from the groundstation by the SCARES system.

A typical mainframe format might consist of 128 8 bit words which are being downlinked from the satellite every 8 milliseconds. Most of these words are independent telemetry signals which are downlinked in each mainframe. Some low frequency signals, however, may be telemetered less often while high frequency signals, such as the earth sensor HCS, may be telemetered several times per mainframe. Additional telemetry formats for handling data from remote tracking stations are also provided.

Monitor

The SCARES monitor provides the main source of fault detection in our system. It does so by performing three types of real time checks on downlinked telemetry. First, a limit check is done on individual telemetry points. Next, a rate check is done on two or more telemetry points in the same channel. Third, cross-channel checking tests for consistency of the telemetry. By contrast, current ground station monitoring consists almost entirely of simple limit checks.

The SCARES monitor actually consists of five separate monitors, the Earth Sensor monitor, the Sun Sensor monitor, the Control Electronics monitor, the Valve Driver monitor, and a master monitor which controls the previous four. The master monitor receives telemetry data and control information from the SCARES environment and distributes it to the appropriate monitors. The individual monitors then process the data and return error messages to the master monitor. The error messages are then returned by the master monitor to an alarm handler residing in the SCARES environment.

The highest data load is handled by the Earth Sensor monitor, which must monitor three signals which are downlinked four times per mainframe (every two milliseconds). These signals are the Horizon Crossing Strobe (HCS), the Raw Radiance Signal (RAD), and the Pointing Angle Signal (PAS). Rather than attempting to analyze all this data, which would be computationally inefficient, the monitor only receives an eight mainframe sample (32 values) of each signal once every two seconds. These samples are sufficient for the monitor to detect any fault in the earth sensor and allow the monitor to provide real time performance.

Diagnosis

Once an alarm has been generated by the SCARES monitor, it is the job of the diagnosis component to isolate the alarm to a faulty assembly. There are several steps to this process which can be separated into prediagnosis and diagnosis actions.

During prediagnosis, a monitor alarm or sequence of alarms is first translated into a form suitable for presentation to the user. Some alarms contain information related to the monitor design which should not be presented to the user, who is only expected to be familiar with the actual satellite. For instance, the ESA monitor produces the alarm

"stuck in state 2"[1]

which in itself contains no useful information for the user. However, when coupled with several proceeding alarms, the "stuck in state 2" alarm allows the prediagnosis component to conclude that the HCS signal is stuck high.

The prediagnosis component also handles reconfiguring the monitor when necessary. For instance, if the HCS signal is stuck high, the ESA monitor will continue to fail in state 2, thus losing any potentially valuable information which might come from other states. In order to allow the ESA monitor to continue operation in a degraded mode, the prediagnosis component instructs the ESA monitor to ignore the HCS signal in future operations. The ESA monitor can now continue to cycle through all its states and produce additional alarms.

At this point the actual diagnosis process begins. Each new set of alarms triggers a production system which attempts to produce conclusions in the form of diagnostic information for each alarm. Various diagnostic knowledge sources contain rules for localizing faults to line replaceable units or assemblies, determining the potential impact of faults on the attitude control system's operation, and for recommending corrective action. Alarms can recycle through this process for several iterations, as new alarms or conclusions may help to diagnose a previous alarm. Eventually, if no conclusions can be reached, or alternatively if requested by the operator, an alarm may pass to the hypothesis generation and test stage.

---

[1]The "stuck in state 2" alarm refers to the Earth Sensor Monitor finite state machine which is used to test for cross channel telemetry consistency.

285

Hypothesis Generation and Test

The hypothesis generation and test stage consists of a number of loosely structured routines which attempt to guide the user to solutions which cannot be reached by the diagnosis component alone. The hypothesis generation stage attempts to reason from both a functional flow orientated and a hardware orientated perspective.

At a functional level, rules attempt to determine the high level function which caused the alarm. The corresponding functional flow diagram is then displayed for the user. An attempt is then made to draw correlations, both automatically (by SCARES) and manually (by the operator), between the alarm, its related function, and other potentially related alarms or functions.

On a hardware level, the same type of reasoning is followed. The alarm is mapped to a set of hardware tables which indicate potentially faulty hardware. A top level view of the circuit board or component in question is then displayed. Operators can choose to highlight all related components, all unique components, or both. The operator can pan and zoom over large displays, requesting more detailed diagrams, even down to the gate level, if desired.

All operator activity in the hypothesis generation stage is logged for future reference. If the same alarm ever triggers the hypothesis generation stage again, SCARES remembers what the operator did the last time this alarm appeared and the conclusion of the actions taken. This information can then be used to guide the user through the current alarm's hypothesis generation stage. In this manner, SCARES has a limited ability to actually "learn" how to resolve an anomaly.

Once a hypothesis has been generated, it then needs to be validated by testing. In order to do this, SCARES uses a version of the deduction algorithm [2] to diagnose faulty hardware. The deduction algorithm uses a circuit description of the part in question and primary input and output points (provided by telemetry values) to deduce if the hypothesis is correct. The deduction algorithm is guided by the circuit's network topology (rather than an algebraic description of the circuit) to a

posteriori deduce the internal line values of the circuit. For example, if a nand gate output is "0", then both inputs should be "1". The "1's" are then traced back until a primary input point is reached. If there is a contradiction at the primary input, the algorithm backtracks to the last decision point, undoing all values along the way. Using hierarchical design models of the hardware components, as proposed in [3], the deduction algorithm first localizes faults at a board or component level and only proceeds to the circuit and gate level when additional information is needed.

Example Scenario

The current SCARES system is designed to handle a wide range of anomalies, spanning both electrical and mechanical failures and chosen from historical and postulated scenarios. The following is a sample anomaly scenario which deals with an ESA related anomaly sequence and exercises the three major SCARES components; the monitor, the diagnosis module, and the hypothesis generation and test module. A timeline of events is shown in Figure 4.

Event E1 represents an operator shift change. At shift change, the new operator is briefed by SCARES which displays the current satellite configuration and key performance statistics while allowing him to review the previous operator's daily log.

A loss of track check alarm is generated by the ESA monitor at event E2. The diagnostic module immediately discovers that this is due to a moon avoidance maneuver. When the moon appears on the earth's horizon, the earth sensor is programmed to scan around the moon, thus briefly losing the earth track check. The operator requests a display of the ESA raw radiance, horizon crossing, and track check signals and confirms that the earth sensor was undergoing a moon avoidance maneuver at the time of the alarm. Since the moon will stay on the earth's horizon for approximately twenty minutes, the diagnosis process makes a note of this in the situation data base. While the monitor will continue to generate loss of track check alarms, future alarms of this type will be ignored as long as they can continue to be explained by the moon

avoidance maneuvering.

At event E3, while the moon avoidance warning is still in effect, the ESA monitor reports that the HCS and RAD signals are out of sync. Normally, when the raw radiance exceeds a threshold the HCS goes high indicating that the earth sensor is on the earth. The opposite happens as the sensor scans off of the earth. In order to determine if the HCS or the RAD is stuck high or low, the diagnosis process automatically requests that the monitor rerun the current data set, once ignoring the HCS and once ignoring the RAD signal. The outcome of this test determines that the HCS is stuck high. Once this is determined, the diagnosis process instructs the ESA monitor to ignore the HCS in the future and alerts the operator to the condition.

The preliminary diagnostic conclusion determines that the HCS stuck high alarm is due to a mirror pivot failure in the primary earth sensor. At the same time, however, the diagnostic process notices that the track check is high while the moon avoidance maneuver is occurring. This cannot be explained by any of the diagnostic rules so the hypothesis generation and test stage is automatically invoked.

At this point it is necessary to generate a hypothesis which explains why the TC and HCS signals are both stuck high. An ESA mirror pivot failure is discounted since such failure could not effect both the TC and the HCS without also effecting the RAD signal, which is still nominal. By checking hardware tables, it is determined that the TC and HCS signals are both routed through amplifiers on a single chip in the CEA. That chip is therefore hypothesised as being faulty.

This hypothesis can be tested by using the deduction algorithm [2]. By using CEA telemetry points as primary input and output references, the deduction algorithm deduces that only "1" values have occurred at the output of the chip in question, while both "1" and "0" values should have occurred. The conclusion is then made that the alternate CEA should be switched in.

At event E4, the operator switches in the alternate CEA. In a real system, this would be a lengthy procedure involving many ground station components in addition to SCARES.

The master monitor is then automatically reinitialized and the ESA monitor begins monitoring the HCS again. The HCS and TC signals now appear normal. All actions taken have been logged for reference in case similar problems occur in the future. The operator also makes an operator log entry to supplement the automatic logging and for future reference by other operators.

Extensions

The current SCARES prototype only handles anomalies in the attitude control system of a satellite. Planned extensions would allow monitoring and diagnosis of other satellite systems, including thermal, power, and payload systems. This would be accomplished by adding additional monitor components for each new subsystem and increasing the diagnostic rule base.

Because of SCARES current object orientated implementation, communication and interaction with new modules via message passing could be easily accomplished. This feature would simplify the sharing of interim conclusions among multiple components. For instance, a transient power subsystem fault, if detected, could help explain a related earth sensor signal anomaly. The object orientated approach also lends itself to distributed processing architectures, with each object being implemented as a independent asynchronous process.

The SCARES prototype was designed to be expanded into a functional workstation which provides automatic diagnosis and manual (operator directed) assistance to current mission operations in a ground station environment. Future extensions would add additional automatic diagnosis capabilities and would make SCARES suitable for use in mobile ground stations. Eventually, the entire anomaly resolution process could be handled on board a spacecraft which would then be able to operate autonomously for months at a time.

Conclusions

Longer mission lifetimes, increased satellite complexity, and the reduction of expert assistance availability have all led to requirements for more highly autonomous spacecraft.

287

While traditional ground station monitoring eventually detects most anomalies, diagnosis of even straightforward problems often requires expert intervention. Often, such aid can come too late. Initial fault signatures may have spread throughout the telemetry and the anomaly resolution process becomes dragged out and time consuming.

The SCARES prototype demonstrates how expert system technology can be applied in large real-time fault diagnostic systems to detect, diagnose, and recover from complicated anomalies. With its three stage approach of monitoring, diagnosis, and hypothesis generation and test, SCARES technology can be used today in ground station environments to help resolve spacecraft anomalies. Once mature, these techniques could be coupled with next generation compact LISP machines and actually be placed onboard what would be a highly autonomous spacecraft.

Acknowledgements

References

[1] A. Brindle and M. Pazzani, "An Expert System for Satellite Control," *Proceedings ITC/USA/'85 International Telemetering Conference,* 1985

[2] M. Abramovici and M. Breuer, "Multiple Fault Diagnosis in Combinational Circuits Based on an Effect-Cause Analysis," *IEEE Transactions on Computers,* Vol. C-29, No. 6, June 1980

[3] M. Genesereth, "Diagnosis Using Hierarchical Design Models," *Proceedings AAAI-82,* pp. 278-283

[4] W. Arens, "ARMMS Spacecraft Health and Maintenance Software Performance and Design Requirements," *Technical Report 7030-15,* Jet Propulsion Laboratory, 1984

EARTH RADIANCE

SUN RADIANCE

FROM GROUND STATION

EARTH SENSOR

SUN SENSOR

CONTROL ELECTRONICS

COMMAND DECODER

INPUT PROCESSING

CONTROL PROCESSING

OUTPUT PROCESSING

TELEMETRY PROCESSING

VALVE DRIVERS

TO GROUND STATION

CONTROL JETS
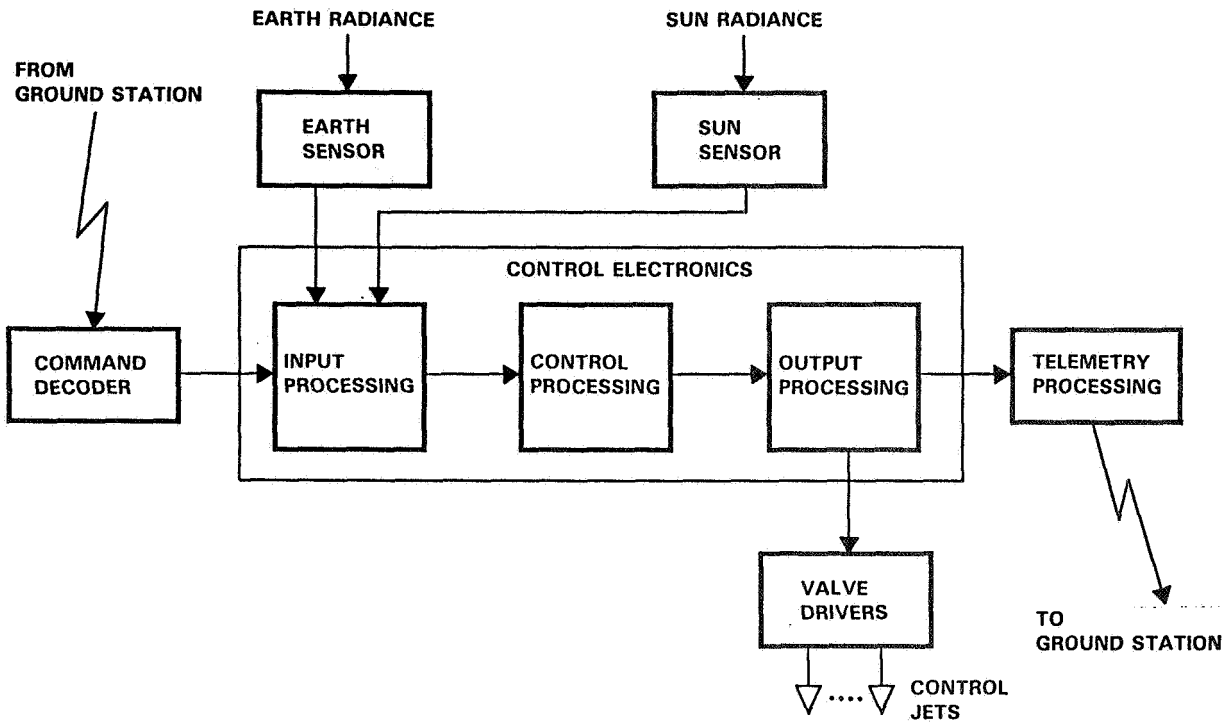
Figure 1. Spacecraft Attitude Control
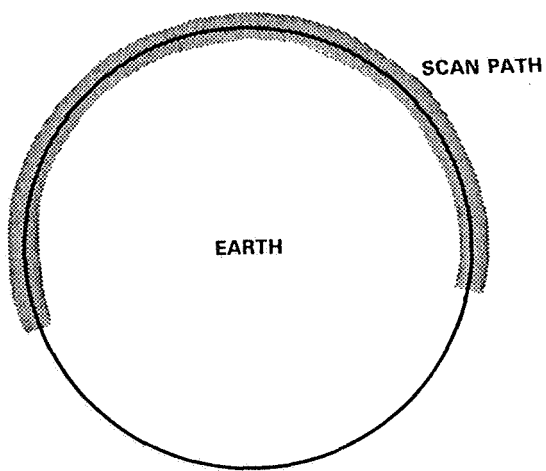
SCAN PATH

EARTH

Figure 2a. Earth Sensor Scan Path

RR

PAS

HCS
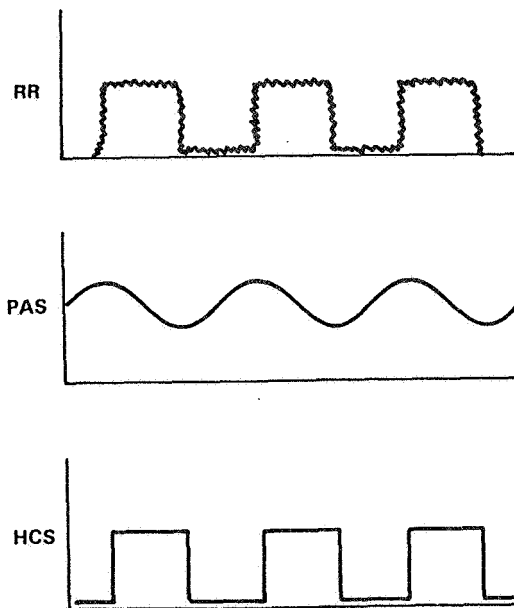
Figure 2b. Earth Sensor Signals

289
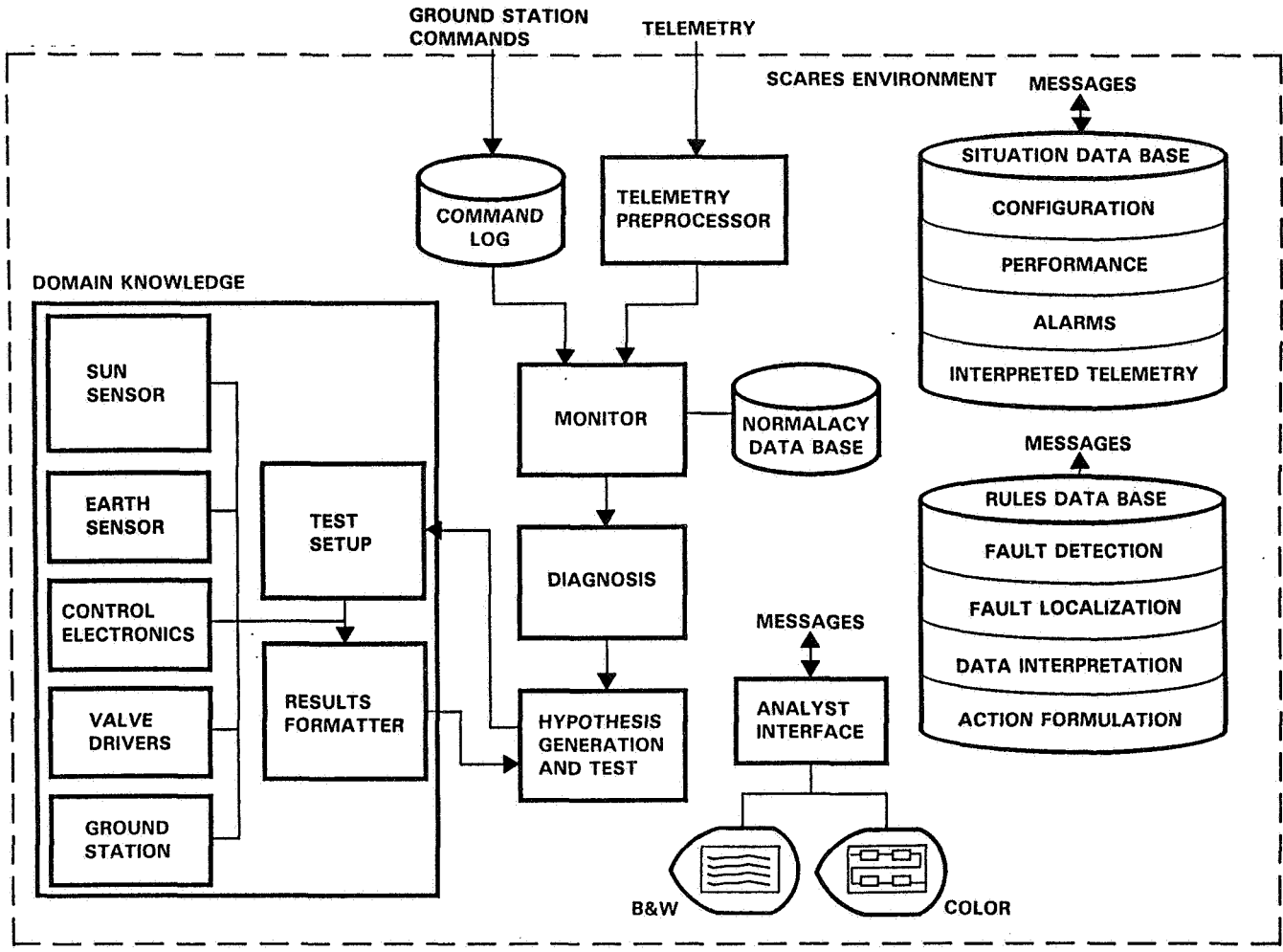
Figure 3.   SCARES Architecture

290
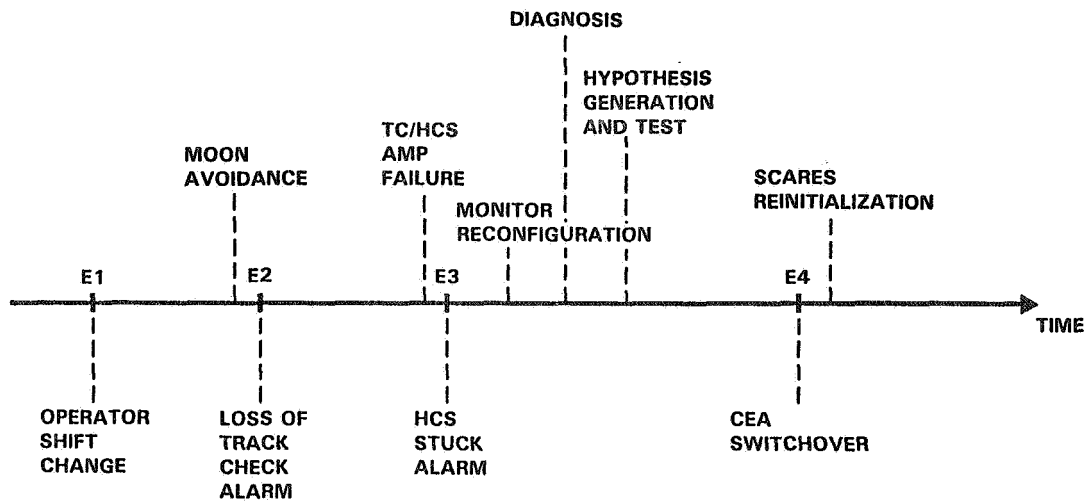
Figure 4. Sample Scenario Timeline

Archetyping - A Software Generation and Management Methodology

Hugh B. Rothman and Stanley M. Przybylinski
General Dynamics Data Systems Division
P. O. Box 85808   V2-5530
San Diego, California 92138

Many knowledge-based software generation methods have been proposed to improve software quality and programmer productivity. Several government and industry initiatives have focused on software reusability as one solution to these problems. DARTS™, a General Dynamics-proprietary symbolic processing technology, provides a unique solution to the reuse problem: archetyping.

Archetyping is the embedding of high-order language statements in text files. An advanced macro-processor uses the text files to generate new versions of complex software systems. A DARTS program, the Software Generation and Configuration Management (SGCM) System automates the archetyping process and maintenance cycle. This paper briefly discusses the DARTS technology, describes archetyping, and presents in detail, the SGCM system.

1.   DARTS

The DARTS technology provides a language, an interpreter, and storage files. The language of DARTS is AXE™. It offers many advanced software concepts such as procedural constructs, pattern matching, dynamic knowledge representation, and list processing. AXE statements are demarcated by '<' and '>'.

The BOLT processor interprets and executes all AXE statements in a text file. All other text is passed unchanged to a result file.

Knowledge bases are files in which permanent storage areas called "metasymbols" reside. Integers, strings, lists, and executable AXE code can be stored as metasymbols.

```
          ┌──────────┐              ┌──────────┐
          │    A     │              │Knowledge │
          │  DARTS   │─────────────▶│  Bases   │
          │Interface │              └──────────┘
          │ Program  │
          └──────────┘
```

FINAL TEXT
PRODUCTS

ARCHETYPE FILE

This is archetype:
<_\:ARCHNUMBER>

- - - - - - - - - - - - - - -
- - - - - - - - - - - - - - -

date is <_DATE>

- - - - - - - <- - - ->- -
- - - - - - - <- - - - ->-
- - - <- - - > - - - -
- - <- - >- - <- - >- - - -
- - - - - - - - - - - - - - -

- - - - - - <- - - - - - >- - -
- - <- - - - >- - <- - - >-
- - - - - - - <- - >- - - - -

BOLT
PROCESSOR

This is archetype
1
date is 12-sep-86

This is archetype:
2
date is 13-sep-86

This is archetype:
n

date is dd-mm-yy
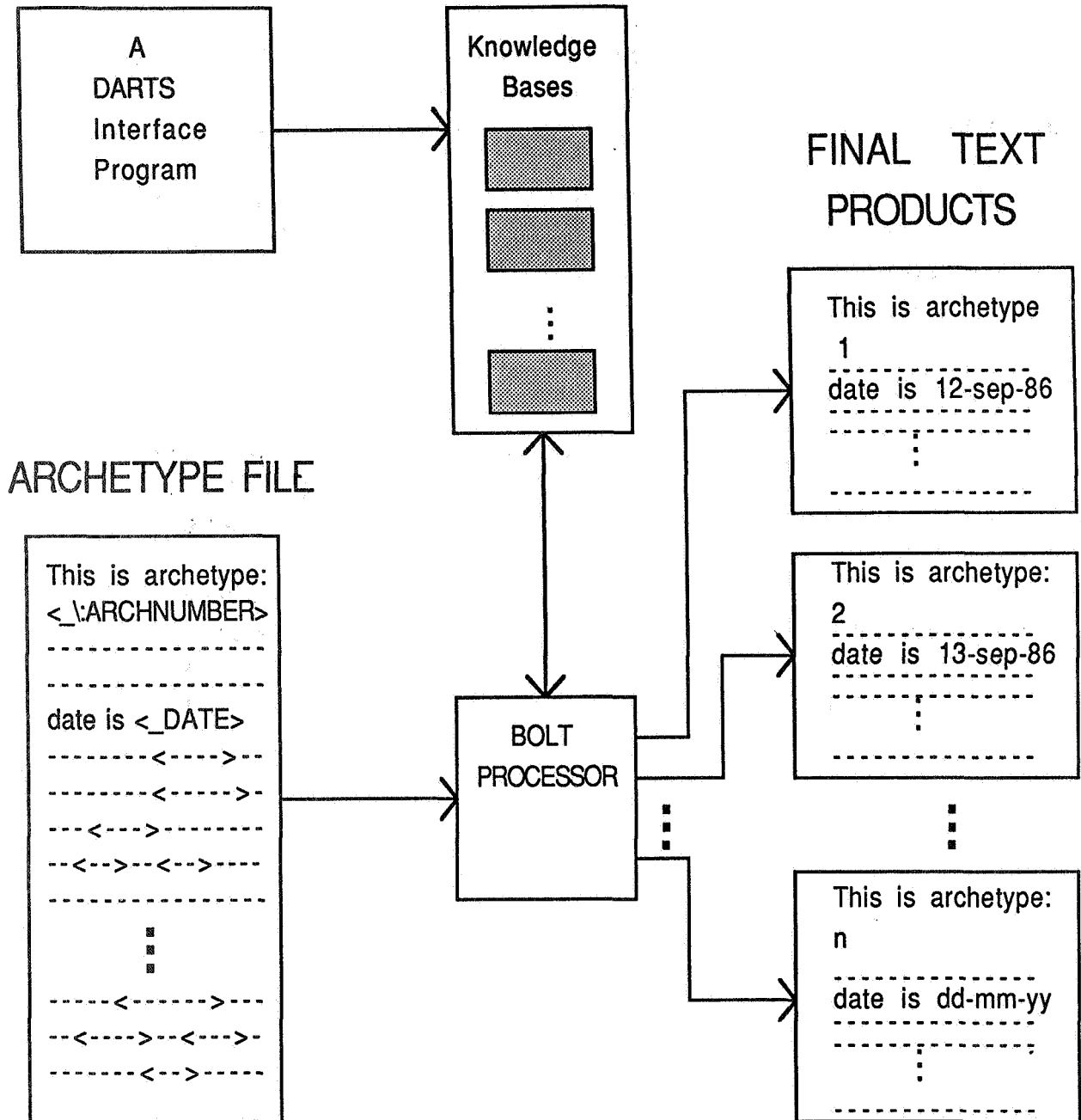
FIGURE 1:   The   Archetyping   Process.

2.   ARCHETYPING

Any text file may be archetyped. This includes source code, documentation, design specifications, and test requirement documents (TRDs). A text file is made into an archetype file by embedding AXE statements within the text. The AXE statement "JUMP" sends the

archetype file to the BOLT processor which processes all AXE statements in the archetype file and passes all non-AXE text to a result file. AXE statements control the actions of the BOLT processor. Some actions are:

1). Skip a section of an archetype file conditionally or unconditionally.
2). Jump to another archetype file. Processing will continue at the beginning of the called archetype file.
3). Iterate on a section of an archetype file.
4). Return a value and pass it to the result file.

At the end of a called archetype file, processing returns to the position immediately following the jump call in the calling archetype file. When archetyping is complete, the result file is the final text product.

Knowledge bases play a key role in archetyping. Most AXE statements in an archetype file access metasymbols stored in a knowledge base. Most archetyping applications have a user-friendly DARTS program that lets the user manipulate the metasymbols and consequently, determine the final text product. Figure 1 illustrates the archetyping process.

3. THE SGCM SYSTEM

The SGCM system was designed and built to automate the archetyping process. Manual archetyping is too unwieldy to be practical for a large software system. Large archetype files become unmaintainable and it is impossible to manually keep track of the duties of many archetype files. Ambitious software manufacturing requires the automatic archetyping of the SGCM system. Figure 2 shows the components of the SGCM system.

The system is built around two basic structures, the configuration and the segment. A user-definition language defines and uses these structures. A user interface performs the manufacturing and management. The system generates an archetype file for each configuration and segment. It also stores pertinent information about each configuration and segment as metasymbols in two management knowledge bases. The archetype files and management knowledge bases are handled internally by the system and are invisible to the user.

A configuration is a short-hand version of a final text product. An archetype file is generated from a configuration definition. The system uses this archetype file to create a stand-alone final text product.

A segment is a reusable module of text designed to be repeatedly used by configurations and other segments. Segment parameters can be defined and used to obtain multiple variations of a segment. Segment definitions generate archetype files just like configuration definitions. However, segment archetype files can be jumped to only by other archetype files. A segment archetype file is never used to create a stand-alone final text product.

The system provides "environments" for problem partitioning. All configurations and segments are defined in some environment. Configurations and segments may only use other configurations and segments defined in the same environment. This lets a user define configurations and segments for several environments but give each the same name. For example, a segment 'start' may be defined in environment source-code, documentation, and TRD. If a configuration defined in the documentation environment uses segment 'start', the segment accessed is the segment 'start' also defined in the documentation environment.

3.1. THE USER-DEFINITION LANGUAGE

All configuration and segments are defined and used in definitions with the User-Definition language. Configuration and segment "definition" statements create archetype files. Configuration and segment "use" statements become "jump" statements in an archetype file. See figures 3 and 4 for examples of the User-Definition language.

3.1.1. The Configuration Definition Statement

The configuration definition statement requires a configuration name and an environment specification. The definition contains a mixture of free text and "use-segment" and "use-configuration" statements. When encoded, the resulting archetype file will contain the free text and AXE "jump" statements to other archetype files. The created configuration archetype file is accessed in the manufacturing phase to create the final text product. The configuration definition is also used to update the management knowledge bases with the new information.

296

SGCM SYSTEM

Support Files

```
┌─────────────┐      ┌─────────────┐      ┌──────────────┐
│   Manager   │      │   Segment   │      │Configuration │
│  Knowledge  │      │  Archetype  │      │  Archetype   │
│    Bases    │      │    Files    │      │    Files     │
└─────────────┘      └─────────────┘      └──────────────┘
```

User Interface

```
┌──────────┐  ┌──────────┐  ┌──────────┐  ┌──────────────┐
│  System  │  │ Encoder  │  │ Manager  │  │ Manufacturer │
│ Services │  │          │  │          │  │              │
└──────────┘  └──────────┘  └──────────┘  └──────────────┘
```

```
┌─────────────┐      ┌─────────────┐      ┌─────────────┐
│ Definitions │      │    SGCM     │      │    Final    │
│    File     │      │   System    │      │    Text     │
│             │      │    Info     │      │   Product   │
└─────────────┘      └─────────────┘      └─────────────┘
```
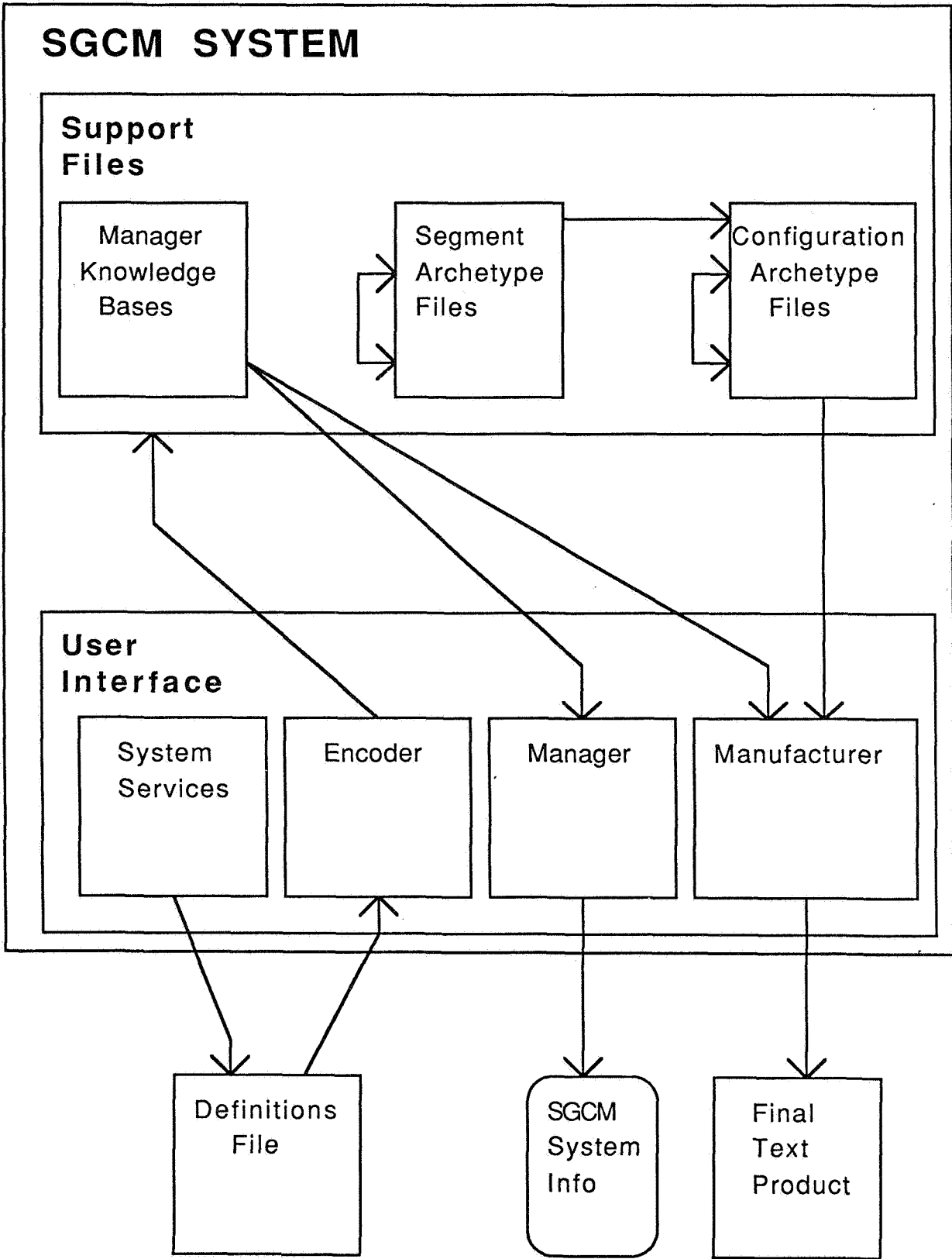
FIGURE 2: The SGCM System

297

3.1.2.   The Segment Definition Statement

The segment definition statement allows the same segment name to be defined in several environments at the same time. Each segment definition contains a mixture of text and "use-segment" statements. The segment may also have parameters defined inside itself. The parameter values are obtained from the "use-segment" statement when the segment is used. In addition, an optional description section is provided allowing a user to describe a segment in a few sentences. This description is later accessed by users who are unfamiliar with the segment. The definition statement generates an archetype file for each segment defined and updates the management knowledge bases.

3.1.3.   The Segment Use Statement

The "use-segment" statement appears inside a configuration or segment definition. This statement results in an AXE "jump" statement being placed in the defined configuration's or segment's archetype file. During the manufacturing phase, the "jump" statement causes the used segment's archetype file to be processed. The final text product contains the results of the jump. The use segment statement allows an optional list of parameter values, which replace the parameter definitions inside the used segment. Values passed to undefined parameters are ignored.

3.1.4.   The Configuration Use Statement

A configuration may use another configuration. The new configuration is an exact copy of the used configuration if no segment manipulation is specified. The system provides segment deletion, replacement, and appending within the configuration use statement. The deletion utility prevents specified segments from appearing in the new configuration. The replacement option takes two lists of segments as input. Each segment in the second list replaces a corresponding segment in the first list. Appending also takes two lists of segments as input. Each segment in the second list is appended onto the corresponding segment in the first list.

3.1.5   Segment Definition Nesting

Segments can be defined anywhere, even inside other definitions. In the encoding phase, outer definitions are processed as usual until a nested

DEFINITIONS FILE

```
{DEF-SEGMENT SEG-A}
  {BEGIN-SOURCE}
      -----------
      -----------
    -source  code-
      -----------
      -----------
  {END-SOURCE}
  {BEGIN-DOC}
      -----------
      -----------
    documentation
      -----------
      -----------
  {END-DOC}
{END-DEF}

{DEF-CONFIG  CONFIG-A}
  {SOURCE}
      -----------
      -----------
    -source  code-
      -----------
    {USE-SEGMENT  SEG-A}
      -----------
      -----------
  {END-DEF}

{DEF-CONFIG  CONFIG-A}
  {DOC}
      -----------
      -----------
    documentation
      -----------
    {USE-SEGMENT  SEG-A}
      -----------
      -----------
  {END-DEF}
```

ENCODER

ARCHETYPE FILE
FOR
SOURCE SEGMENT
SEG-A

ARCHETYPE FILE
FOR SOURCE
CONFIGURATION
CONFIG-A

ARCHETYPE FILE
FOR
DOCUMENTATION
SEGMENT
SEG-A

ARCHETYPE FILE
FOR
DOCUMENTATION
CONFIGURATION
CONFIG-A

FIGURE 3: A Definitions File is Encoded

definition is reached. The outer definition processing pauses and the
nested definition is processed. When processing of the nested definition

DEFINITIONS FILE

```
{DEF-SEGMENT SEG-B}
  {BEGIN-DESC}
    THIS SEGMENT WILL
    REPLACE SEGMENT
    AAA IN CONFIGURA-
    TION CONFIG-B
  {END-DESC}
  {BEGIN-SOURCE}

      -------------
      -------------
      -source  code-
      -------------
      -------------

  {END-SOURCE}
{END-DEF}

{DEF-CONFIG CONFIG-B}
  {SOURCE}
  {USE-CONFIG CONFIG-A}
    {REPLACE (SEG-A) (SEG-B)}
  {END-USE}
{END-DEF}
```

ARCHETYPE FILES

ENCODER

ARCHETYPE
FILE FOR
SOURCE
SEGMENT
SEG-B

ARCHETYPE
FILE FOR
SOURCE
CONFIG-A

SEGMENT
SEG-A
REPLACED
BY
SEG-B

ARCHETYPE
SOURCE
FILE FOR
CONFIG-B

FIGURE 4: A Definitions File with a
"Use-Configuration" Statement is Encoded.

300

is complete, the outer definition processing continues as if the nested definition never existed. Therefore, text does not have to be moved to define a segment. Transforming existing text files into definitions files is fast and easy because of this feature.

3.2.   THE USER INTERFACE

The user interface of the SGCM system is a DARTS program that allows a user to generate and manage software. There are four main modules:

1). System Services
2). The Encoder
3). The Manufacturer
4). The Manager

3.2.1.   System Services

The system sevices module allows the user to access the outside system commands such as print, directory, purge, delete, and edit. Each command behaves inside the user interface as it does outside. For example, a definitions file is created with the editor either inside or outside the user interface.

3.2.2.   The Encoder

The encoder creates an archetype file for every configuration and segment defined in a definitions file. It also updates the manager knowledge bases with the new information.

3.2.3.   The Manufacturer

Given a configuration name, the manufacturer produces a final text product from the corresponding configuration archetype file. Other archetype files are accessed if specified by AXE "jump" statements. The manager knowledge bases obtain the correct archetype filename for each specified configuration and segment. They also keep track of segment deletions, replacements, and appends for the "use-configuration" statement.

301

3.2.4.   The Manager

The manager obtains information about configurations and segments from the manager knowledge bases. For each configuration and segment, the manager provides date and time created, archetype filename, configurations and segments used and that use it.   When a configuration uses another configuration, all delete, replace, and append information may be obtained.   The manager also marks as obsolete any segment or configuration that uses either a changed or obsolete configuration or segment.   The manager can list all obsolete segments and configurations and tell why each has been marked as obsolete.   This tracking feature allows for easy maintenaince of configurations and segments.

4. SUMMARY

Archetyping saves text.   Many different text products can be generated from just one archetype file by manipulating knowledge bases. For large software systems, however, manually creating and maintaining archetype files is impractical. The SGCM system automates the archetyping process and provides a simple but powerful user-definition language.   Users produce definitions files from scratch or from existing text files.   The encoder uses a definitions file to create archetype files and update the manager knowledge bases. Text products are manufactured from the archetype files with direction from the manager knowledge bases. The manager provides maintainance information for the user.   The SGCM System can generate and maintain complex software systems quickly and easily.

A SCHEDULING AND RESOURCE MANAGEMENT SYSTEM
FOR SPACE APPLICATIONS

Daniel L. Britt
Amy L. Geoffroy
Martin Marietta Denver Aerospace

John R. Gohring
Martin Marietta Data Systems

INTRODUCTION

Every spacecraft, whether in orbit around the earth or on a deep-space flight, has at its disposal limited amounts of the resources required for it to accomplish its mission. The availability of these resources depends upon spacecraft internal systems as well as its orientation, location and environment. A mission typically will involve the operation of a set of experiments along with spacecraft housekeeping, control, propulsion, energy generation, collection and storage, etc. The environment surrounding a spacecraft, especially one orbiting the earth, is continually changing, and conditions under which many operations can be carried out are present only periodically or rarely. Because of this, most space missions are planned in detail long in advance, in an attempt to accomplish as much as possible while staying within resource budgets. Such advanced planning first involves deciding what the goals of the mission are and what operations must be carried out to achieve them, then scheduling activities to accomplish these operations. Activities must be scheduled so that they have the necessary resources and conditions available to them, do not interfere with one another and do not endanger the mission. As a mission progresses its goals may change, subsystems may become incapable of operation, or other unforeseen situations may arise. These changes require on-the-fly revisions to the schedule of experiment and subsystems activations.

Activity scheduling is currently a costly, human-intensive task which requires a great deal of expertise. It belongs to a class of problems whose complexity increases exponentially with the number of operations. NASA has in the past accomplished this task by using a great deal of manpower, a large number of negotiating sessions, interminable bouts of phone tag, and mountains of paperwork. Lately the situation has improved with the introduction of automated scheduling techniques, but these to date still require expert involvement and fall short in some important ways. This paper introduces a prototype activity scheduler, MAESTRO, capable of meeting the needs of many NASA missions, eventually to include Space Station. We first discuss the approach to resource-constrained scheduling, then describe the intended domain for MAESTRO, its design and current capabilities, and conclude with a description of planned enhancements and revisions to the system.

FOUNDATIONS OF APPROACH

MAESTRO is designed to operate in the domain of resource-constrained scheduling problems. In resource-constrained scheduling activities are scheduled subject to the availability of the resources. In the real world of space applications resources are a limited commodity. A primary criterion for a good schedule in this domain is that the scheduled activities use as much of the allocated resources as possible.

Resource-constrained procedures may be categorized by their approach to the problem and evaluated according to their utility in application. The two main

approaches are optimal methods and heuristics. Optimal methods are designed to produce the best possible solution to the resource utilization problem. These include linear programming and other mathematical models. While a proveably optimal schedule is by definition the best solution, there are limitations on where optimal methods may be utilized, as we will discuss below. In contrast, the heuristic approach can quickly produce good (but not necessarily optimal) activity schedules, approaching maximal consumption of the available resources.

Several problems may preclude the use of optimal methods for resource-constrained scheduling problems. First, uncertainty in the predicted environment may make optimal schedules infeasible. Optimal scheduling methods assume an apriori view of the environment, where exact knowledge of durations and resource levels is possible. The real world of spacecraft operations is uncertain. Experiments may run longer or shorter than anticipated and the resources themselves may fluctuate based on conditions external to the spacecraft. The scheduler must be able to react to these uncertainties in order to produce and maintain good schedules. Optimal methods currently have no facility for dealing with such uncertainties in a timely fashion.

A second major obstacle for the use of optimal methods is the size of the scheduling problem, both in terms of the number of activities to be scheduled, and the number of resources which must be simultaneously allocated in order to satisfy operational requirements. Typically the process of producing an optimal schedule is time consuming, requiring an exhaustive analysis of the search space which grows exponentially with the number of activities and resources. The computational impracticability of determining optimal schedules is definitly evident in resource-constrained scheduling. Weist, one of the fathers of linear programming, spoke of the impracticallity of solving resource-constrained scheduling problems utilizing the mathematical approach. He showed that a 55 activity network with 4 resource types required 5000 equations and 1600 variables. Davis points out that "These mathematical optimization procedures are capable of handling far less complex problems than the heuristic category."

Finally, optimal methods require that optimality be strictly defined. It is rarely known in complex problems if the schedule produced is the optimum. This would require that the predefined evaluation criteria be matched against every possible schedule, which is hardly practical. This is especially true of the MAESTRO domain where the precedence relationships between activities are minimized and resources are the constraining factor, providing a massive explosion of possible schedules that may be produced during a particular scheduling period.

Resource-constrained scheduling problems are amenable to heuristic problem solving, utilizing scheduling rules-of-thumb that are capable of producing reasonably good schedules, which may not be optimum. The heuristic approach differs from optimal methods in that it is not necessary to search the space of all possible schedules, but to incorporate techniques that will assist in shrinking the problem to manageable proportions. Heuristics utilized to select activities for scheduling may produce differing quality schedules based on the specific conditions, activities and resources for each scheduling period. Experience has shown the heuristic approach can produce schedules in a timely fashion permitting the option of producing several good schedules in order to select one that may be "best". It is important, due to changing mission requirements and spacecraft management goals, that the scheduling heuristics be readily modifiable in order to produce potentially differing schedules for evaluation.

It is likely that it will prove fruitful for researchers to pursue work on the problem by developing both the mathematical and heuristic-based approaches. A truly robust

304

system will probably evolve utilizing the strong points of both. MAESTRO has been implemented utilizing the heuristic approach to resource-constrained scheduling.

MAESTRO DOMAIN

The scenario in which MAESTRO is intended to operate is that of a spacecraft which supports a set of experiments. An experiment's activations are constrained by resource availability, environmental conditions, spacecraft position and orientation, and sequencing restrictions imposed by mission goals and requirements.

Resources can include electrical power, thermal energy rejection, crew, data processing and storage, several types of communication, and consumables such as water and liquid nitrogen. Some of these are both quantity-controled and rate-controled, i.e. there is a maximum rate at which they can be expended and a maximum amount which can be consumed over a scheduling period. Electrical power is an example of this type of resource.

Environmental conditions include day and night, temperature, composition of gasses near the spacecraft, various targets, vibrational stability, etc. These are similar to quantity and rate controled resources except that they are available on an all-or-nothing basis. Each activity which requires a certain condition (e.g. day cycle) will be simultaneously satisfied (with respect to that requirement) by the existence of that condition.

Sequencing restrictions among activities include the temporal relations identified by Allen in his work on temporal reasoning, such as before/after, during, overlapping, etc. These may be imposed by mission requirements or activity characteristics.

The MAESTRO scenario includes the existence of a predeveloped schedule which is to be followed to some extent, a currently active schedule of operations which are being executed while the new schedule is being developed, and Crew or Mission Control directives which alter the schedule, under development or currently active.

Activities begun during one scheduling period may carry over to succeeding periods indefinitely. Activities are considered to be composed of one or more subtasks which have sequencing constraints among them. Each activity typically is performed many times over the course of a mission. It is assumed that the success of a particular activity may depend upon the number of performances of it that are executed. Individual performances of activities can be categorized by whether they are continuous, intermittent or one-shot, and by whether once started they can be interrupted and restarted with little impact on their overall success.

Some activities may require either of two resources, but not both. The availabililty of a ground station for communications may obviate the need for data storage, for example. In addition, resource use by an activity or subtask may vary over its duration. Activities are sometimes considered to be producers of a resource as well as consumers, and can be producers of conditions not acceptable to other activities. Most spacecraft subsystems are examples of the former, creating or regulating resources necessary to other activities, while an experiment which produced vibrations would be typical of the latter.

It is assumed that each activity will have a priority, or importance attached to it, and further that these priorities change over the course of the mission or even a single scheduling period. For instance, a non-restartable experiment will acquire a higher priority after a large amount of time has been invested in it, especially if its success is all-or-nothing, for example.

305

CURRENT MAESTRO IMPLEMENTATION

At present MAESTRO is capable of handling only a subset of the scheduling problem as described above. The current implementation is described in more detail in the next sections. In general, given a list of activities and a set of resource availability and operating conditions projections, MAESTRO will place as many performances of as many activities as possible on a schedule. In the event of a change in resource availabilities or the conditions under which activities are running, the scheduler can decide which, if any, activities need to be unscheduled and which can be rescheduled at different times.

ACTIVITIES

An activity is typically an experimental payload, with certain requirements for power, crew time, ambient temperature ranges, etc. Activities may also include non-payload events which require particular resources or environmental conditions (eg. crew lunch requires some power, a crew member, etc). Currently, MAESTRO offers a set of 200 activities from which the user may select any subset for scheduling. The activities are models of payloads which appear in the Mission Requirements Data Base (MRDB) for space station, maintained by Johnson Space Center. An activity description contains its resource utilization profile, constraints on operating conditions, a measure of its importance, a number of performances requested and other characteristics. After the initialization of the scheduling system, each activity also contains a description of its possible opportunities to be scheduled - intervals on the timeline when its resource requirements and operating constraints can be met.

USER CONTROL

MAESTRO allows variable levels of user intervention. The user can request that the scheduler put one or more activities on the schedule, or can himself put activities on the schedule, inspect their opportunities to be scheduled, etc., before, during and after the scheduling process. He can also allow the scheduler to run completely automatically, in which case it will schedule activities until there are no more which can be scheduled. The user may deschedule a performance of an activity. He may query the system regarding remaining opportunities for an activity, or for profiles of the remaining availability for the various resources. Resource availability is also user alterable. The level of interaction and intervention is thus determined by the user.

AUTOMATIC SCHEDULING

Whenever an activity is placed on the list of those requested for scheduling, either through system initialization or as a result of any activity being unscheduled, its opportunities to be scheduled are calculated. Then, when the scheduler attempts to schedule an activity, it first selects a small subset of the outstanding scheduling requests by comparing opportunities and priorities, then looks at the opportunity rating, priority and percent of performance requests satisfied for each activity in this subset (its success rating), and selects one which is deemed most "interesting" to schedule next. An interesting activity is one which has few opportunities in the current scheduling period, has high priority relative to other activities requested, and/or has accrued relatively little success compared to other activities. Upon making this selection the scheduler uses a set of scheduling heuristics to determine where within the activity's opportunities to schedule a performance of it. Scheduling heuristics can include earliest-possible, resource-use-leveling, crew preference, and more sophisticated methods such as opportunity comparison, among others.

306

RESOURCE UPDATE

When the scheduler or user has put a performance of an activity on the schedule it then updates resource availabilities based on what resources the activity will use, then recalculates opportunities for all activities. A check is performed at this point to determine if the scheduling of that activity caused there to be other activities which have no opportunity to be scheduled. If a high-priority activity has been rendered unschedulable the user can remove one or more previously scheduled lower-priority activities and schedule the higher-priority item.

CONTINGENCY SCHEDULING

After a schedule has been created there can arise circumstances under which the schedule becomes invalid, usually as a result of a change in resource availability or spacecraft conditions. In this event, the scheduler first updates its resource availability estimates, then checks to see if any resource has become over-used. If so, the scheduler will proceed to unschedule activity performances which can impact the availability of an affected resource until the availability of that resource is again within acceptable limits. The order in which performances are unscheduled depends upon both the priority of the activity and the degree to which the unscheduling of the performance will affect the resource situation. Activities whose resource use profile better fits the resource over-use are selected first.

After any unscheduling that may be necessary in a contingency situation, the scheduler will again recalculate opportunities for all activities, and then gives the user the same scheduling options as before. This rescheduling is allowed in the event of an unexpected increase in resource availability as well.

EVALUATION

A simple evaluation function will calculate total percent resource use and percent of performance requests satisfied in each priority class, combining these into a single rating for the schedule which ranges between 0 and 100.

INTERFACE CAPABILITIES

The user interacts with MAESTRO through the use of the display device, the keyboard and the mouse. The interface is composed of a collection of screens subdivided into panes. Three major screens are currently available: the base, schedule, and resource screens.

The base screen contains panes that graphically display the schedule, chart resource utilization, and provide user interaction messages. The schedule screen provides a larger schedule display area by eliminating the resource pane. The resource screen provides utilization graphs for each of the 12 resources defined in the system. Each screen contains command menus to assist the user in operating the system.

The schedule and resource utilization panes are viewports into a larger background image. Panning within the larger image is supported through a menu of scrolling commands or actual mouse activation on the pane. Objects on the schedule pane, such as schedule bars, activity names, and opportunity indicators, are mouse sensitive and provide the user additional modification and interrogation capabilities.

The user may interact with the scheduler to manually or automatically unschedule and schedule performances of activities. Additionally s/he may affect resource availability through interaction with the resource manager. The display of resources provides the capability to graphically depict actual availability, consumption or percentage of utilization. The scheduler is at the user's command through menu

307

options for querying the status of the schedule, evaluating a schedule, specifying the duration of the scheduling period, choosing activities to schedule, manipulating the size of the schedule display, and handling contingency situations. Feedback from the system to the user is provided whenever any action is requested.

REPRESENTATIONS USED

MAESTRO is programmed in Zeta-Lisp, utilizing FLAVORS on a Symbolics 3640. In order to provide natural and flexible representations within the system, object oriented techniques are utilized to represent the basic components of the system - activities, resources, etc.

Interval-based representations are utilized for describing time-varying objects. For example, the resource attribute representing current availability versus time is the triplet quantity-start-end, indicating that "quantity" is available from time "start" until time "end". Whenever a resource availability update occurs new triplets are generated and then coalesced to include as few as necessary to represent availability over the entire scheduling period.

SYSTEM PERFORMANCE

Test data includes two-hundred activities, averaging five performance requests each, totalling around one-thousand performance requests. These activities use twelve different limiting rate-controled resources, eleven quantity-controled resources and two-hundred pieces of equipment, and are requested for scheduling over a twelve-hour period. On a Symbolics 3640, the system can currently schedule these in roughly ninety minutes. Given only forty activities, or two-hundred performance requests, the system requires about eight minutes to complete a schedule, which is consistent with the exponential time complexity of the scheduling problem. MAESTRO can revise a schedule in anywhere from a few seconds to several minutes depending upon the magnitude of the change in conditions. When used for mixed manual and automatic scheduling, the time a user has to wait for the completion of a command will typically be much less than that for the automatic development or revision of a complete schedule. A marked performance improvement has been observed on a Symbolics 3675 with a larger main memory.

FUTURE ENHANCEMENTS

MAESTRO has been under development for approximately six months, utilizing two researchers for its implementation, and as such is still in its infancy. Our list of desired capabilities far outstrips those of the existing system and may not be practical any time soon, but many items from the list are intended to be implemented in MAESTRO within the next several months. Our intent here is to indicate what a very powerful scheduler might be capable of.

HIERARCHICAL REPRESENTATIONS

- Consider activities to be components of other activities, and have various subcomponents themselves.

- Allow various degrees of specificity on the temporal constraints among activities at all levels, and use them to constrain the scheduling of activities.

- Allow resource requirements of activities to be described at various levels of abstraction, such that an activity could specify a need for a crewmember, or a payload specialist, or Colonel Smith, for example.

308

- Allow multiple alternative schedules covering various time scales, with their associated resource use and conditions data, which can be inspected, compared and used to generate other schedules.

- Make use of a Ground-generated weekly timeline and the currently active schedule in scheduling the next shift.

SCHEDULING INTELLIGENCE

- Take into consideration all relevant factors in deciding which activity to schedule next, instead of just priority, opportunity and success.

- When calculating opportunity, consider sequencing constraints as well as resource and conditions availability.

- When deciding where to schedule an activity, consider the opportunities of various other activities, possibly scheduling several at once to make sure they do not conflict.

- Use some form of backtracking to undo scheduling decisions which cause activities not to be schedulable.

- Allow the user to select which scheduling heuristics to use, and in what order(s).

- Explain to the user why a particular scheduling request could not be met, and suggest alternatives with their relative merits and problems.

BREADTH OF DOMAIN COVERAGE

- In addition to the experiments include all of the subsystems aboard the spacecraft in the database of activities.

- Allow the scheduling of activities whose durations are variable.

- Characterize performances as continuous, intermittent or one-shot, and as restartable or nonrestartable.

- Handle resource use trade-offs, such as using TDRSS if available and using data buffering if not.

- Interface the scheduler with other automation software such as resource systems controllers.

- Summarize priority of activites scheduled and communicate with high-level resource allocation manager.

- Display a list of activities using a resource at a certain time, and a description of the resource use by an activity over time.

- Track the use of and plan for resupply of critical consummable resources.

CONCLUSION

As noted above, MAESTRO is a prototype scheduler designed primarily for the Space Station operational scenario. An attempt has been made throughout the development effort to use general and powerful representations, algorithms and heuristics such that the system could be converted with minimal effort to handle scheduling in a variety of other domains. The scheduler could be used as a training tool for human schedulers by making use of the what-if capabilities and evaluation functions to find good scheduling heuristics and methods. The system can be used to schedule activities in any domain wherein activity durations are not a function of the resources made available to them, and wherein sequences of operations and the constraints on them are known in advance. Examples of other potential applications include airline flight scheduling, robotics operations scheduling, factory inventory management and job scheduling, and communications scheduling. Each of these applications would require revisions to MAESTRO, the magnitude of which depends upon the similarity of the problem to Space Station activity scheduling.

REFERENCES

Allen, J.F., Koomen J.A., "Planning using a Temporal World Model", *Proceeedings of the Eighth International Joint Conference on Artificial Intelligence,* Karlsruhe, West Germany, 1983.

Baker, K.R., *Introduction to Sequencing and Scheduling,* John Wiley & Sons,1974.

Bechtel, R.T., Weeks D.J., "Autonomously Managed High Power Systems", *Proceedings of the 20th Intersociety Energy Conversion Engineering Conference,* Miami Beach, Florida, August 1985.

Britt, D., "The Role of Expert Systems in Space Station Power Management" *Proceedings of the 21st Intersociety Energy Conversion Engineering Conference,* San Diego, California, 1986.

Davis EW, "Networks: Resource Allocation", *Industrial Engineering,* 1974. Gohring J.R., Lewy D.,Sauers R., "EMES: An Expert System for Spacecraft Energy Management", Conference on Intelligent Systems and Machines, Rochester, Michigan, 1984.

Loomba, N.P. *Linear Programming, A Managerial Prespective,* Macmillan Publishing Company Inc. New York, 1976.

Miller, D., "Scheduling Heuristics for Problem Solvers", Yale Research Report #264, 1983.

Sathi A.,Fox M.S.,Greenberg M., "Representation of Activity Knowledge for Project Management", *IEEE Transactions on Pattern Analysis and Machine Intelligence,* September 1985.

Vere, S.E., "Planning in Time:Windows and Durations for Activities and Goals", Technical Report, Jet Propulsion Laboratory, 1981.

Vere, S.E., "Temporal Scope of Assertions and Window Cutoff", *Proceedings of the Ninth International Joint Conference on Artificail Intelligence,* Los Angeles, California, 1985.

TECHNICAL PAPER

AN EXPERT SYSTEMS APPLICATION TO SPACE BASE DATA PROCESSING

by

Stephen M. Babb
General Dynamics
Space Systems Division
San Diego, CA 92138
(619) 573-8530

for presentation at the

CONFERENCE ON ARTIFICIAL INTELLIGENCE FOR SPACE APPLICATIONS
HUNTSVILLE MARRIOT
HUNTSVILLE, AL

Conference Date: November 13-14, 1986

AN EXPERT SYSTEMS APPLICATION TO
SPACE BASE DATA PROCESSING

Stephen M. Babb
General Dynamics
Space Systems Division
San Diego, CA 92138

ABSTRACT

The advent of space vehicles with their increased data requirements are reflected in the complexity of future telemetry systems. Space based operations with its immense operating costs will shift the burden of data processing and routine analysis from the space station to the Orbital Transfer Vehicle (OTV). This paper describes a research and development project which addresses the real time on-board data processing tasks associated with a space based vehicle, specifically focusing on an implementation of an Expert System.

INTRODUCTION

Currently under development at General Dynamics / Space Systems Division is an expert system that processes space based vehicle telemetry data. The goal of the research project is to produce an expert system that will ultimately reside on board a flight vehicle. This particular system receives telemetry data from the vehicle and applies various rules to infer Out of Tolerance Conditions (OTC). In it's ultimate realization, only the essential information will be transmitted to the space station; the balance will be retained on board for use in other expert systems which monitor for long term trend analysis and perform mission planning.

Large amounts of performance and status data generated by a contemporary telemetry system will be processed by the proposed expert system. A telemetry system was chosen for study and analysis because future vehicles are expected to have in excess of 500 distinctive measurements and it will be physically impossible

to employ specialists to analyze this quantity of data at the space station because of budget constraints as well as the risks involved supporting human life. Our goal is for the expert system to interpret this data into information of immediate concern to the OTV operator.

It was recognized early on that an orderly development of this expert system would consist of several phases (Fig 1). The initial phase is the development of a non-real time prototype demonstration system with a knowledge base that represents or characterizes several propulsion related measurements aboard a ground launched Atlas vehicle (Fig 2). During this phase several processing techniques were studied, all of them were computer based. During the second phase of the development program, a real time expert system is to be operated in a ground based environment, in parallel with conventional operations, performing validation and verification functions that instill confidence that the computer program does indeed function properly. The expert system would remain transparent to the user, although access to the rule base would be close at hand. The final phase of the project would involve the movement of the developed expert system from a ground based environment to that of a space based vehicle. This transition is planned to be virtually unnoticeable to the end user. Ultimate realization will result in the reduced code being embedded in a computer on board the spacecraft or OTV.

Described in this paper is a demonstration prototype expert system titled Software and Hardware ExpeRt system Analyzing Realtime Data or SHEPARD; which was developed on a Symbolics 3640 computer, running ART (Automated Resource Tool) based in Lisp.

SHEPARD- A demonstration prototype

The SHEPARD expert system is a first cut demonstration prototype program to assess previously recorded flight vehicle data. Early on in our research it became obvious that the concept of utilizing artificial intelligence to accomplish analysis of data would require a demonstration to show its feasibility. After several false starts, it was decided that a small set of closely related vehicle measurements would be used to develop our knowledge base. Again it was felt that a small but comprehensive set of propulsion and related vehicle measurements would adequately

allow us to experiment with the concepts involved with artificial intelligence while forming a credible expert system that would highlight the critical concepts.

Flight data is stored within the program itself in the form of a "callable" ART file, and is read in sequentially with an associated time tag in increments of two seconds. The system has two fundamental modes of operation - automatic increment and continuous. In the auto increment mode, each time slice is called and read at the two second time interval, running until the data is exhausted. If one were to select the continuous mode of operation, the data is read in as fast as the host processor allows, again operating until all data is read.

As a part of our validation and verification exercise, we utilized two sets of flight data, one which contained nominal parameters and yet another which had an observed anomaly greater than 3 σ. This tolerance was chosen simply because we desired to look at gross preturbances, while ignoring those transients which were currently not of interest to us.

As part of the demonstration system, the observer is shown a status window, which displays the latest anomaly (if any), the time of such event and the value (Fig 3). Also displayed will be a popup window which charts the time history of the particular measurement and its upper and lower operating limits; the rules built into the system will fire when the proper rules match the incoming data, causing suggested courses of actions to be announced to the operator. To give the human operator additional confidence in the operation of the expert system a listing of all the measurements utilized in the data base are displayed and selectable by the operator using the mouse.

Summary - Future Considerations

During the upcoming year several intermediate milestones have been identified which will chart our progress toward the ultimate goal of realizing a fully functional on board realtime data analysis expert system. Next years tasks are as follows:

o Increase the depth of analysis (granularity) to include transients

o Supplement the current measurement set to insure a comprehensive subsystem measurement set.

o Relocate the expert system to a new host environment on a Apollo workstation.

o Utilize an updated version of ART, operating in the C language.

This is the first year of a multiyear research and development effort has been deemed successful, largely in part to the development of a demonstration prototype expert system SHEPARD. It has been shown that the program is indeed a potential solution to the problem, and it is justifiable because the task is performed by a computer in a hostile environment and lastly it is appropriate because AI lends itself very will to problems requiring symbolic manipulation which are generally heuristic in nature.

THREE PHASE LONG
RANGE DEVELOPMENT PLAN

CURRENT   INITIAL   ULTIMATE

FIGURE 1

SSD 6046

316
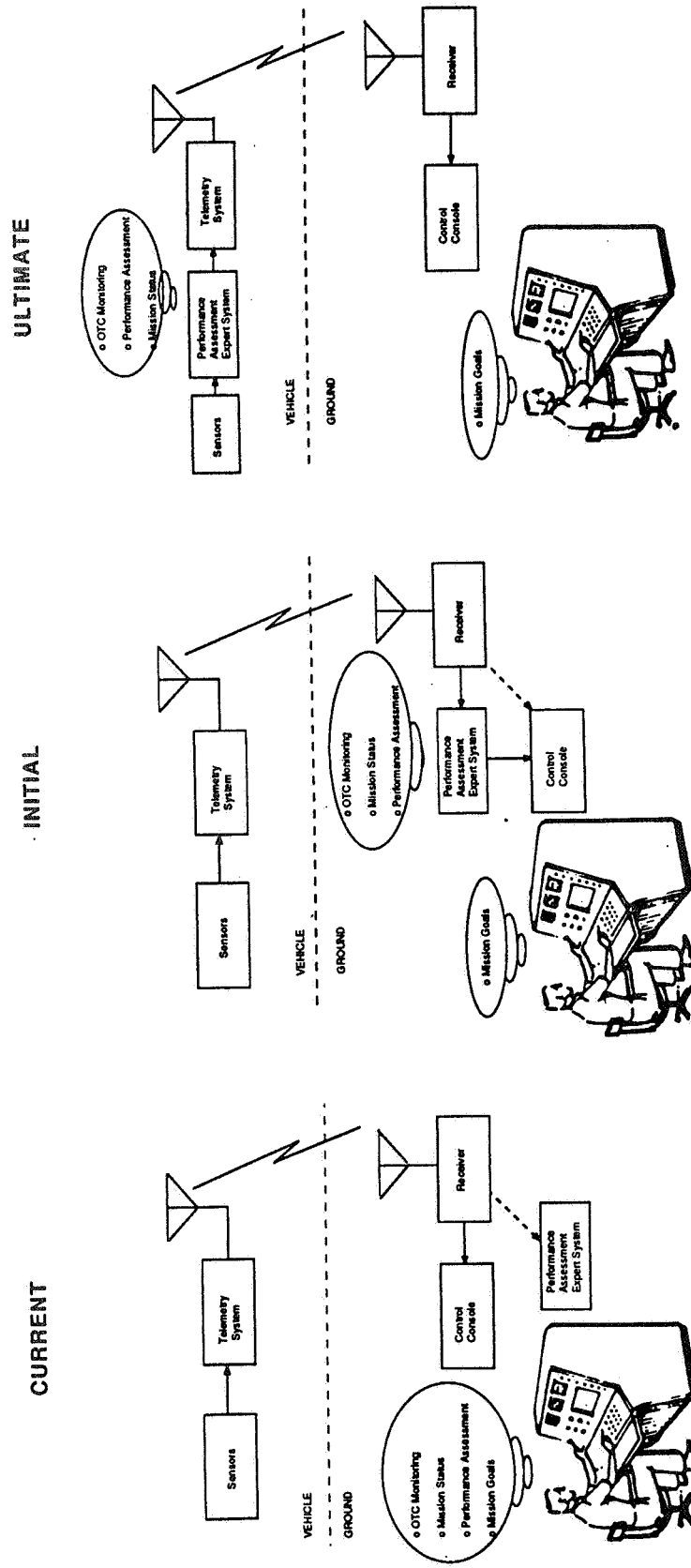
DEMONSTRATION PROTOTYPE OF
A PERFORMANCE ASSESSMENT EXPERT SYSTEM

SYMBOLICS

3640

ART Shell

PERFORMANCE
ASSESSMENT
EXPERT SYSTEM

ART.FILE

User

SGG Combustor Pressure
Sustainer Pump Speed
Sustainer Engine Thrust Chamber Pressure
Vehicle Axial Acceleration
B1 Engine Yaw
B2 Engine Thrust Chamber Pressure
B1 Engine Thrust Chamber Pressure

FIGURE 2

317

SSD 6046

FIGURE 3

318

SSD6046
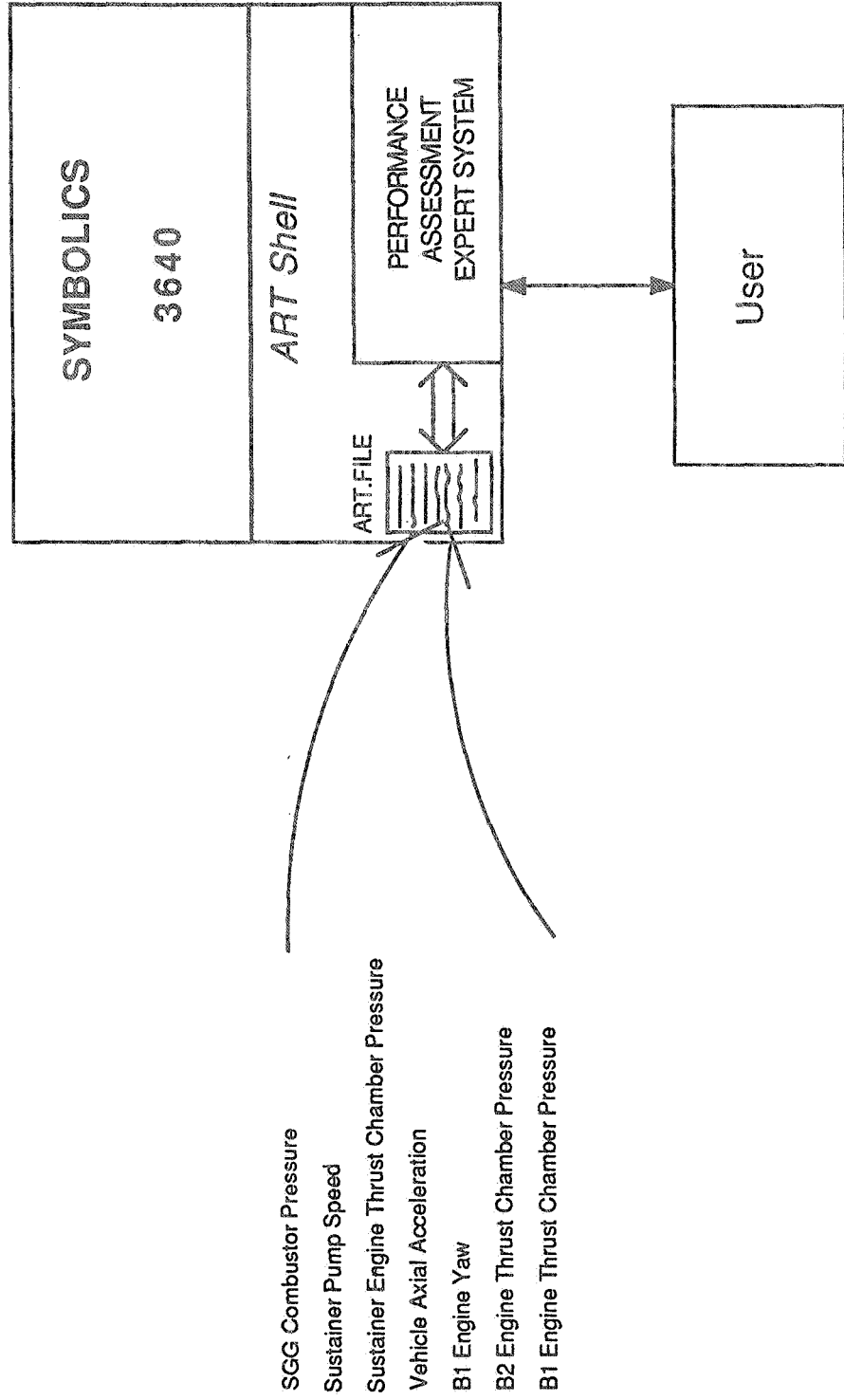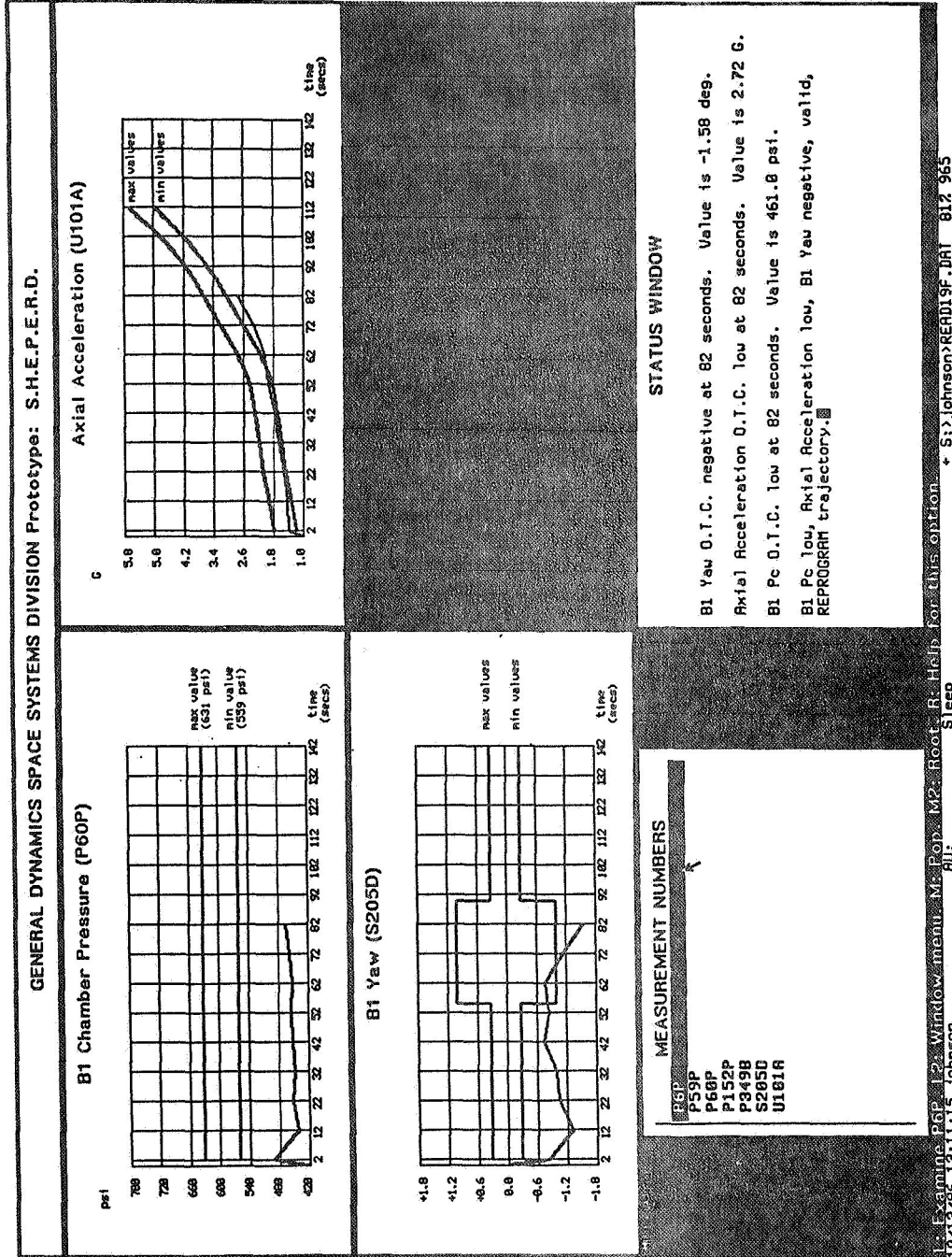
Intelligent Resource Management for Local Area Networks:
Approach and Evolution[1]

Roger Meike
Martin Marietta Denver Aerospace
Space Station Program
P.O. Box 179 (MS D1744)
Denver, Co. 80201

Abstract

The Data Managment System network is a complex and important part of manned space platforms. Its efficient operation is vital to crew, subsystems and experiments. AI is being considered to aid in the initial design of the network and to augment the management of its operation. The Intelligent Resource Management for Local Area Networks (IRMA-LAN) project is concerned with the application of AI techniques to network configuration and managment. A network simulation was constructed employing real-time process scheduling for realistic loads, and utilizing the IEEE 802.4 token passing scheme. This simulation is an integral part of the construction of the IRMA-LAN system. From it, a causal model is being constructed for use in prediction and deep reasoning about the system configuration. An AI network design advisor is being added to help in the design of an efficient network. The AI portion of the system is planned to evolve into a dynamic network management aid. This paper describes the approach, the integrated simulation, project evolution, and some initial results.

The Intelligent Resource Managment for Local Area Networks (IRMA-LAN) project is concerned with the application of Artificial Intelligence (AI)techniques to network management. As part of this project, a prototype AI subsystem is being developed in Lisp on a Symbolic 3675 computer to test and demonstrate several of the concepts central to this project. By implementing an evolutionary design schedule, a rapid prototype was quickly assembled and then augmented. This allowed us to have a demonstration of some basic concepts very quickly. As the AI subsystem develops, more complexity is added to it. There are several benefits of developing an AI subsystem in parallel with the application system. These benefits are expected to generalize to other AI subsystems.

Our initial studies identified eight candidate areas for the application of AI techniques to network configuration and management.[2] The areas identified include fault detection, fault diagnosis, fault recovery, fault prevention and fault prediction as part of fault tolerance, and process addition, process deletion and optimization as part of dynamic reconfiguration. Resource evaluation and configuration analysis are important shared tasks within these areas. A key discovery was that these tasks are important to the design of networks as well as to their operation. The ability to evaluate and use available resources efficiently is central to both a good design and smooth operation. This commonality allowed us to develop some of the concepts central to a dynamic network manager in a network design aid. The process we used is described below.

Problems

Currently, no human has the job to manage computer networks in the same way that we hoped the IRMA-LAN system would (i.e. dynamically, while the network operates). There are no real experts, thus, we had to first invent the job and then find out how to be good at it. There *are* people whose jobs are related to network management, and thus would have more expertise and insight into what this hypothetical network manager should do. We used knowledge and basic logic of these pseudo-experts to develop the rules of the AI subsystem.

Some basic problems that can arise when building expert systems are that: (1) experts sometimes lose interest in the your system, (2) don't understand its design, or (3) consider it a very low priority because they expect little gain from it.[3] We became interested in these problems when we decided to build a prototype IRMA-LAN system but had no experts to assist us. To create a true expert system we needed experts who were interested enough to give their expertise in return for immediate benefits. The way in which the project evolves is critical to keeping the interest of the experts. If there is an immediately useful by-product, experts are much more likely to be interested in helping to develop the AI subsystem. Also, by involving the experts in the evolution, they have a much better understanding of the resulting AI subsystem. Our approach was to make extensive use of our rapid prototyping environment to build a subsystem that did not incorporate expertise, but that would be immediately useful to an expert and could evolve to have the expertise of the network management tool originally desired. A design aid was chosen for implementation because it could both help the expert in the network design and develop into the dynamic manager.

The Design

Although the goal of the IRMA-LAN project is to assess the role of AI techniques in network managment, the prototype AI subsystem must demonstrate those AI techniques as well as the environment in which they operate. As much effort is put into the conventional software as is put into the AI portions of the subsystem. In most cases, the user does not know or need to know whether a specific function is implemented using AI or conventional programming techniques. An emphasis of this project is to make a functional subsystem rather than one to demonstrate AI concepts. Integration is very important to this effort. The qualitative model, quantitative model and expert analysis portions interact with each other continuously in IRMA-LAN. The user interface presents these interacting portions as a single subsystem. Parallel interdependent development ensures that the portions of our subsystem will be well integrated. The various interactions of a complete IRMA-LAN system are shown in figure 1. From the start of this project the simulation model, user interface and expert analysis portions of the subsystem have been developing. Early in the development a network interface was added. Thus, all portions of the subsystem are now being developed in parallel.

Simulation Model

Quantitative Model

Configuration Storage
IEEE 802.4 Token Passing Scheme
Real Time Process Scheduling

Object Definition
Simulation Control
Scenario Definition

User Interface

Color Graphics
Interactive Display
Command and Control

Configuration Data
Simulation Results

Real Data
Real Configuration

Configuration Changes

Network Interface

Configuration Retrieval
Statistic Compilation
Control

Recommendation
Optimization

Explanation

Configuration Data
Simulation Results

Configuration Data
Simulation Results
Trend Data

Query

Limited Control

Autonomous Control
Empirical Test

Qualitative Model

Symbolic Modeling
Prediction
Explanation

Prediction

Expert Analysis

Recommendation
Optimization
Configuration Analysis
Trending

Real Data
Real Configuration

Situation Query

Figure 1 - Final phase components and interaction

Over four phases we plan to evolve our knowledge base from a straight forward expert system for network design to a dynamic network management tool. These phases parallel the expertise that we are receiving. Our network management experts are people who design networks and network management programs, not people who manage networks themselves. Therefore the most direct use of their knowledge was to build a design aid. In this case, the AI subsystem is an advisor to a network designer. By abstracting this information to a higher level, we are building a knowledge base of principles involved in the network management problem. From these basic principles we are deriving a dynamic network management aid which will help in real-time network management.

In the first phase, the subsystem was viewed as a network design aid. Network designers deal with large amounts of information. Our application benefits them by presenting these large amounts of information in a clear and concise manner. Using rapid protoyping techniques, we designed a very flexible system which allows the designer to change configurations easily. The first phase design consisted of a computer network simulation model and a preliminary user interface. Network designers could then test new designs and get immediate feedback through color graphics. The simulation model is flexible enough to allow easy changes, yet complete enough to allow reliable feedback to the user. This system is simple, but it still provides an improvement over the pencil and paper

techniques that network designers typically use. Once this system was shown to the designers (potential users), they became eager to provide suggestions to improve it. Because of the rapid prototyping environment, we were able to implement their suggestions quickly and keep their interest.

The second phase involves connecting the system to a network so the designer can interact with a real network as well as the simulated one. In addition to the simulation model and user interface, a network interface is added. This allows for much more detailed designs and tests on real hardware, while maintaining flexibility through the simulation model. We are currently entering this phase of our project.

During phases one and two, the AI portion of the project begins. Expert analysis rules and a qualitative model are developed in parallel with all the phases of the project. As we add functionality to the whole system, we add AI functions as well as conventional ones. The expert analysis rule base is created using vertical slice[4] techniques in which a particular function is explored in detail and integrated into the system. This entails finding a suitable problem that a designer may want to work on and discovering the rules that govern the decision-making process for that particular problem. Using this method, demonstrations of AI potential can be rapidly developed. Some of the reactions to the initial configuration of the system led to the development of a set of fault diagnosis rules that set the framework for the knowledge base and are still a major part of our demonstrations.

Because of the lack of true expertise in this limited domain, a qualitative model is being constructed to allow for deep reasoning that will fill in the gaps in the knowledge base. This portion of the simulation model is being developed from specific cases and logical analysis of the quantitative model.

The primary goal of the AI work during phases one and two is to develop an advisor. This means that the IRMA-LAN system is to be used as a network design tool in which the designer uses the AI portions of the system to help in various portions of the design. In the next two phases this thrust changes. The IRMA-LAN system makes a transition from design tool to management aid. The AI subsystem makes a transition from advisor to actor. The IRMA-LAN system becomes a part of the network management system and reacts to situations in the network rather than queries from a user. Based on these decisions it takes the appropriate actions.

In the third phase the IRMA-LAN system recommends an initial configuration of the network based on various actions of the designer. The system is still used as a design aid, but now it has taken on a more active role. Based on scenarios that the designer runs, and other information it gathers during the design process, it takes an active role in initializing the network for use through the network interface.

In the final phase, the IRMA-LAN prototype acts as a manager which reacts to the network and dynamically interacts with it to ensure efficient operation. This may include allocating processes to processors, *tuning* timers and buffer sizes, reacting to predicted changes in performance, and other active functions. The AI system will have made the complete change from advisor to actor.

Development Stage	AI Function	AI Development	System Function
Phase 1	Advisor	Rapid Prototype Vertical Slice	Design Aid with Simulation Model
Phase 2		Integrate Causal Model	Design Aid with Interface
Phase 3	Actor	Refine Rule Base	Initialization
Phase 4		Real Time	Dynamic Management

Figure 2 - Four phase design

Analysis

The design schedule presented here (figure 2) appears to be applicable to AI subsystems in many different domains. To achieve a truely integrated subsystem, it is a good idea to be involved with the design of the conventional system which the AI subsystem is to augment. Because of the rapid prototyping environments commonly used in AI development, it is possible to help in the design of the whole system rather than just the AI portions. Providing such tools for designers can result in many advantages. First, it allows feedback from experts throughout the design of the system. Experts are actively involved in the design of the AI portion of the system by reacting to it as it develops. Similarly, the AI system can react to and become part of the design. Second, the schedule provides demonstration capabilities very early on in the development cycle. This provides visibility that can assist in integrating AI into the system. Third, it provides a base for testing and growth. The more complete the environment is for testing code, the more reliable the tests are. A tool that is used for design purposes will be well tested by the users and will become a good test ground for new code. Fourth, the knowledge acquired in the design phase may be directly applied to the final system. In many cases, the principles involved in designing a system are central to its operation. AI functions that perform tasks such as abstracting out the causes of certain actions in the simulation model may be just as applicable to monitoring performance in a real system. Fifth, in the case where expertise is not readily available, a qualitative model may help. When there are no specific rules of thumb, a more

323

general qualitative model can help the reasoning process and make for a more complete solution. Knowledge gained in developing a *quantitative* model was found to be very useful in the design of the *qualitative* model that we are currently working on for IRMA-LAN.

Notes

1 The IRMA-LAN project is supported by Martin Marietta Denver Aerospace, Independent Research and Development task D-47S

2 Meike, R. C., "The Application of Artificial Intelligence Techniques to Network Reconfiguration for Space Station", Proceedings: Artificial Intelligence - From Outer Space Down to Earth, October 1985, University of Alabama, Huntsville

3 Some of the problems with dealing with experts are presented in
   Bailey, P. A., and Doehr, B. B., "Knowledge Acquisition and Rapid Prototyping of an Expert System: Dealing With 'Real World' Problems", Proceedings: Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, November, 1986.

4 Vertical slicing technique refers to studying a particular problem in depth, thus taking a vertical slice of the problem domain. The design aid itself is a horizontal slice of the whole system under development, but the knowledge base (at least at first) is developed one problem at a time. For more information on vertical slicing see
   Gates, K. H., Adelman, L., and Lemmer, J. F., "Management of AI System Software Development for Military Decision Aids" IEEE Symposium on Expert Systems and Government, Washington D.C., 1985.

324

Distributed Cooperating Processes

in a Mobile Robot Control System

Thomas L. Skillman, Jr.

Intelligent Robotics and Perception

Advanced Technology Center for Computer Sciences

P.O. Box 24346

Seattle, WA 98124

Introduction

A mobile inspection robot has been proposed for the NASA Space Station. It will be a free- flying autonomous vehicle that will leave a berthing unit to accomplish a variety of inspection tasks around the space station, and then return to its berth to recharge, refuel, and transfer information. This "Flying Eye" robot will receive voice commands to change its attitude, move at a constant velocity, and move to a predefined location along a self generated path. It will also receive task commands that will require coordination of vehicle motion, camera lighting and control, and interpretation and reaction to its sensory input.

This mobile robot control system requires integration of traditional command and control techniques with a number of AI technologies. Speech recognition, natural language understanding, task and path planning, sensory abstraction and pattern recognition are all required for successful implementation. The interface between the traditional numeric control techniques and the symbolic processing of the AI technologies must be developed, and a distributed computing approach will be needed to meet the real-time computing requirements.

To study the integration of the elements of this project, we developed a novel mobile robot control architecture and simulation based on the blackboard architecture. This paper discusses the operation of the control system, its structure, and some novel implementation details.

Control System Requirements

To aid in the establishment of system requirements, a discussion of the functional elements of a mobile robot control system follows. A functional system overview is shown in Figure 1. Voice commands are recognized, parsed, and passed to the camera control, guidance and control system, or path planner. Camera commands adjust the current zoom setting with the camera actuators. Guidance and control commands adjust the location or attitude of the Flying Eye robot by issuing commands to the thrust actuators. The Path Planner responds to commands to move to a named location by issuing a sequence of commands to the guidance and control system. The effect of the various actuations are simulated in the world and resulting changes are
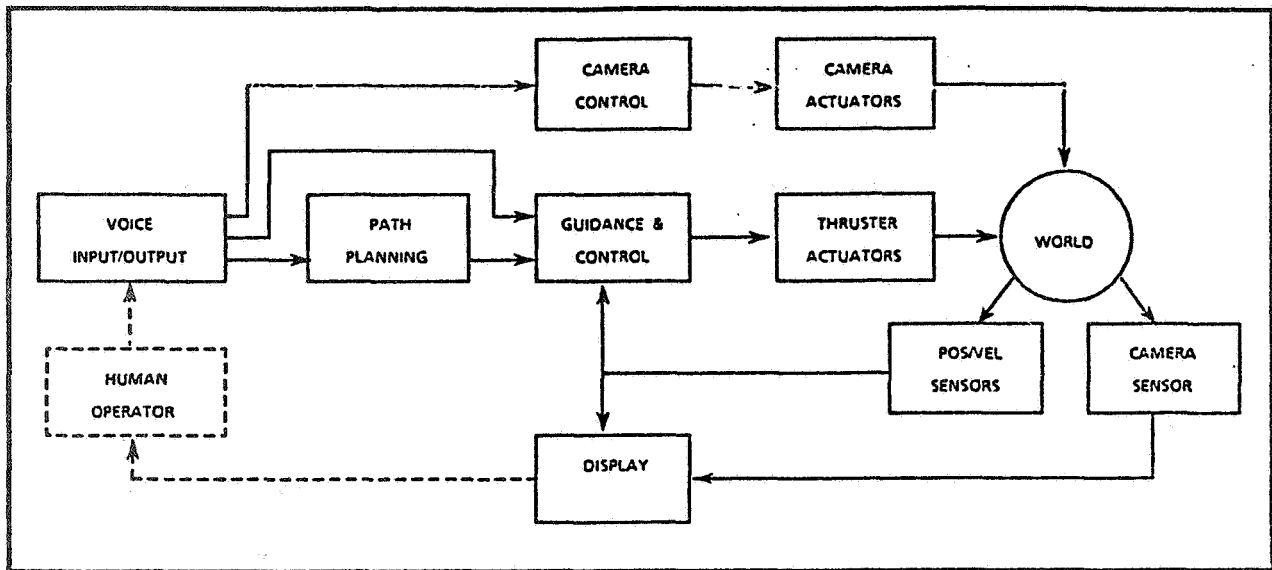
Figure 1 - Flying Eye robot functional block diagram

measured with simulated sensor readings, such as position, velocity, camera image, etc. The sensed data is displayed for the human operator to view and to guide further commands.

The system functions were broken down into a set of independent cooperating processes. The Listener/Talker provides voice input/output to the system. The Path Planner monitors human directed motions and generates safe paths to symbolically named locations. The Command Interpreter translates motion commands from symbols into numeric parameters. The Motion Controller is a closed loop proportional feedback system which compares current to commanded position and velocity, and generates thruster commands to reduce the difference between the two. The World Simulation process takes actuator commands, calculates changes to the modeled world, and generates simulated sensor readings.

These functions are independent subsystems that need shared access to global data and the ability to pass messages. The functions can operate in parallel most of the time, but must be able to synchronize there activities with the other functions. The operations performed by each function involves the reading of information represented at one level of abstraction, performing some computation upon it, and writing out the results of the computation in a representation at a different level of abstraction. For instance, the path planning function's input is a symbolic representation of a desired destination (i.e. Go to the left wingtip.). It must compute a path and then present its results as a sequence of absolute coordinates that the motion control function can accomplish. The motion control function converts absolute coordinates into commands to the thrusters. The different types of processing required by different functions implies a heterogeneous hardware and software implementation environment. Several of the functions must be directly tied to external devices such as the Listener/Talker.

The requirements of the control system architecture are therefore:

1) Support multiple independent processes executing in parallel

326

2) Support a heterogeneous hardware and software environment

3) Support interprocess synchronization

4) Provide I/O support to each process independently

5) Provide a shared data base of state information

6) Support a hierarchical representation of system state information.

Control System Structure

To implement this system, an approach was developed that would support the multiple levels of abstraction and cooperating

processes inherent in the problem definition. Figure 2 shows the relationship between a hierarchical control system and the
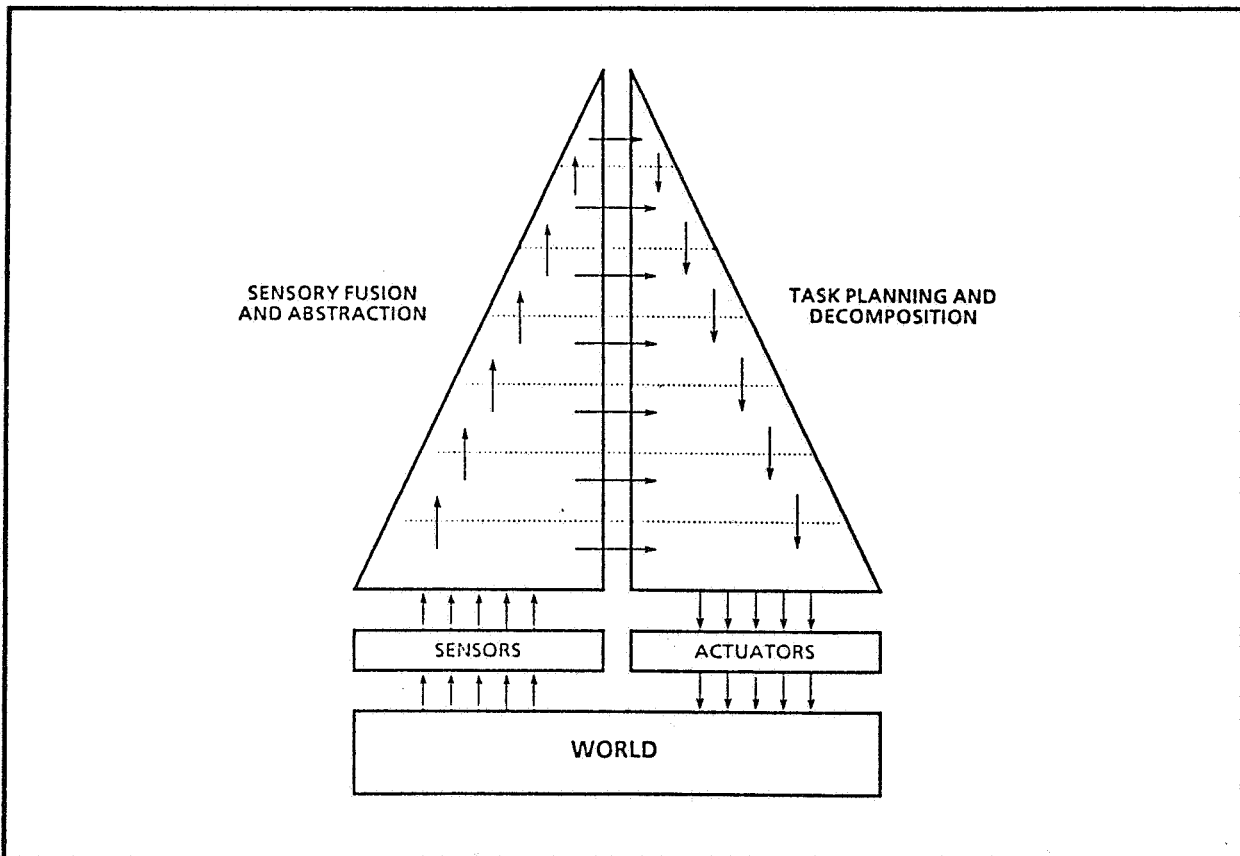
Figure 2 - Hierarchical real-time closed loop control

real or simulated world it is controlling. In our application, the levels of abstractions are 1) sensors and actuators, 2) relative

spatial position and motion information, 3) absolute spatial position and motion information, and 4) symbolic position and goal

information. The sensors in our application are position, velocity and camera image, and the actuators are thrusters and

327

camera zoom. The world we are controlling is a simulation of the dynamics, mass, and geometry of the Flying Eye robot, thruster performance, and an inertial navigation system.

The control system structure must also allow the use of numeric processing languages like C[1] and symbolic processing languages like Lisp[2], OPS5[3], and MRS[4]. Knowing that eventually the Flying Eye robots computing would be distributed over the free flyer, the berthing unit, and a supervisory console, a way to coordinate distributed, loosely coupled, parallel computing was required.

These goals led us to develop a custom control system architecture based on the AI blackboard architecture.

Blackboards: The blackboard architecture has been defined and developed in a variety of applications. It has been a useful tool for solving problems, such as speech recognition in Hearsay [5], ocean vessel tracking in HASP/SIAP[6], and medical diagnosis in PUFF [7].

The three main components of any blackboard system are :

1) The *blackboard*, a shared global database

2) *Knowledge sources,* that embody specialized procedural knowledge and can interact with the blackboard database, but not directly with each other

3) A *trigger mechanism*, which invokes activity in knowledge sources based on changes and transactions with the blackboard database.

All blackboard systems have these components in common, but then diversify in numerous details.

The blackboard architecture provides a means for representing multiple levels of abstraction, and the knowledge sources provide a modular way to implement the control task. To provide multiprocessing capability to the system, it was decided to use the UNIX operating system's multitasking capability to simulate multiple concurrent processes. Local area network technology is used to actually run some processes concurrently on other machines.

To simplify the explanation of our system, it will be compared to the Hearsay II system as described in [8]. This paper provided the basic ideas from which our system evolved.

Our system differs from Hearsay II in that:

1) Preconditions are not used. If a blackboard access causes a triggering event, then the data associated with the triggering event is sent to the knowledge source specified in the trigger pattern

2) Knowledge sources are not instantiated with a local context when the trigger condition is met. Rather, since the knowledge sources are separate processes with their own computing context, only the trigger event is passed to the process.

3) Blackboard monitoring and knowledge source scheduling have been combined into one function. The multitasking operating system (UNIX) is providing for the apparent simultaneous execution of knowledge sources, so monitoring

becomes a matter of listening for blackboard requests and responding to them, and scheduling becomes a matter of deciding in what order to send data created by triggering events to knowledge sources.

4) Process synchronization primitives (tags) at this time have not been implemented.

The key features of our system are:

1) The blackboard, triggering mechanism, and knowledge source handler are implemented with Franz Lisp in a UNIX environment.

2) Knowledge sources exist as separate processes within the UNIX environment and communicate with the blackboard through stream I/O Franz Lisp has a set of convenient commands that give direct access to many operating system features.

3) Because of the stream-based I/O, knowledge sources can be developed with any language, AI tool, or application package, including C, Lisp, MRS, OPS5.

4) All interaction with the blackboard is based on the knowledge source approach, so the user interface is viewed as another source of knowledge and is implemented that way.

5) The knowledge sources are created and maintain their own local computing environment (procedures, functions, variables, constants).

6) UNIX provides a multitasking capability and, hence, the knowledge sources look like independent cooperating processes and the transfer to a true multiprocessor implementation is simplified. (Note: For this to be true, the rate of interaction between knowledge sources has to be small compared to the operating system process scheduling rate.)

Figure 3 shows how the independent cooperating processes, and knowledge sources of our application map into the blackboard architecture. A more detailed discussion of the application can be found in [9].

Blackboard Transactions and Triggering: The interaction between knowledge sources and the blackboard system can be viewed as a form of message passing with pattern-driven routing. If a knowledge source wishes to access data from the blackboard, it sends a message to the blackboard. This message could look like "(read (position ?x))"or "(replace (here-is station1))". The first message asks to read the current value of position stored in the blackboard. The blackboard system will search the blackboard for an entry matching the request and return the result to the knowledge source issuing the request. The second message asks to replace the current "here-is" entry in the blackboard with a new entry "(here-is station1)". A knowledge source can interact with the blackboard with a READ, WRITE, REPLACE, or DELETE request.

A potential side effect of any of the blackboard interactions is that the blackboard triggering mechanism might match against a trigger pattern and cause a trigger event. When this occurs the knowledge source message that caused the trigger event is put on a queue to send to a knowledge source. The trigger list is a set of message patterns associated knowledge source identifiers. Every blackboard message is compared against the trigger list patterns and, if there is a match, the message is queued to be
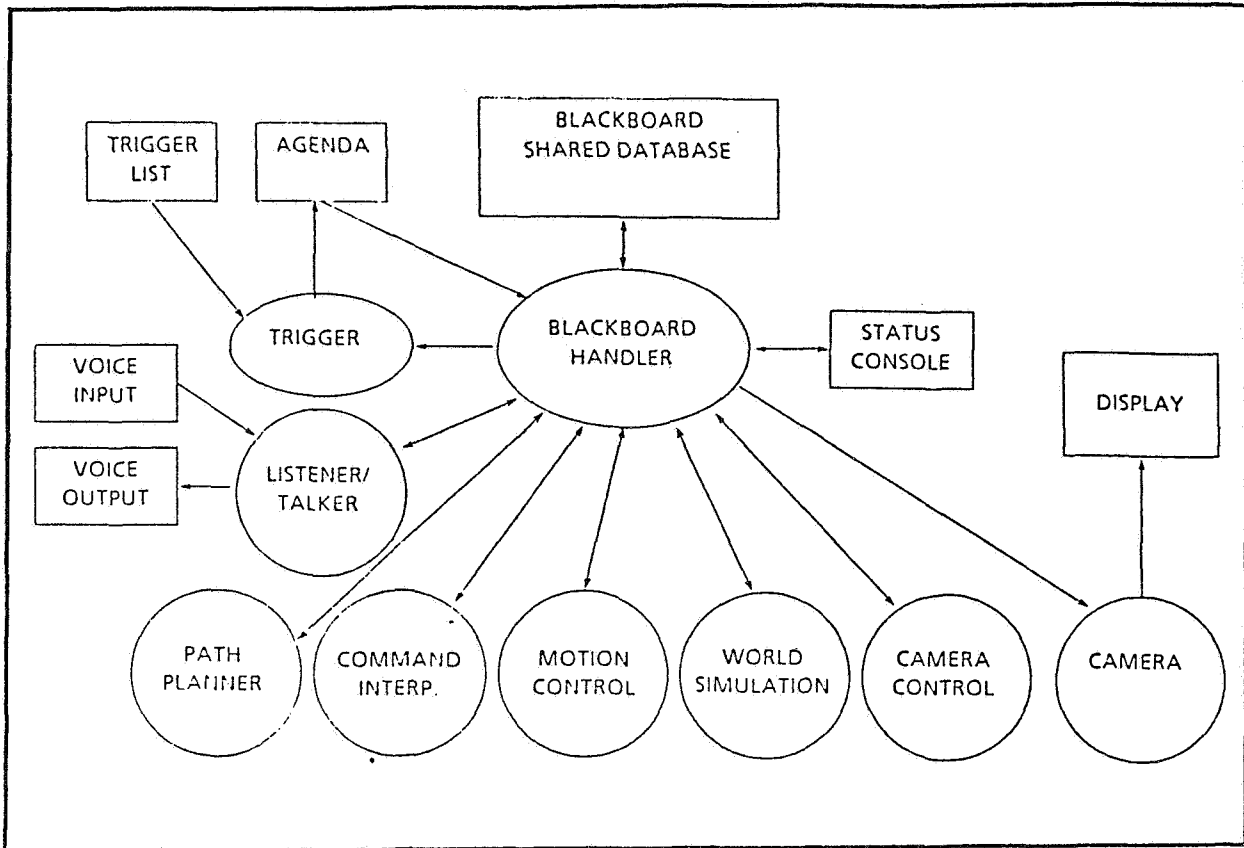
Figure 3 - Mobile robot blackboard architecture block diagram

sent to the identified knowledge source. The queue holds "(message . knowledge source)" pairs and the handler/scheduler sees that the messages are sent to the proper knowledge sources.

Knowledge Source Types: The knowledge sources in our system have a set of distinguishing characteristics. These characteristics are:

control = *master or slave*

trigger = *synchronous or asynchronous*

location = *local or external*

Master knowledge sources interact with the blackboard system based on events in their internal environment. This could be a real-time clock interrupt, I/O with the external world, or the completion of some computation. The knowledge source initiates a read or write from the blackboard and that interaction may or may not result in a triggering event and other knowledge source activity. A slave knowledge source simply waits for a message to be sent to it from a blackboard triggering event and then performs some computation, perhaps posting some result back to the blackboard.

330

Synchronous knowledge sources interact with the blackboard system based on a real-time clock interrupt. This sets performance criteria for the blackboard system because it must be able to respond to the knowledge source's request in a timely manner. Asynchronous knowledge sources are triggered either by external events or by the arrival of a message from the blackboard system. This characteristic is found in Motion Control knowledge source of our system to drive the servo update rate.

A local knowledge source is one that for simplicity or efficiency is run within the blackboard system process environment. Typically these are system initialization, monitoring, and shutdown knowledge source. They are constrained to interact with the blackboard data base in the same manner as all knowledge sources. An external knowledge source is one that exists as a separate process in the computing environment. It can be another process within the UNIX environment on the same machine, or it can be a process running on a remote machine connected via serial or Ethernet communications.

Table 1 show the characteristics of the knowledge sources used in our application.

	Control	Location	Trigger
Listener/Talker	master	external	async
Path Planner	slave	external	async
Command Interp.	slave	external	async
Motion Control	master	external	sync
World Simulation	master	external	sync
Camera Control	master	local	async
Camera Simulation	master	external	sync
Initializer	master	local	async

Table 1 - Knowledge sources and their characteristics

A knowledge source is considered active if it currently exists in the computing environment and can transact with the blackboard. The term *computing environment* is used to extend the concept of a process environment to a distributed set of computing systems and their communications media. An inactive knowledge source is one that is known to the blackboard system but is not currently created and running. If a trigger event is addressed to a knowledge source that is inactive, the blackboard system can create the knowledge source in the computing environment and send it the trigger message.

A knowledge source is typically organized as a block of initialization code that is executed once when the knowledge source is created followed by the body of the knowledge source. The initialization code is used to set up the local process environment

331

and may also interact with the blackboard to get initial values or to set initial values in the blackboard. The body is often an infinite loop that is executed until the knowledge source is terminated.

Table 2 shows the diversity of languages and machines used in our system.

Element	Hardware Host	Software Host	Communication to Blackboard
Blackboard & Handler	Apollo DN660	Franz Lisp	-
Trigger & Agenda	Apollo DN660	Franz Lisp	-
Voice Input	Interstate VRT200	(integral to h/w)	SIO
Voice Output	DEC DECTALK	(integral to h/w)	SIO
Listener/Talker	Apollo DN660	Franz Lisp	Stream IO
Path Planner	Symbolics 3670	Zetalisp	Ethernet / TCP/IP
Command Interpreter	Apollo DN660	"C"	Stream IO
Motion Control	Apollo DN660	"C"	Stream IO
World Simulation	Apollo DN660	"C"	Stream IO
Camera Control	Apollo DN660	Franz Lisp	Function call
Camera & Display	Apollo DN660	Apollo GMR3D/"C"	Stream IO
Status Console	Apollo DN660	Apollo Display Mgr.	Stream IO

Table 2 - Flying Eye hardware and software architecture

Control System Operation

Referring back to the functional block diagram shown in Figure 1 and the blackboard implementation as mapped out in Figure 3, the following discussion provides some insight into the real-time operation of the control system.

Example 1 - "Move left slowly." A time line for the system's reaction to a "Move left slowly" command is shown in Figure 4. At time $t = 1$, the Listener accepts voice input, recognizes words, parses and checks syntax, and translates the command into an internal representation. At $t = 2$, the command interpreter receives the command and performs the necessary coordinate transformations and parameter settings. It then sets the motion control law variables. These variables represent the desired state of the robot system. In this case, it is to be moving at a constant velocity in the left direction as defined by the local
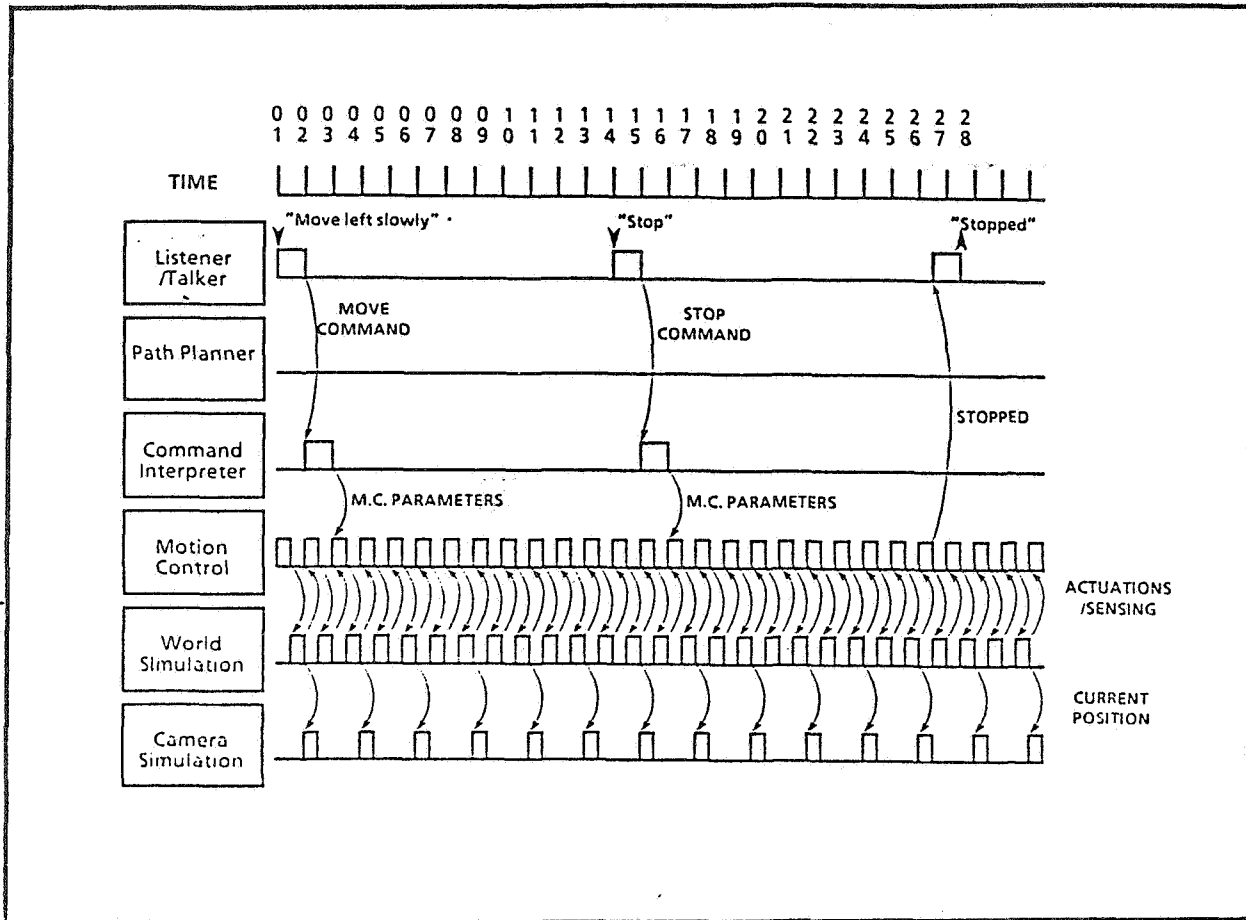
Figure 4 - Time line for "Move left slowly"

reference frame of the Flying Eye look axis. At every tick, the Control Law reads the current control law variables and current sensed data, position and velocity. It then generates a thrust command that will bring the desired state and actual state into agreement. At time $t = 3$, the control law variables are updated by the command interpreter and from that point on the control law generates thrusts to bring the robot to the desired velocity (speed and direction).

At time $t = 13$, a new voice command is issued "Stop." This command is again recognized, parsed, checked, and translated and then passed to the command interpreter. At $t = 14$, the command interpreter interprets the stop command as a desire to be at the current position with zero velocity. It sets the control law variables to represent this and makes the new data available to the control law. At $t = 15$, the control law sees the new control law variables and starts readjusting the thrusts to achieve the goal state; to be at the desired position and have zero velocity. Because the robot was moving at some velocity and does not have infinite thrust capability, it takes some time to achieve this goal. When it does, it observes that it is stopped and makes this fact available to the Talker. At $t = 26$, the talker receives the message and translates it into an English phrase and passes it to a speech synthesis device which emits the message "Stopped."

333

The camera simulation is activated every two ticks. It reads the current robot position and generates a simulated camera view of the world from a 3D graphical database. Future work will use this image for navigation and obstacle avoidance studies.

Example 2 - "Go to Station 1." Figure 5 shows a time line for the "Go to Station 1" command. In this case, at t = 1, the
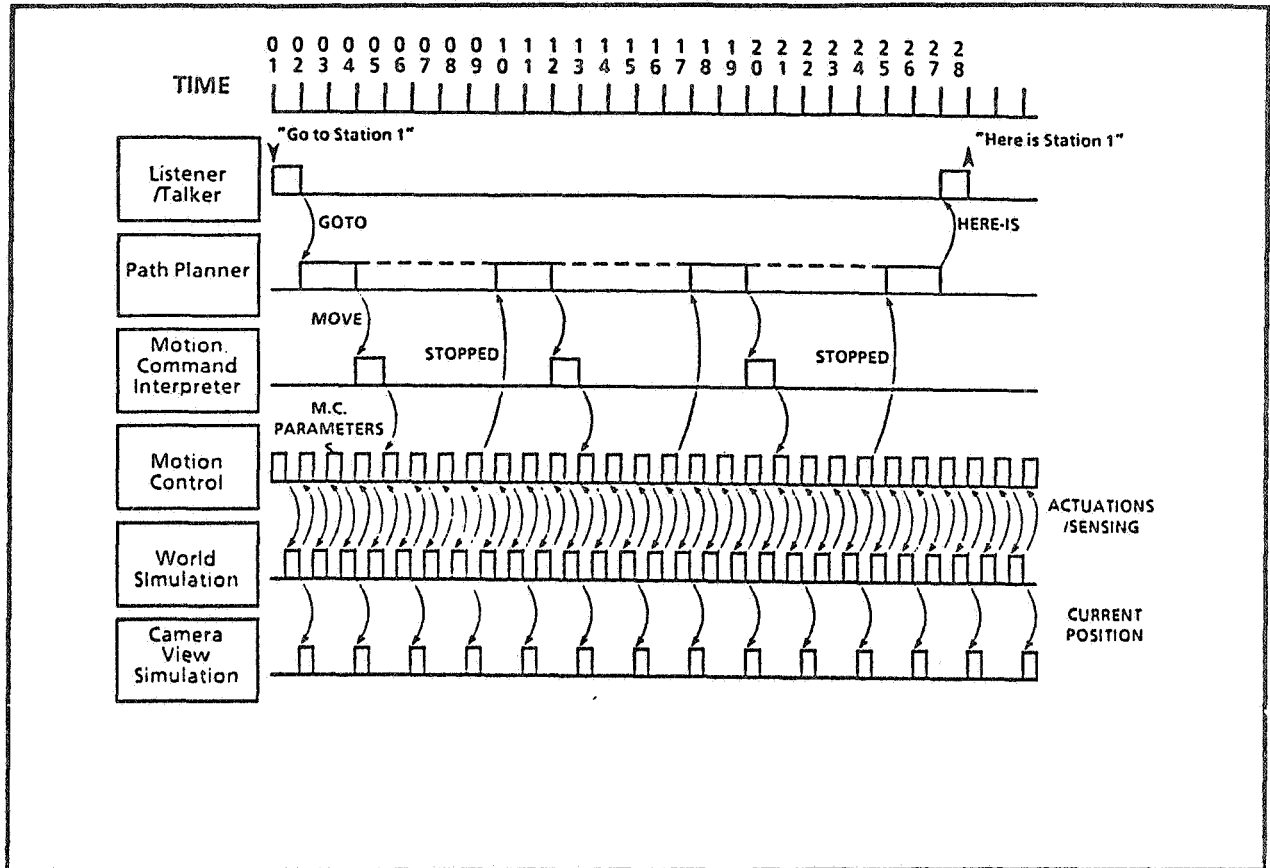
Figure 5 - Time line for "Go to station 1"

command is recognized and passed to the Path Planner instead of directly to the Command Interpreter. The Path Planner must interpret the symbolic name of a location and generate a path to that point. The path is represented as a series of straight line moves between points. At t = 4, the first point in the path has been determined, it is passed to the Command Interpreter to begin the motion. Meanwhile, the Path Planner completes the path generation while, at t = 5, the robot begins its motion. When the first point in the path is arrived at, t = 9, a stopped message is generated and the Path Planner issues the next point in its path, t = 12. This continues until the Path Planner has issued moves to all point in the path. At this point, by observing that it has stopped and that there are no more points in its path, t = 25, it forms the abstraction that it is at the symbolically named point Station 1. At t = 27, it asserts this and the talker turns the message into the English phrase "Here is Station 1" and speaks it.

Conclusions

Mobile robot control systems decompose into a set of distributed processes which can cooperate effectively to perform robot control. A hierarchical real-time closed loop control system can be implemented within a blackboard framework. The blackboard supports the multiple levels of abstraction and the knowledge sources provide for a simple decomposition of the control system into distributed cooperating processes.

The blackboard-based system developed for this project met the following requirements:

1) Support multiple independent processes executing in parallel

2) Support a heterogeneous hardware and software environment

3) Support interprocess synchronization

4) Support anonymous destination message passing

5) Provide I/O support to each process independently

6) Provide a shared data base of state information

7) Support a hierarchical representation of system state information.

We have defined a set of characteristic of knowledge source that are found in real-time control system applications. This will be useful to others analyzing a control problem.

The UNIX-based approach provided the means to simulate parallel knowledge source operation and interaction. The stream-based I/O approach simplifies the interaction of multiple knowledge sources developed on different machines with different languages and internal data representations by communicating in serial streams of ASCII data.

TCP/IP[10,11] provided the ability to communicate between different machines via a local area network with sockets and remote shell commands. This allowed true as well as simulated parallel knowledge source operation.

The multiple knowledge source approach also proved to be a valuable software engineering approach. The various knowledge sources were developed independently once the shared data required in the blackboard database was defined.

Acknowledgments

335

References

1. Kernighan, B. W. and Ritchie, D. M., *The C Programming Language,* 1978 Prentice Hall.

2. Winston, P. H. and Horn, B. K. P., *Lisp 2nd Edition,* 1984 Addison Wesley.

3. Forgy, C. L. OPS5 Users Manual, *Report CMU-CS-81-135,* Computer Science Dept., Carnegie Mellon Univ., Pittsburg, Pa., July 1981.

4. Genesereth, M. R. The MRS Project, *The AI Magazine,* p. 89, Fall 1983.

5. Erman L. D.,Fennel R. D., Lesser V. L. and Reddy D. R., "System Organizations for Speech Understanding: Implications of Network and Multiprocessor Computer Architectures for AI," *IEEE Transactions of Computers,* vol. C-25, no. 4, pp. 414-421, April, 1976.

6. Nii, P. H., Feigenbaum, E. A., Anton, J. J., and Rockmore, A. J., "Signal-to Symbol Transformation: HASP/SIAP Case Study," *The AI Magazine,* pp 23-36, Spring, 1982.

7. Aiello N., *A comparative Study of Relative Strategies for Expert System: AGE Implementation of Three Variation of PUFF,* Heuristic Programming Project, June 1983.

8. Fennel, R. D. and Lesser, V. R., "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II," *IEEE Transactions on Computers,* vol. C-26, no. 2, pp. 98-111, February 1977.

9. Hammond, R. A. and Skillman, T. L., "Simulation of a Free-Flying Inspection Robot," *Proceedings of ROBEXS 1986,* June 1986.

10. Postel, J., "DOD Standard Transmission Control Protocol," IEN 129, RFC 761, USC/Information Sciences Institute, NTIS document number ADA082609, January 1980.

11. Postel, J., "DOD Standard Internet Protocol," IEN 128, RFC 760, USC/Information Sciences Institute, NTIS document number ADA079730, January 1980.

Concepts for Robot Motion Primitives Required for Space Station Teleoperations

Jeffrey L. Grover
Georgia Institute of Technology
Georgia Tech Research Institute
Atlanta, Georgia 30332

Steven A. E. Suchting
Boeing Aerospace Company
Space Station Program
Huntsville, Alabama 38507

ABSTRACT

Ground-controlled teleoperations are expected to be used to augment Space Station manned extra-vehicular activities (EVA) and intra-vehicular activities (IVA). However, ground-controlled teleoperations will encounter communications time delays of from three to eight seconds. Time delays greater than one second have been shown to be detrimental to safe and efficient teleoperations. Therefore, concepts must be developed to overcome the hazards and limitations of time delays when performing teleoperations using robots (telerobotics). The concepts for robot motion primitives incorporate force/torque and tactile sensor feedback to implement the degree of autonomy required for interactive, ground-controlled telerobotics.

Several primitives are studied that augment human initiated actions by providing rapid response (shorter than the time delay) interaction with the physical environment of a telerobot. One primitive provides a shielding mechanism which prevents damage to a robot tool and its payload. This "force shield" stops the motion of a robot if forces and torques measured by a wrist sensor exceed expected threshold values. Threshold values might be exceeded if, for example, the tool or payload comes in contact with an unexpected surface. Another primitive, movement to a constraint, involves shielded movement to the vicinity of a target point and then additional movements until the expected force/torque values are reached. A compliant motion primitive demonstrates the use of force/torque feedback to autonomously complete a task, such as the insertion of a tool into a hole. If a force threshold is exceeded using compliant motion, but it is not the success constraint threshold, attempts are made to comply by altering the robot's path based on the direction of the experienced forces. A primitive for slip detection and recovery is also examined. The orientation of an object being grasped by a robot gripper may be evaluated by using a tactile array sensor. Slippage may be detected by noting a change in the orientation or by a shift in the object's center of mass. Recovery may be accomplished by increasing the grip force and then taking other appropriate action.

These concepts for robot motion primitives constitute a level of intelligent sensing and reaction required to augment human actions through autonomous interaction with the physical environment.

1.0 Overview

The productivity of Space Station laboratory activities will be enhanced by providing continuous experiment operation. To enable this, some experiments are expected to be performed in a telescience mode. In a telescience mode, an experimenter on the ground can monitor an experiment's progress and change its operating parameters via communications between the ground and the space station. A telerobotic manipulator system will facilitate this process by enabling the physical manipulation of experiment samples and facilities from a ground-based control center.

1.1 Development Goal

Ground-controlled teleoperations are being evaluated for use in order to augment Space Station manned activities. However, ground-controlled teleoperations will encounter communications time delays of from three to eight seconds. At best, such delays hamper the operators ability to expediently perform the required duties. At worst, such delays may prevent the operators ability to safely conduct the required maneuvers. This time delay may be the cause of damage to the experiment, experimental setup, or the teleoperation manipulator. Concepts must be developed to overcome these hazards and limitations.

1.2 Current Approach

The present design strives for operator direction of semi-autonomous operation. The operator informs the manipulator of a desired action via the control console. The robot attempts to perform the desired function while utilizing the forces encountered. These forces are analyzed to determine their significance for the attempted operation. Force interpretation discriminates between unexpected forces which suggest the need for corrective action and expected forces which are utilized for comformal motions.

1.3 Attained Capabilities

The present implementation contains a database of all known objects and positions in the work space. This information is used to reduce the need for explicit directives. In this way the operator interface has been implemented in a very efficient manner.

Several low level sensing and motion operations have been developed and combined via a hierarchical control system to implement a flexible manipulator control system. Error conditions are sensed and handled at as low a level as is practical. Conditions which are not efficiently identified or handled by the system are presented to the operator for resolution.

1.4 Future Directions

A baseline experimentation system has been constructed from several low level sensing and control operators. Enhancements to the system will include the introduction of additional operators at the same level or more sophisticated complex operators at the higher levels. The higher level operators are comprised of selected low level operators. The identification and implementation of such operators at all levels will be from experience gained as a result of continued operation of the baseline system. Two high level enhancements currently planned for are active vision and a strategic planning sub-system.

2.0 System Structure

A ground-based telerobotic system has been developed to evaluate design concepts for the laboratory manipulator. The system has been developed to provide generic laboratory technician functions independent of specific experiment requirements. The developed system consists of two primary sub-systems; the control sub-system, and the robotic sub-system. The remainder of this section describes the equipment utilized and the software techniques employed.

2.1 Equipment Used

A system has been constructed to facilitate experimentation in robotic manipulation. Commercially available items have been utilized to the fullest extent possible. The major items are briefly described in the following sections.

2.1.1 Hewett Packard Model 9836 Computer System

The HP 9836 is a 68000 based general purpose computer system. The Pascal programming environment has been utilized to develop the system presented herein. The hardware configuration consists of the system unit, a fixed disk mass storage device, and an RS-232 serial communications controller card.

2.1.2 Unimate PUMA 560 Robot Arm and Controller

The Unimate PUMA 560 Robot Arm is an industrial manipulator which has been used as the basis for all motion activities. The arm provides movement with six degrees of freedom. Motions may be specified relative to a fixed world coordinate system or relative to the current position and orientation of the end effector.

The Unimate controller is an LSI-11 based processing system. The primary functions of the controller are that of development system and manipulator control. A multiprogramming environment supports manipulator control activities by providing a context switch mechanism between the robot control and process control domains. The standard programming language offered in the Unimate control unit is VAL II. Special purpose interrupt driven device handlers have been coded in assembly language and installed in the system to facilitate sensor and command port communications.

2.1.3 LORD Force Torque Sensor

A LORD Corporation Model F/T 15/50 force/torque sensor provides the primary sensing capabilities on the robotic manipulator. The LORD F/T 15/50 has been affixed to the manipulator at the flange of joint six. The end effector normally attached at that point is connected to the opposite end of the sensor.

The LORD F/T 15/50 sensor provides force readings along three axes in the range of 0.2 ounces to 15 pounds. Torque readings are provided about three axes in the range of 0.2 ounce-inches to 50 pound-inches. In addition to these sensing capabilities, the model F/T 15/50 offers preprocessing capabilities. Force and torque thresholds can be specified and monitored by logic internal to the sensor control unit.

2.1.4 LORD Tactile Sensor

A LORD Corporation Model LTS-200 Tactile Sensing Array provides an ability to "feel" objects grasped by the robotic end effector. The LORD LTS-200 has been affixed to the inner surface of one of the pneumatic gripper fingers.

The LORD LTS-200 sensor provides an imaging array of 10 X 16 sensing sites with a 0.7mm inter-site separation. Each site provides an independent linear displacement reading with a resolution of 16 levels. This capability provides a means of determining the shape, orientation, cross section, and texture of objects within the manipulator grasp.

The LORD LTS-200 also provides a force vector sensing capability at the surface of the sensing array. The sensing and processing capabilities of this feature are similar to that described for the LORD F/T 15/50 Force Torque Sensor. This capability provides a means of detecting slip forces which may be experienced between the manipulator surface and an item within the end effector.

2.2.0 Interconnection Scheme

The items described in section 2.1 have been integrated into a system to support the experimentation with robotic concepts. The following sections discuss those implementation components most crucial for understanding the systems operation.

2.2.1 HP Directed Control

The HP 9836 computer is used as the primary operator interface. A robotics oriented control language has been developed to assist the process of describing and directing robot activities. There are three methods by which the operator may direct the robot operations; via interactive command line processing, by evoking a predefined sequence of command line directives, or with the aid of an interactive teach pendant mechanism.

340

2.2.2 VAL Interfacing

The Unimation controller hosts the robot sub-system software. The sub-system software is primarily coded in the VAL language. There are a few routines coded in machine code (assembly language).

2.3 Software Control Structure

The system software has been developed as cooperating systems hosted on an executive processor (HP9836) and a robotic sub-system processor (VAL). Each of these sub-systems has been implemented as a self sufficient system requiring the services of one another. The interaction between sub-systems is via a standard RS-232 link incorporating a sychronous message protocol with CRC error detection. The individual sub-system structures are identical in principle. Each consists of a local executive, command sequencer, data management facility, and a mechanism for control of a peripheral sub-system.

2.3.1 Parallelisms

The dual processor system provides one level of system parallelism. Each processor may be performing independent or loosely coupled activities concurrently. Within each processor the use of real time clock and asynchronous interrupts provides another level of parallelism.

The VAL environment provides a multi-processing arrangement. There are two software partitions executing code in support of the experimental setup. One partition, the process control, handles all external communications and device handling responsiblities. This includes interaction with the executive processor, tactile sensor, and force/torque sensor. The other partition, robot control, handles all command interpretation and execution. This includes normal motion, use of sensors in advanced motions, and error detection and recovery operations.

The process control and robot control partitions interact with one another in a closely coupled arrangement. Control messages are received under interupt control by the process control partitions. After error detection and protocol handshaking, the command is passed to the robot control partition for execution. The robot control partition assumes control for commmand execution. The robot control partition assumes control for command execution and utilizes the process control partition for sensor interaction. Coordination between these partitions is achieved via two mechanisms. Interactions which are not time critical are signalled via global memory variables. Time critical interactions make use of a software interupt facility referred to as the react mechanism. This facility is utilized extensively in support of the error detection and recovery subsystem.

341

2.3.2 Error Handling

A detection mechanism for error conditions was implemented at all levels of the system hierarchy. In general, an error condition initiates the orderly shut-down of the immediate process and an error return to the calling sequence. The error condition is subsequently detected and interpreted to indicate the need for some corrective action. No attempt has been made to resolve all possible error conditions at any given level of the hierarchy. When an unserviceable error condition has been detected at a given level, the detecting sequence similarly effects an orderly abnormal termination. Resolution of error conditions are attempted at as low a level as practical. The uppermost level of error resolution involves an operator action based on a system error message.

The error detection and resolution techniques vary throughout the system implementation. The variation is based on the error's occurance in the overall hierarchy and the specific processes being controlled. In its simplest form it appears as an if-then-else construct. In its most complex form it may require the manipulation of multiple physical objects.

3.0 Sensor Directed Movements

A primary objective of this project is to integrate robot motion with environmental sensing. This objective resulted from two independent operational requirements. First, a teleoperated robot system must be capable of averting potentially damaging situations. Secondly, it is desirable to control a manipulative process based on interaction of the object with the workspace.

3.1 Basic Techniques

The movement of an object from point A to point B is ultimately achieved via a VAL move command. There are three ways of moving an object: normal, shielded, or constrained. Differing conditions dictate a particular mode of movement. The different modes utilize the sensors in different ways. In some cases the determination of success for a particular operation may also vary.

3.1.1 Shielded Motion

One primitive provides a shielding mechanism which prevents damage to a robot tool and its payload. This "force shield" stops the motion of a robot if forces and torques measured by a wrist sensor exceed expected threshold values. Threshold values might be exceeded if, for example, the tool or payload comes in contact with an unexpected surface.

342

The basic concepts associated with shielded motion rely on the identification of expected forces. The current implementation allows for three possible sources of forces to be experienced. The first possible source of experienced force is gravity. Although the system has been developed for on-orbit operations, the gravity consideration has been included for two reasons. The provision of a gravity force vector component facilitates Earth-based development and testing. This provision also allows for operations in a simulated microgravity enviornment. The second source of experienced force results from overcoming inertia due to movements. A third potential source for experienced forces is due to contact with another object in the workspace. This type of force is desirable in compliant modes of motion but is considered abnormal for movements a presummably unoccupied region.

The first step in performing a shielded motion is to calculate the force vector which is expected during the course of the upcoming motion. This vector should have two components. A gravity vector "downward" and an inertial vector directed opposite to the direction of movement. This infomation is used to construct a force shield for the subject movement. This shield is in reality a description of forces in three dimensions which are tolerable for the subject movement in the specified environment. Any experienced force not accounted for by the specified force shield constitutes a breach of the shielding mechanism. Such a situation will immediately terminate the current movement and initiate a fault isolation and recovery sequence.

3.1.2 Constrained Motion

Another primitive, movement to a constraint, involves shielded movement to the vicinity of a target point and then additional movements until the expected force/torque values are reached. The concept of constrained motion suggests two possible scenarios. First, the goal may be to move to some destination defined in terms of an expected force rather than by a coordinate in the workspace. This situation may arise when securing a latch. In this case a locking mechanism may need to be moved until some specified resistence is experienced. It may not be known at what physical point this locking position will occur. For this application the constrained motion command is used to identify an experienced force that will indicate success in completing the attempted operation.

Another type of constrained motion is required when manipulating an item which cannot tolerate excessive forces. This case may be applicable when securing an object in a clamping fixture. Although a physically "locked" position may be known for a clamping mechanism, that position may exert a damaging force on the object being secured. For this case the constrained motion command would be used to specify an experienced force which is not to be exceeded in attempting to complete an operation.

343

3.1.3 Compliant Motion

A compliant motion primitive demonstrates the use of force/torque feedback to autonomously complete a task, such as the insertion of a tool into a hole. If a force threshold is exceeded using compliant motion, but it is not the success constraint threshold, attempts are made to comply by altering the robot's path based on the direction of the experienced forces.

3.1.4 Alignment Verification

A known alignment of an object in the end effector is useful information. Motions are devised based on the best available model of the arm-tool configuration. Indicators exist in the VAL control system for determination of arm position at any given time. When an object is grasped by the end effector, it is the object's position and orientation that are of interest. Search and insert operations are greatly simplified with the use alignment information derived from the tactile sensor. The object creates a depression in the sensitive surface of the tactile sensing array. The array is scanned to produce an image of the tactile imprint. This image may then be processed to recognize key features for object recognition and alignment and positioning operations.

3.1.5 Slip Detection and Recovery

A primitive for slip detection and recovery is also provided. The orientation of an object being grasped by a robot gripper may be evaluated by using a tactile array sensor. Slippage may be detected by noting a change in orientation or by a shift in the object's center of gravity. Recovery is then accomplished by increasing the grip force and then taking other appropriate action. In a gravity environment an immediate recovery technique involves repositioning the gripper directly below the object being grasped. This technique makes use of any continued slippage to guarantee reacquisition of the object by the gripping mechanism. The zero gravity recovery actions currently being considered include bracing the object against a fixed surface or seating the object in a suitable position nearby.

3.2 Combined Movements

The basic techniques described in section 3.1 are combined to construct more advanced techniques. The lowest level of combined motion are described in the following sections. Still more complex combined motions are defined at higher levels.

3.2.1 Searches

Searches are conducted as a part of the insert process. The search phase consists of placing a portion of the object within the physical space defined by the position of interest. This process generally consists of multiple motions satisfied by the eventual placement of the object tip within the opening known as the position. The success state is reached when the plane of the object tip is ultimately placed beyond the plane of the position entry. A cylindrical search pattern is used for insert operations.

The cylindrical search pattern is initiated while in some arbitrary location A. The search command contains information which specifies a first attempt search point B for the iterative search operations. The robot sub-system reacts by attempting a shielded motion from this point A to point B. If a successful movement is made the search is satisfied and control is returned to the control processor. In the event that the initial search attempt is not successful a region would be searched in hopes of accomplishing a similar motion successfully. The search region covered is defined by an expanding spiral perpendicular to the axis defined by points A and B. The spiral occurs on a pair of parallel planes, one containing point A and the other containing point B. The cylinder results from the alternating points in plane A and plane B as the spirals are traversed. The resulting cylinder is traversed from the axis outward. The rate at which the spiral expands is determined by the precision, or degree of fit between the object and the position. A closely sized pair would specify a fine search pattern with closely spaced search points.

3.2.2 Insertions

The insert process consists of advancing the object toward a point in a compliant manner. This process generally consists of multiple motions satisfied by the eventual placement of the object tip at a point which provides the positive seating of an object in a specific position. The success state is reached when the plane of the object tip is ultimately placed beyond the plane of the seat position. Two insert modes are currently supported. The modes supported are referred to as the cylindrical and conical insert patterns.

The insert command is initiated while in some arbitrary location A. The search command contains information which specifies an initial goal point B. The search consists of incremental steps toward the goal point B, and all movements are directed toward the goal point B. After each movement the position of goal point B is adjusted based on forces experienced at that position. Corrective motions to compensate for experienced forces are effected. The resulting position is the new search starting point. A corresponding transformation is applied to the old point B to create a new goal point B. This sequence continues until the end of the object intersects the plane of the goal point B. The resulting possible search region is defined as a cone with apex at initial point A and base centered about the point B. Only

a single point on each incremental plane between plane A and plane B will be searched. The correction at each plane defines the point in the next plane for exploration. The resulting path which is traversed defines the path of least resistance from the apex to the plane of the base.

The conical insert pattern allows four degrees of freedom for corrective actions. The degrees relative to axis of movement, are: (1) translation in the x, (2) translation in the y, (3) rotation around the x, and (4) rotation around the y. The cylindrical insert pattern allows only two degrees of freedom, (1) translation in the x and (2) translation in the y.

THE USE OF COMPUTER GRAPHIC SIMULATION IN THE DEVELOPMENT OF ROBOTIC SYSTEMS

Ken Fernandez

National Aeronautics and Space Administration
Marshall Space Flight Center
Information and Electronic Systems Laboratory
Huntsville, Alabama 35812

ABSTRACT

This paper describes the use of computer graphic simulation techniques to resolve critical design and operational issues for robotic systems. Use of this technology will result in greatly improved systems and reduced development costs. The major design issues in developing effective robotic systems are discussed and the use of ROBOSIM, a NASA developed simulation tool, to address these issues is presented. Three representative simulation case studies are reviewed: off-line programming of the robotic welding development cell for the Space Shuttle Main Engine (SSME); the integration of a sensor to control the robot used for removing the Thermal Protection System (TPS) from the Solid Rocket Booster (SRB); and the development of a teleoperator/robot mechanism for the Orbital Maneuvering Vehicle (OMV).

KEYWORDS

Robotics, Simulation, Computer-graphics, CAD, CAM, Off-line-programming, Robotic-welding, Robotic-spraying, Satellite-servicing.

INTRODUCTION

Robotic systems have become increasingly important to all facets of manufacturing: space is no exception. Perhaps the most publicized space robot is the Remote Manipulator System (RMS) which was built by Canada for the U.S. Space Shuttle. Prior to the RMS, robot manipulators were used on unmanned spacecraft to investigate soil properties on the moon and on Mars. Plans for the U.S. Space Station which will become operational in the early 1990's include the use of teleoperators and robots to perform routine station tasks e.g., inspection and maintenance. Earth-bound robots have also been used extensively to support the manufacturing of spacecraft components (Fernandez 1983,1985). Although the applications for space and earth seem radically different there remain many common issues in the procedures for design and testing of robot systems. Graphic simulation has proven to be extremely effective in the design of both types of system. In this paper we will examine: design issues for robots; ROBOSIM, a NASA developed computer graphic simulation tool; and three robotic systems that were developed using computer graphic simulation techniques.

Kinematic Design Issues

In designing a robot cell the selection of the robot's kinematic design is usually considered first. The number of robot joints, the type of joint (revolute or prismatic), and the physical configuration of each jointed segment are all elements of the robot's kinematic design. The position of the last reference frame (hand frame) is determined by the joint positions and the geometric relationships (kinematics). Minor changes in the kinematic design of a manipulator can greatly affect the volume through which the robot's hand may be moved. The design of the end-effector (tool) and the orientation of the part (workpiece) with respect to the robot (part positioning) also greatly affect the ability of a robot to perform a given task. For applications which will use an existing robot the designer must choose the appropriate robot, design the workcell layout and part fixturing. For systems which will use a custom-built robot, the task of designing the robot is added. A mistake in the design of a cell without the use of computer graphic simulation may not be detected until the hardware integration phase. This can result in costly schedule delays, procurement of incorrect components, and a greatly increased system cost.

Robot Motion Control

Robot control development is another area which can benefit from the use of computer graphic simulation techniques. Robot control algorithms may be viewed as existing at two levels: the kinematic control level; and the path planning level. Kinematic control algorithms are a function of the arm's kinematic design. These algorithms relate the position of the end-effector's reference frame to the joint position commands required to achieve the commanded position. These algorithms are a software implementation of the inverse kinematic equations. Prior to the use of graphic simulation, the control programs were debugged by observing the robot's motion subject to the commands of the experimental computer program. For robot systems with relatively low lifting capacity, a faulty program resulted in little more than embarrassment for the developer, however robot capacities have increased to the point where payloads are in the hundreds or thousands of pounds. Mistakes in programming can be serious. Another difficulty encountered in using the actual mechanism in the debugging process occurs for robots designed for use in zero-G which may not operate in a one-G environment. Again graphic simulation is the indicated procedure for this type of development.

Robot Path-Planning/Verification

Robot path-planning is the process of developing the sequential position, orientation and velocity commands that the robot's end-effector must execute in order to perform the desired function. Most current industrial robots are programmed using a teach pendant to manually command the robot to the desired points, this is the on-line manual programming method. Manual programming is highly in-efficient since the robot must be taken out of service, the path generated manually, replayed for verification and ultimately executed. On robots whose path programming is changed infrequently this is not significant, but for systems in which programming must be flexible manual programming is not satisfactory. Just as numerically controlled (NC) machine tools have become entirely programmed by off-line algorithms, the programming of robots will also eventually all be automated. Graphic simulation is a vital step that must be performed prior to the execution of an off-line generated robotic path program. Simulation will verify that: (1) the path specified is correct for the task; (2) the inverse kinematic equation may be solved at all points along the path program (controllability); and (3) the arm or other components will not collide accidentally with obstacles within the workcell.

Robot Dynamics

In industrial applications the primary dynamics issues are that the robot chosen for a task is capable of handling the required payload weights and transport velocities. Industrial robots are typically rated for lifting capacity only. An approximation of the robot's ability to perform a task dynamically can be made through dynamic simulation of the loaded robot. The maximum joint loads recorded during the dynamic simulation are compared to the loads that result if the manipulator were statically loaded per the manufacturer's specifications. If these joint loads are exceeded by the dynamic tests, then the robot may not be capable of performing the task. Since this is only an approximation, a safety margin should be used in making the final decision.

Although dynamic simulation is important for industrial robot systems, it is mandatory for systems used in space. Manipulator mechanisms and joint actuators are limited in weight due to launch considerations. Power supply limits reduce the size and rating of the mechanism's actuators. Dynamic studies will help to insure that the planned robotic tasks do not exceed the limits of the mechanism. The zero-G environment may be an advantage for handling larger payloads than would be possible on earth, but the dynamic interactions of the loaded manipulator and its mounting platform are significant for a space based robotic system. The possibility exists for parasitic oscillations to occur between the manipulator and the spacecraft's attitude control system. Simulation studies may reveal the existence of these or other undesirable effects.

ROBOSIM OVERVIEW

Simulation Procedure

ROBOSIM was developed over a three year period at the Marshall Space Flight Center (MSFC) to facilitate the design and development of robotic systems. Prior to ROBOSIM, robotic simulations were limited to the construction of scale models. Using ROBOSIM the kinematic design of the manipulator mechanism and other workcell components are modelled via a simulation language. The model consists of solid primitive shapes which approximate the robot's shape and mass properties. The joint configuration and type, either revolute, prismatic or fixed, are also specified. Once modelled, ROBOSIM computes the standard linkage parameters (Hartenberg 1955), the inverse kinematics and the manipulator's dynamics. The designer may also specify the joint actuator transfer functions. Path motion is specified by position and velocity language constructs.

348

ROBOSIM Hardware Configuration

ROBOSIM is resident on a Digital Equipment Corporation (DEC) VAX11/780 processor. During simulation development the user may use a low cost terminal with TEK 4014 graphics compatibility. Although a simulation may be executed using a non-real-time terminal, the use of a real-time graphics display is preferred. Interfaces have been provided for several dynamic display systems including Evans & Sutherland PS330, GTI Poly 2000, Silicon Graphics IRIS with other interfaces planned. A limited Initial Graphics Exchange Standard (IGES) pre- and post-processor allows ROBOSIM to communicate graphics and tool motion commands with any CAD/CAM system adhering to the standard which was developed by the U.S. National Bureau of Standards.

The simulator's speed for non-dynamic studies is greater than real-time. This speed is decreased for very large models with multiple robots or robots with many degrees-of-freedom. Studies that required the modelling of dynamic effects also load the simulation processor. An Applied Dynamics AD10 parallel processor is used to improve the simulator's response in these situations.

ROBOSIM Software System Structure

ROBOSIM's software structure may be characterized as a hierarchy of three levels of software utilities. This structure is typical of large software systems. At the core or kernel of this system are routines that provide support for the most rudimentary of simulation tasks. Included among these functions are vector and matrix arithmetic and display control. The typical user of ROBOSIM interacts with these routines indirectly through his use of higher level utilities. A characteristic of routines at this level is their inflexibility in their interfacing requirements i.e., data must be provided in specific formats. By interfacing via the higher levels a user avoids these requirements, however direct access is available when needed. Typically, a ROBOSIM user who is performing simulation studies involving externally supplied mechanism control algorithms must communicate directly with the kernel routines.

The second level within ROBOSIM integrates the lower level routines into more complex algorithms that perform often needed tasks in display management and robot control. Examples of graphics routines that function at this level include subroutines to perform viewpoint and perspective transformations. Examples of routines that service robot kinematics and control issues include those which perform end-effector position computations and formulations of the manipulator's Jacobian matrix.

The highest level within ROBOSIM provides the human interface. At this level robots, workpieces, and fixturing assemblies may be modelled, placed within a workcell, programmed, dynamically simulated and viewed using fewer than forty distinct language instructions. The simplicity of this software interface greatly increases ROBOSIM's use and it is this interface that is perhaps the most important feature of ROBOSIM.

SIMULATION EXAMPLES

ROBOSIM V1.0 became operational in July 1985. In the year since, ROBOSIM has been applied to numerous robot simulation studies, the three listed below are typical. The studies include: the development of an off-line programming algorithms for welding on the SSME; the development of vision sensor guided control for a robot used to refurbish the SRB; and the design of a robot manipulator for the Orbital Maneuvering Vehicle. For each study a discussion of the application, the simulation goals, and the results will be presented.

Downhand Control for SSME Robotic Welding

The Space Shuttle Main Engine is constructed of stainless steel using over 2000 welded seams. At the present time 30% of these welds are performed by fixed automation while the remaining 70% are performed manually. A study performed of the manufacturing operation indicates that an additional 30% could be automated in a cost-effective manner using robotic welding techniques. The primary goal of this effort is the improvement of weld quality and reliability. A further improvement in manufacturing efficiency could be obtained by using automatic off-line robot programming techniques with downhand welding control. Downhand welding is the term applied to arc welding with the part in an orientation that maintains the weld puddle in a horizontal plane. This allows increased puddle size with a resulting greater deposition rate, fewer passes and reduced welding times.

Manual robot programming to perform downhand welding is extremely tedious and the results are only approximate. The algorithm for automatic off-line programming of the downhand position (Fernandez 1986) was developed using ROBOSIM as a test bed. The algorithm programs the robot and part positioner so that their coordinated motion results in a constant weld travel speed while maintaining the downhand position. Figure 1 depicts the robot cell with the six degree-of-freedom robot and a two degree-of-freedom part positioner. The part in figure 1 is a corrugated metal sheet. The part geometry may be read from a CAD data base using IGES format, or it may have been inferred from a manually generated path program sent to the downhand algorithm via the robot's communication interface. In either case the algorithm computes the

349
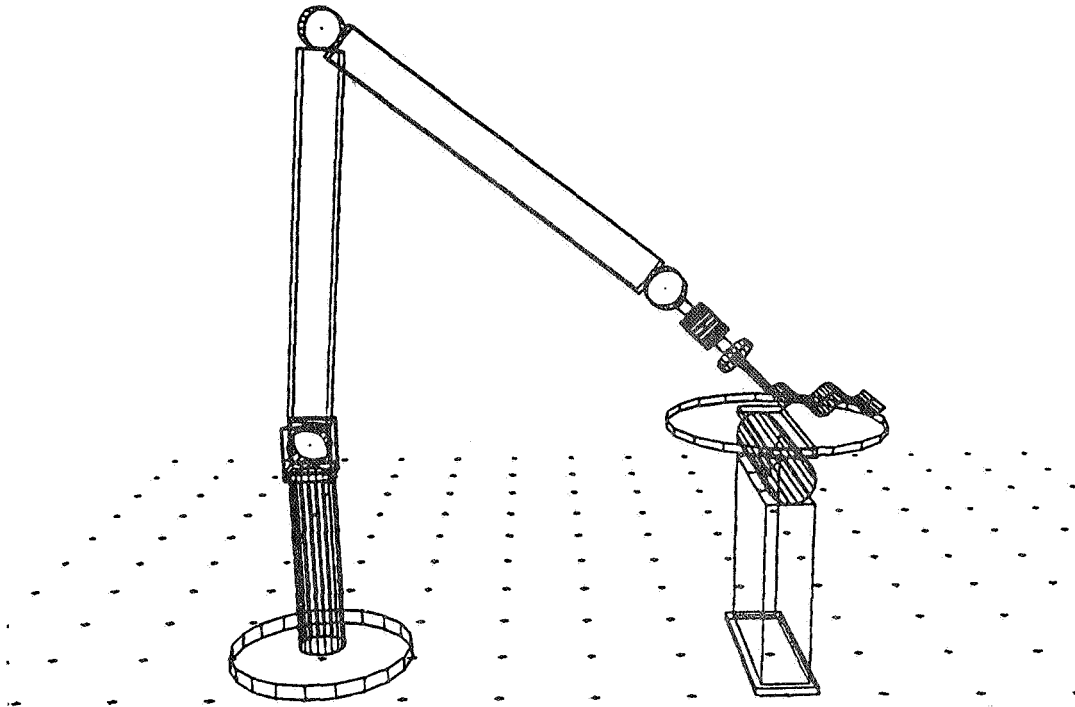
Fig. 1. Simulated Six Axis Robot and Two Axis Positioner.

desired local vertical in a reference frame moving along the weld seam at the specified weld velocity. Weld positioner commands are computed so that the desired downhand orientation is achieved. Robot position and velocity commands are also generated to keep the torch moving in the weld seam at a constant surface feedrate. Figure 2 depicts several frames from the simulation of the downhand welding algorithm. In figure 2 we note that the algorithm is functioning since the tangent to the weld seam remains horizontal at the point where the torch is in contact.

Vision Guided Off-Line Programming for SRB Refurbishment

The Solid Rocket Boosters used to assist in launching the Space Shuttle are designed to be re-used. To achieve this the Thermal Protection System (TPS) prevents the erosion of the booster's casing during the heat of re-entry. The main component of the TPS is the Marshall Sprayable Ablator (MSA) which reduces the booster's skin temperature by controlled evaporation. After recovery at sea the SRB is returned to the booster processing facility at the Kennedy Space Center (KSC).

High-pressure waterblast (20000 psi) is used to remove the partially burned ablative material prior to its re-application. Due to the difficulty in performing the cleaning operations manually, robotic workcells were developed (Fernandez 1983). Prototypes of these workcells were implemented at the MSFC Industrial Productivity Facility in Huntsville, Alabama. A computer graphic simulation of the prototype robotic cell is shown in figure 3. The robot, a Cincinnati Milacron HT3, is equipped with the high pressure nozzle. The aft booster section is shown mounted on a computer controlled rotary positioning table. In the initial implementation of this cell, manual robot programming methods were employed. The current operation includes both manual and off-line programming techniques. One problem in the operation of this cell occurs when the water blast fails to remove the MSA in the first cleaning pass. At this point the robot and turntable must be re-programmed manually to perform the touch-up cleaning.

A solution to the problem of programming the robot to perform touch-up cleaning of the TPS residue is the development of a vision sensor and off-line programming utilities. Graphic simulation via ROBOSIM was used to develop these programming utilities without the danger of damaging the actual workcell during initial development and de-bugging procedures. In operation the vision sensor will scan sections of the SRB that are presented by rotating the turntable. Due to the spray and debris real-time visual inspection is not possible, instead the inspection is performed after the entire cleaning pass is completed. The vision algorithm within the sensor provides information on the location of the MSA residues as x and y-coordinate locations referenced to the image plane of the sensor camera. Although the vision routines were developed under a separate effort, the camera is simulated in the graphic system by placing an "eye-point" in the same location and orientation as the hardware system. The focal length of

350

Frame A   Frame B

Frame C   Frame D
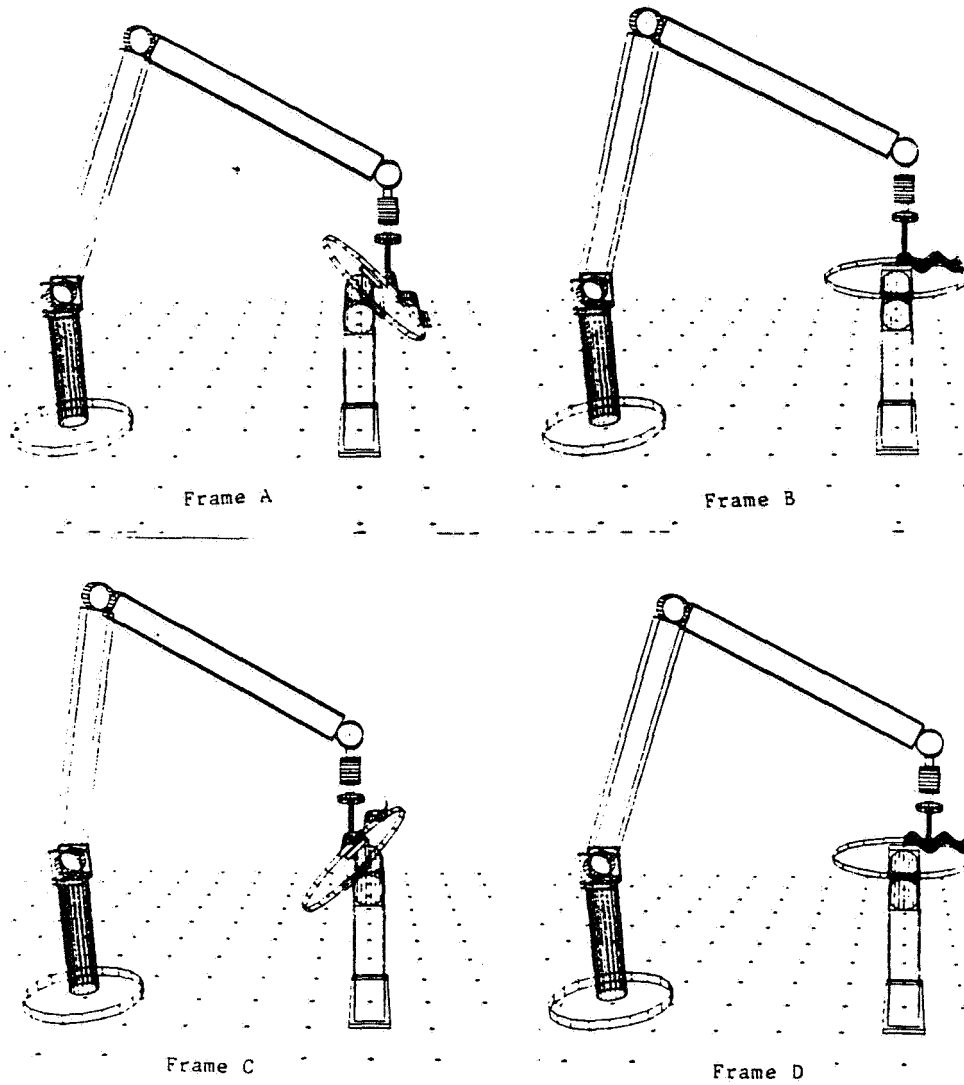
Fig. 2. Simulated Weld Operation with Automatic Downhand Position Control.
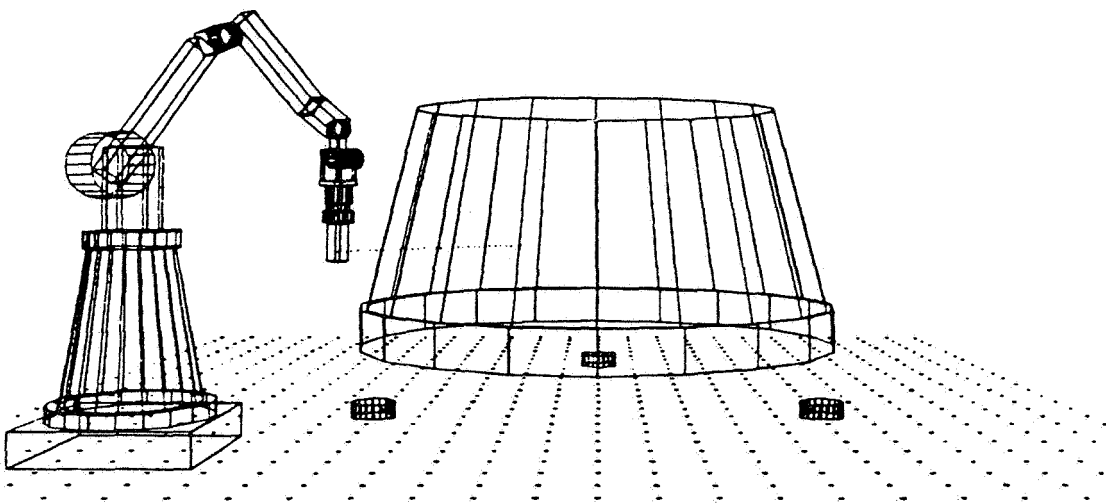
Fig. 3. Simulated SRB Refurbishment Workcell.

351
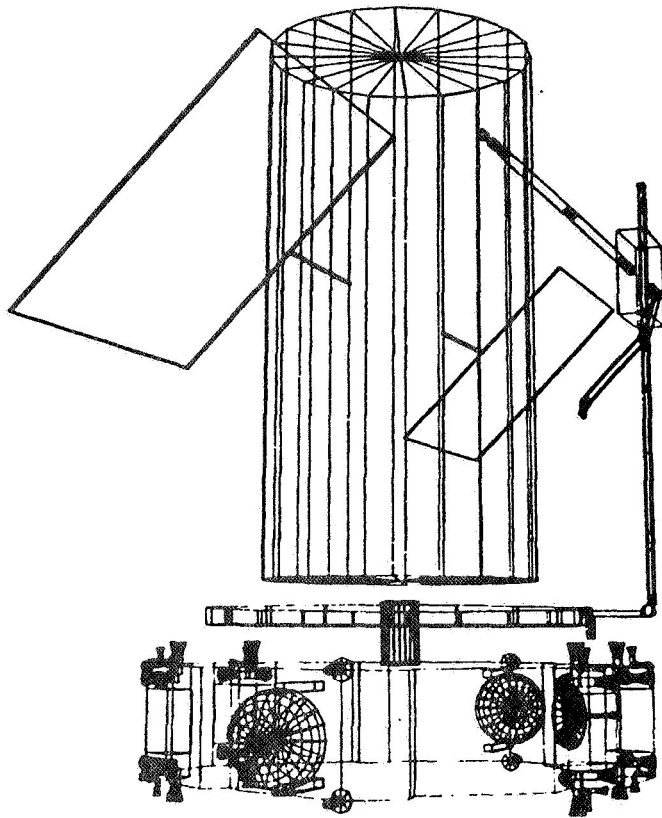
Fig. 4. Simulated On-Orbit Satellite Servicing Mission.

the perspective transformation (Duda 1973) associated with the "eye-point" is adjusted to match the field-of-view of the sensor camera's lens. In evaluating the off-line programming algorithm a simulated MSA residue is placed on the modelled SRB. The residue is placed within the field-of-view of the "eye-point" by rotating the turntable in the graphics model. To simulate the sensor's output the screen x and y-coordinates are noted and passed as input to the off-line programming utilities in a manner similar to the actual sensor. The resulting turntable and robot motion commands were executed by the graphic model, and the resulting operation was viewed in graphics to determine if proper cleaning motion would have occurred. This result is established when the graphic representation of the spray (dotted line in figure 3) impinges on the simulated residue. The use of graphic simulation will continue when the sensor is integrated into the cleaning workcell. During operations the simulation will serve as a preview verification of the off-line generated cleaning paths.

Design of A Robot for the Orbital Maneuvering Vehicle

The Orbital Maneuvering Vehicle is designed as a re-useable, remotely controlled, free-flying vehicle capable of performing a wide range of on-orbit services in support of orbiting assets. It is projected as an important element of the Space Transportation System (STS), designed to operate from either the Shuttle , the Space Station or from the ground. The descriptions of the OMV or manipulator mechanism contained in this paper are not specific to any designs which may be currently under consideration by the U.S. National Aeronautics and Space Administration, however, the functional concepts described are correct and have been published elsewhere (Huber 1984).

The concept of the OMV includes the ability to accept mission kits to allow it to perform a variety of tasks in addition to its role as recoverable booster. One such kit is a manipulator/teleoperator, the "Smart Front-End" (SFE), which will allow remotely controlled manipulation to accomplish satellite and Space Station service tasks on-orbit. Figure 4 illustrates this concept. The OMV is shown equipped with a generic SFE manipulator. The SFE pictured consists of a bi-lateral pair of six degree-of-freedom (DOF) manipulators and a manipulator transport mechanism. The transport system provides three DOF: a rotary track which encircles the docking adapter; a hinged boom; and a sliding joint allowing the bi-lateral pair to traverse the boom. The generic satellite which is being serviced in figure 4 is shown detached from the OMV/SFE cluster for clarity. In normal operation a solid connection would be established by a docking mechanism.

352
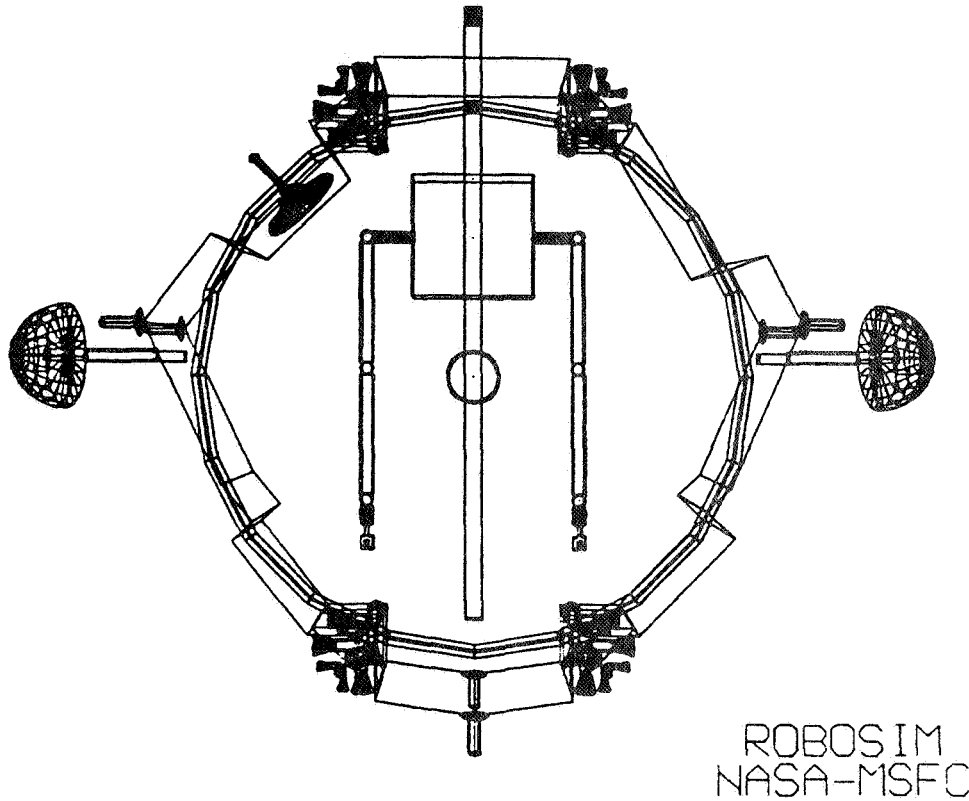
Fig. 5. OMV Shown with SFE Manipulator in Stowed Position.

ROBOSIM will be used extensively to assist in the development and evaluation of concepts for the SFE manipulator. Kinematic studies will reveal whether the SFE mechanism can be folded and stored within the space allocated on-board the Space Shuttle. Other kinematic studies will be required to determine if the OMV/SFE cluster can be successfully deployed from the cargo bay by the Space Shuttle's RMS. In figure 5 our generic OMV/SFE cluster is shown with the SFE folded in the stowable configuration. Further kinematic studies will determine if collisions between the SFE manipulator and satellite appendages occur during the execution of planned motion paths.

The implementation of an SFE manipulator will also require the development of several modes of mechanism control. An algorithm to control the SFE during deployment or un-folding will be developed. Although this type of algorithm usually involves a predetermined sequence of joint motions, provision must be included to override this sequence, if necessary, and execute new motions to correct or avoid anomalies. During docking operations the mechanism can take a passive or an active role. If a passive role is assumed, control algorithms for the SFE can improve the maneuverability of the OMV by arranging the arm's configuration to minimize inertial imbalance, avoid obstruction of the target satellite and prevent the reaction control system (RCS) thruster plumes from impinging on the SFE. Strategies of controlled compliance in the SFE joint servo control loops may further improve the controllability of the OMV during fine docking maneuvers by de-coupling the SFE's mass or actively using the SFE's momentum to affect additional control.

Once the OMV is docked with the target satellite a variety of different control issues must be resolved. As previously mentioned, algorithms that use mechanisms with kinematic redundancy to avoid collisions and minimize disturbance torques could significantly improve the system's performance. Real-time computer graphic simulation coupled to prototype teleoperator workstations can aid in resolving many issues relating to man-in-the-loop control. The placement of cameras may be simulated to insure that the field-of-view (FOV) is not obstructed. If a dual arm SFE design is chosen, graphic simulation could help to determine the most effective human interface for controlling the bi-lateral mechanism. Graphic simulation will not end with the successful SFE design, during servicing activities, a graphic display will allow the human operator to preview service tasks in simulation. Since communication delays in the man-in-the-loop control system may be large and varying, the use of a "predictive graphic display" to supplement the delayed visual feedback may improve the efficiency in performing operations remotely. When semi-autonomous or "supervisor control" methods are developed, the

353

graphics display would allow the human to verify mechanism motions that are proposed by the controller. One final note relates to the design of the satellite rather than the OMV itself. Current satellite design philosophy is oriented toward multiple redundancy and no post-launch servicing, the advent of on-orbit service techniques will relax some of these design constraints, but satellite design must change to take advantage of these new possibilities. Hardware simulations of servicing missions on modular satellites have been performed (Fernandez 1980a,1980b,1984 and Scott 1985a,1985b), but computer graphic simulation provides a cost-effective means of preliminary evaluation of the compatibility between a satellite and the servicer.

CONCLUSIONS .

The experience gained at the Marshall Space Flight Center indicates that the use of computer graphic simulation in support of robot systems development is extremely important. Although hardware implementation is not replaced by these simulators, a considerable cost savings is experienced by delaying hardware implementation until the designs have matured. Once a robot system becomes operational the value of graphic simulation continues as a means of previewing planned task execution. It is expected that as the performance of computer graphic simulators increases and as hardware costs decrease the use of graphic methods will become widespread.

REFERENCES

Duda, R. O. and Hart, P. E. (1973). Pattern Classification and Scene Analysis. John Wiley & Sons, New York. Ch. 10, pp. 379-404.

Fernandez, K. R. (1980a). Computer Control of a Robotic Satellite Servicer. Proc. of the IEEE Southeastcon '80. Nashville, TN. pp.237-240.

Fernandez, K. R. (1980b). Application of a Computer Controlled Robot to Remote Equipment Maintenance. Proc. of the IEEE IASCON80. Cincinnati, OH. pp.1180-1184.

Fernandez, K. R., Jones, C. S. III, and Roberts, M. L. (1983). NASA's Use of Robotics in Building the Space Shuttle. Proc. of 13th ISIR/ROBOTS 7 of the SME. Vol. 1. Chicago, IL. pp.11-35 to 11-43.

Fernandez, K. R., Purinton, S. C., and Bryan, T. (1984). Simulation of a Robot System Used to Remotely Service Satellites. Proc. of the ANS Robotics and Remote Handling in Hostile Environments Nat. Topical Meeting. Gatlingburg, TN. pp.317-321.

Fernandez, K. R., et al (1985). In Robert L. Vaughn (Ed.) Space Shuttle: A Triumph in Manufacturing. SME Dearborn, Michigan. Chapter 5, pp.229-248.

Fernandez, K. R. and Cook, G. E. (1986). Computer Graphic Simulation of An Algorithm for Controlling Downhand Position in Robotic Welding. Proc. of the SME Conf. on Robotic Solutions in Aerospace Manufacturing. Orlando, FL. 10 pages.

Hartenberg, R. S. and Denavit, J. (1955). A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices. ASME Jour. of Applied Mechanics June, pp.215-221.

Huber, W. G. (1984). User's Guide for Orbital Maneuvering Vehicle. NASA, MSFC. 12 pages.

Scott, D. R. (1985a). Remote Satellite Servicing. Proc. of the NASA Workshop on Proximity Operations in Earth Orbit. Houston, TX. 14 pages.

Scott, D. R. (1985b). Concepts in Remote Satellite Servicing. Proc. of the NASA Workshop on Satellite Servicing. Houston, TX. 16 pages.

ORIGINAL PAGE IS OF POOR QUALITY

GENERIC SUPERVISOR: A KNOWLEDGE-BASED TOOL FOR CONTROL OF SPACE STATION ON-BOARD SYSTEMS

J. R. Carnes
BOEING COMPUTER SERVICES
P. O. Box 1470, JA-65
HUNTSVILLE, AL 35807-3701

R. Nelson
BOEING AEROSPACE COMPANY
P. O. Box 1470, JA-68
HUNTSVILLE, AL 35807-3701

ABSTRACT

The concept of a generic module for management of on-board systems grew out of the structured analysis effort for the space station software. Hierarchical specification of subsystems software revealed that nontrivial "supervisory" elements are required at all levels. The number of supervisors (and subsequent software) required to implement the hierarchical control over on-board functions comprise a large portion of the space station software. Thus, a generic knowledge-based supervisory module significantly reduces the amount of software to be developed. This module, the Generic Supervisor, depends on its knowledge of control to provide direction for subordinates and feed-back to superiors within a specific subsystem area.

The Generic Supervisor provides an adaptable and maintainable control system. A portion of the Space Station Environmental Control and Life Support System (ECLSS) has been implemented as a hierarchy of supervisors. This prototype implementation demonstrates the feasibility of a generic knowledge-based supervisor, and its facility to meet complex mission requirements.

1.0 INTRODUCTION

1.1 PROBLEM

Many system control tasks on space station are complex, and will require substantial human attention if performed through conventional technologies. Examples include:

(1) Complex on-going background tasks using the same resources must be monitored and controlled, and coordinate with a variable activity schedule,

(2) Individual activity schedules change rapidly to adapt to changing circumstances and requirements, and impact from other schedules,

(3) Complex subsystems must act together to efficiently perform delicate and difficult functions with complete safety, and

(4) Prompt fault detection and circumvention (essential to continuing system functions).

The aggregate workload of such tasks could be a constant drain on crew availability and productivity, especially on-board the space station. It appears increasingly likely that the crew may be unable to cope with the probable workload unless assisted by advanced machine intelligence.

The development of complex systems and subsystems will require substantial human resources (throughout the software development lifecycle) performed through conventional technologies. Hierarchical specification of subsystems software reveals that non-trivial supervisory elements are required at all levels. The number of "supervisors" (and subsequent software) required to implement the hierarchical control over on-board functions comprise a large

portion of the overall space station software.

1.2 SOLUTION

The structured analysis efforts have shown the feasibility of a knowledge-based supervisory module for the management of the on-board systems. This common representational environment can significantly reduce the amount of software development through the repeated use of a generic control element.

This approach which will enhance the ability of the crew to efficiently run the space station. The system has both control capabilities as well as the ability to dynamically schedule continuing or pending activities in response to crew instructions, subsystem conflicts and changing conditions.

The system has several communicating knowledge-based elements working on portions of the control problem. Together, they comprise an intelligent control system using these knowledge-based systems in a distributed fashion. They can be designed to cooperatively handle and execute station control activities under crew management. The system can accept performance and scheduling knowledge from standard operating procedures, and produce task schedules for various supervisors within the structure. These supervisors provide subsystem control by executing schedules and solving problems within their supervisory domain.

However, this solution does present some difficulties. The large number of system and subsystem controllers that will be required to manage station functions will certainly lead to a corresponding number of knowledge-based systems. Each of these intelligent systems will control a specific portion of the station's function

and will therefore be designed to operate within a specific knowledge domain. This can produce a wide variety of fundamentally different knowledge-based systems. However, if enough of fundamental properties can be abstracted into the Generic Supervisor, this generic system/subsystem representation will provide a common knowledge-based environment.

The representation of supervisory/control knowledge is the crucial issue for the generic supervisor. The knowledge of subordinate and superordinate elements for a diverse set of applications (command validation, resource planning, and fault management) is represented so that a single software module can control arbitrary systems. The techniques used to manage the interaction of these hierarchical knowledge bases play a significant role in the Generic Supervisor. Functional relationships that synthesize module operations from command, sensor and subordinate inputs are represented by symbolic expressions embedded in frames. These expressions contain representational formalisms and control constructs for encoding the wide variety of problem solving domains necessary in this environment.

2.0 THE SUPERVISOR STRUCTURE

Control knowledge in the Generic Supervisor is represented in both internal (within the supervisor module) and external (the skeletal structure) forms. Internal knowledge represents the basic supervisory engine (and is the topic of this section). External knowledge describes the control requirements of an application and any relationships embedded in the overall problem decomposition. These relationships are derived indirectly from the control structure and passed to the supervisory engines they impinge upon.

2.1 SUPERVISOR INTERNALS

The Generic Supervisor is a frame of twelve slots interacting with themselves and a thirteenth slot representing the external knowledge base. (The knowledge base is discussed in Section 3.) The supervisor concept is achieved through a well-connected causal network of demons, each of which represents a vital supervisory task. These tasks include command scheduling and execution, command and schedule analysis, command decomposition and reintegration, fault detection and correction, performance evaluation and reporting. Within any instance of a supervisor frame, the general "control knowledge" used to coordinate the different nodes in the network is represented locally, and the more specific "control knowledge" required to drive a particular application is represented in the knowledge base.

The network of functional nodes, or demons, actually resembles a custom blackboard architecture designed for solving control problems. The structure of this network remains an invariant engine for every distributed supervisor in an application. A certain class of inputs (commands) move through the network and are analyzed, scheduled, decomposed, and executed at various stops. Other classes of input (external states and command results) are analyzed, reintegrated, evaluated, and reported. All nodes in the internal network monitor their inputs for fault tolerance and performance.

The demon structure can be reduced to three nodes (see Figure 1): communication, command, and core. The communication node represents all interaction with the world outside of the supervisor. This node handles communication between a group of external sources and sinks (other supervisors, the crew interface, and sensors) and the supervisor inner command layer. It receives raw command and sensor inputs, and generates an initial analysis of command and feedback messages for consumption by the command node. In return, the command layer generates command decomposition and performance information to be distributed by the communication node. The communication node is responsible for fault detection and correction, and performance evaluation and reporting.

Detailed command processing, such as command scheduling and execution, and command decomposition and reintegration, takes place within the command node. The command node produces command schedules and analysis of reintegrated commands which it delivers to the core node for further correlation/verification. It receives optimal schedule analysis and direction on the reintegration of feedback information from the core node, which provides the deepest analysis of command/feedback data and their impact on supervisor schedules.

2.2 COMPOUND STRUCTURES

The flexible design of the Generic Supervisor makes it an ideal building block for compound control structures (the most common of which is the hierarchy, see Figure 2). The supervisor's ability to decompose an incoming message, or command, permits control models to explicitly represent a control application from the top-down point-of-view.

The supervisory control engine is already developed, so building a system of supervisors involves only two steps. First, determine the structure of the overall control system, and the second, provide the behavior rules for each supervisor. The knowledge base structure

is transparent to the system developer, and is created automatically.

3.0 THE KNOWLEDGE STRUCTURE

A supervisor responds to command stimulus by examining its localized knowledge structure (see Figure 3). Each control class has decomposed subclasses and a parallel command class. A command instance describes the procedure, decomposition, and responses of a correlated supervisor stimulus through its relationship with rule classes. Decomposition information is derived from command class/subclass and command instance relationships.

3.1 SUPERVISOR KNOWLEDGE

The knowledge slot in the supervisor frame allows the supervisor control functions to interact freely with the application control knowledge. Knowledge within a supervisor instance is partitioned into two sets. The first set is the supervisor/world knowledge. This knowledge drives the supervisor in its communication with elements outside the supervisor (i.e., superordinate and subordinate supervisors). The other set is the supervisor/subordinate knowledge. This set governs the actions of the demon nodes (within the communication, command, core nodes) for a given supervisor.

Supervisor/world knowledge is represented in a set of relationships between the command classes and command instances. A command class is partitioned into subclasses which represent the granular class decomposition. These command subclasses become command classes at a lower level. Each command class has an enumeration of command instances which represent different functional activities possible at the current command level.

Command instances are functionally partition into subcommand instances which are represented both as instances subclasses and command subclass instances. This complex set of relationships allows the supervisor to derive standard command decomposition information (i.e., parent instance and child instances) in an environment rich with inherited class knowledge.

Supervisor/Subordinate knowledge is represented in the relationships between a command instance and a rule class. Each command instance is linked to a rule class designed to direct the supervisor in the solution of a specific problem class. Implicit inheritance relationships (from command instances to rule classes) are used to gather command rules into the rule classes required for supervisor operation. The stored rules represent a small partition of conditional/procedural information pertaining to a command instance or instances. Rules are entered in a standard

· IF <condition> THEN <assertion> DO <procedure>

form, and are later compiled for use within the supervisor (see Figure 4 for the internal rule form).

3.2 SYSTEM KNOWLEDGE - EXAMPLE

The following example shows a control software implementation using the Generic Supervisor package. The subject is a small portion of the Environmental Control and Life Support Subsystem (ECLSS). (A more elaborate version of ECLSS using 45 supervisory modules has been implemented and is running in the BCS AI Lab.) The control problem, ECLSS, is broken into two subproblems, with one subproblem broken down into two state/control monitors (see Figure 5). The command knowledge for this

C-5

example shows the relationships present for a simple class and instance case. (Square nodes are used to represent classes and round nodes to represent instances. Solid lines depict class-subclass relationships, and dashed lines show the class-instance relationships.)

The flow of control knowledge is shown in the decomposition of command instances, while the links to class knowledge are shown through the command class decomposition. For example, the command instance "MANAGE ATMOSPHERE PRESSURE" inherits general knowledge from the ATMOSPHERE COMPOSITION AND SUPPLY CLASS and specific control commands from MANAGE ATMOSPHERE, and when presented to the supervisor engine, generates specific control commands for its subordinates.

REFERENCES

1. Carnes, J. R., Generic Supervisor Project Plan, BCS Memo 56400-086-031, Huntsville, AL, July 31, 1986.

2. Carnes, J. R., Requirements for a Generic Supervisor, BCS Memo 56400-086-028, Huntsville, AL, July 22, 1986.

3. Ensor, J. R., Gabbe, J. D., Transactional Blackboards, Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Vol. 1, Los Angeles, CA, August, 1985.

4. Environmental Control and Life Support System (ECLSS) Preliminary Architectural Control Document (ACD), Space Station Work Package 01 Level C, Marshall Space Flight Center, Huntsville, AL, May 30, 1986.

5. Fox, M. S., An Organizational View of Distributed Systems, IEEE Transactions on Systems, Man, and Cybernetics, Vol. SMC-11, No. 1, January, 1981.

6. Hayes-Roth, F., Waterman, D. A., Lenat, D. B., Building Expert Systems, Addison-Wesley, 1983.
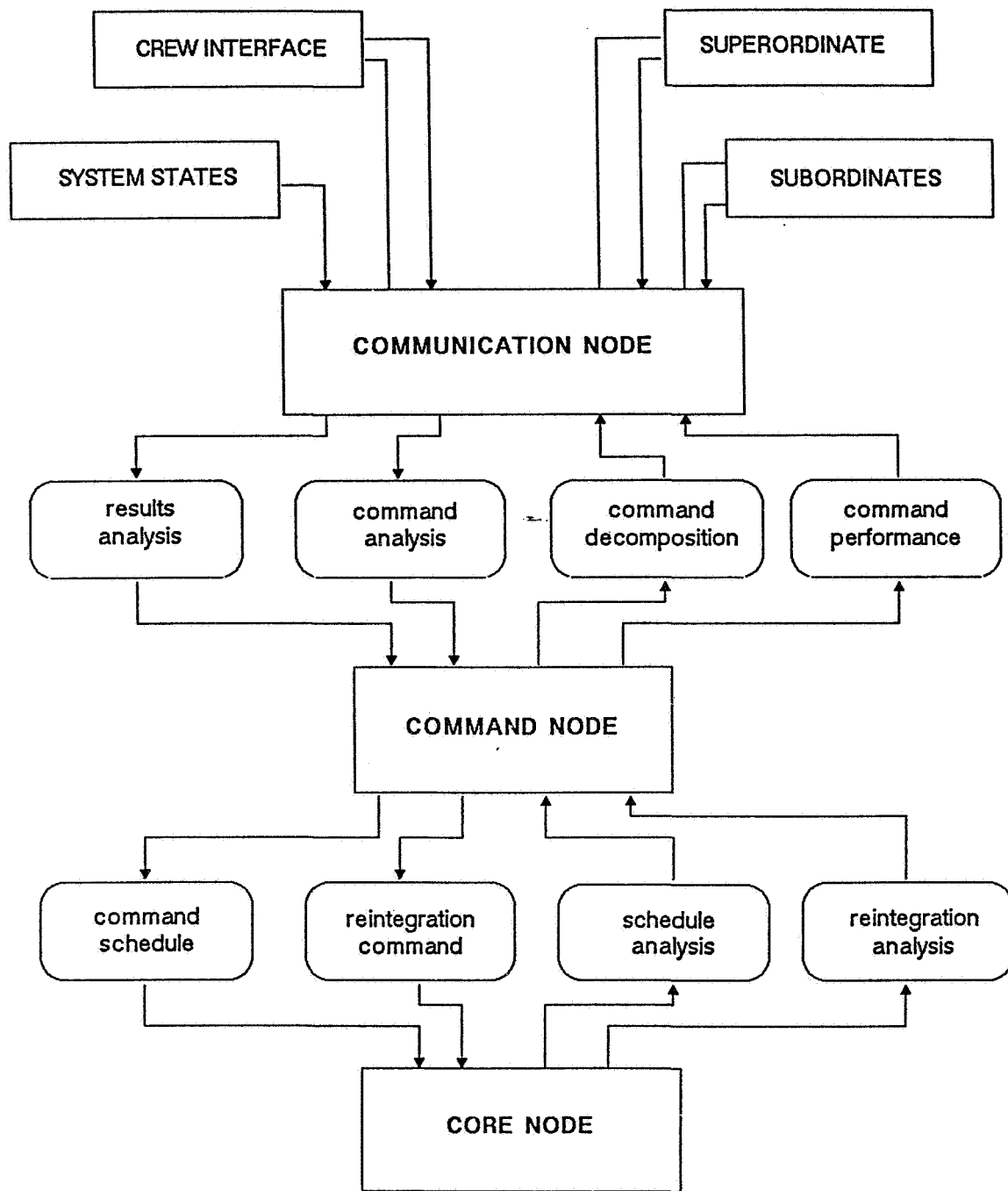
Figure 1: *Supervisory Engine — Demon Control Flow*
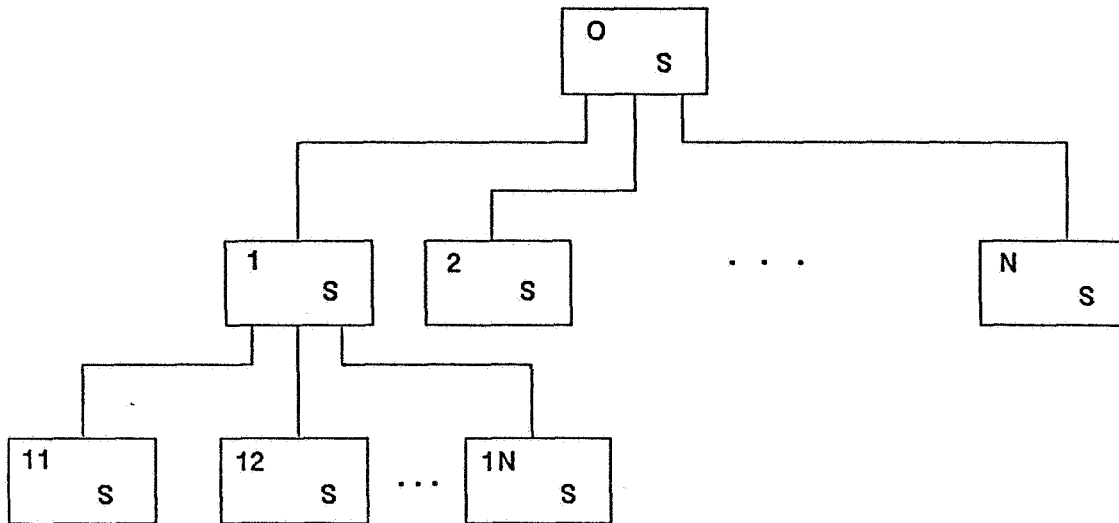
Figure 2: Supervisor Hierarchy — Conceptual Structure
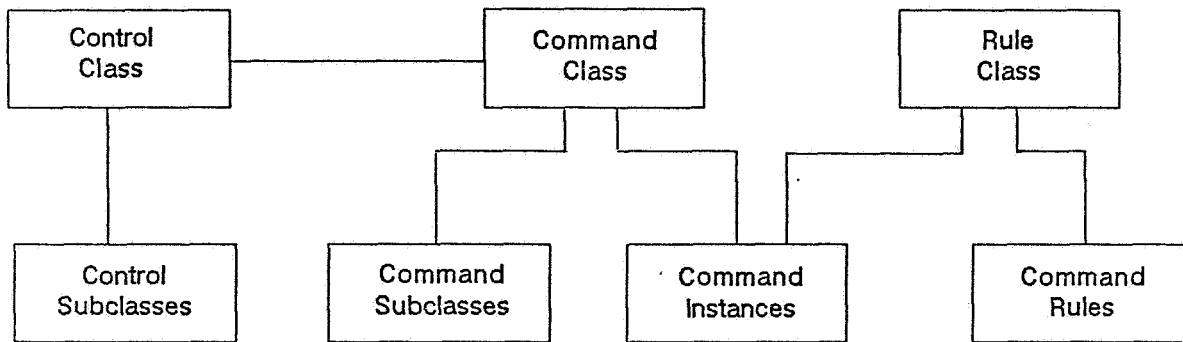
Figure 3: Supervisor Knowledge Structure

```
(local – command – description – i
    ((supervisor – destination – 1 demon – name – 1 subdommand – name – 1
        (argument – list – 1) (condition – response – 1))
    (supervisor – destination – 2 demon – name – 2 subcommand – name – 2
        (argument – list – 2) (condition – response – 2))
    . . .
    (supervisor – destination – n demon – name – n subcommand – name – n
        (argument – list – n) (condition – response – n)))
    (command – argument – list) (command – condition – response))
```

Figure 4: Interior Rule Structure

361
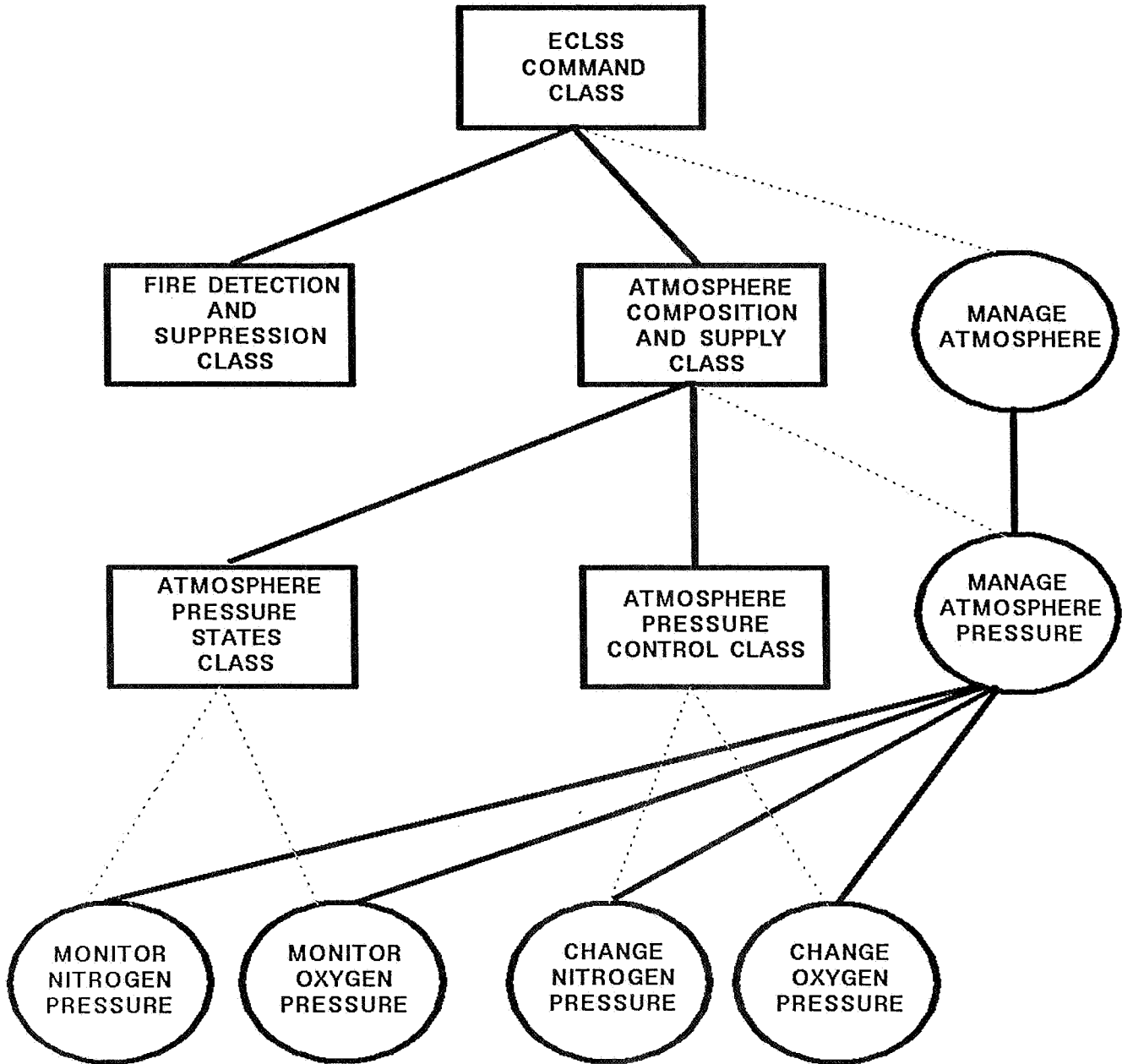
Figure 5: Example — ECLSS Knowledge Base

362

PROGRAMMING MODEL FOR DISTRIBUTED INTELLIGENT SYSTEMS

J. Sztipanovits, C. Biegl,
G. Karsai, N. Bogunovic
Vanderbilt University
Nashville, Tennessee 37235

B. Purves, R. Williams, T. Christiansen
Boeing Aerospace Company
P.O. Box 1470
Huntsville, Alabama 35807

ABSTRACT

This paper describes a programming model and architecture which has been developed for the design and implementation of complex, heterogeneous measurement and control systems. The Multigraph Architecture integrates artificial intelligence techniques with conventional software technologies, offers a unified framework for distributed and shared memory-based parallel computational models and supports multiple programming paradigms. The system can be implemented on different hardware architectures and can be adapted to strongly different applications.

INTRODUCTION

Evolving classes of intelligent systems are based on integrating artificial intelligence (AI) techniques with conventional software technologies. Intelligent measurement and control systems, and test systems are implemented by providing a "knowledge-based control level," (or "knowledge level") for low-level signal processing and control modules. One of our basic goals was to develop a hybrid architecture that would support merging symbolic and numerical computations.

A typical feature of the intelligent measurement and control systems is the computational heterogeneity. Signal processing modules, databases, reasoning agents, etc., have to be implemented and connected to form an integrated system. This heterogeneity requires a programming environment that supports multiple programming paradigms. The other goal of this research was to develop a multi-paradigm programming model directed toward measurement and control applications.

Due to the real-time environment, and to the complexity of application systems, intelligent measurement and control is usually computing intensive. The reasonable way to ensure large computing power is to use multiple processor systems and parallel computing techniques. Computational heterogeneity has an impact on the approach to be followed. Loosely coupled distributed computing models have to be integrated with shared memory-based parallel models in order to ensure proper flexibility. The third major objective of our research is the integration of parallel computing models supporting different computing granularity.

This paper describes a prototype programming environment that focuses on the issues of integration. The Multigraph Programming Model has been

tested in a specific application, the Intelligent Test Integration System (ITIS) [1]. The purpose of the prototype development was to analyze the feasibility and properties of integrated systems. First, we give an overview of the system architecture, then the key concepts and components are discussed.

MULTIGRAPH ARCHITECTURE

The Multigraph Architecture provides a generic framework for the integrated programming environment. Multigraph is a layered architecture having: **knowledge-based layer, module layer, system layer** and **hardware layer.** In our current implementations, the hardware layer and the system layer are provided by a particular computer system and are "external elements" to the Multigraph Architecture. The specific software components are defined and developed for the module layer and knowledge-based layer.

The **hardware layer** might include single processor systems, multi-processor configurations with shared memory architecture, loosely coupled computer networks, and their combination. The higher level computing models can take advantage of special hardware components, such as an array processor or other resources. The prototype system has been implemented on a network of VAX computers [1].

The **system layer** includes an operating system, providing a standardized access mechanisms to the hardware resources. In the case of multiple processor configuration, the system layer facilitates task management, intertask, (interprocessor) communication and synchronization and real-time services. In the prototype configuration, the system layer is the VMS/DECNET operating system.

The **module layer** is an intermediate layer between the knowledge-based and system layers. The primary function of the module layer is to provide a virtual machine for the Multigraph Computational Model (MCM). MCM is a parallel model of computations and is used for supporting medium/high computational granularity on shared memory architectures. The module layer includes a run-time system for MCM, called the Multigraph Kernel. The Multigraph Kernel is a dataflow/demand-flow scheduler, passing the elementary computations to the system resources for execution. A detailed description of MCM and the Multigraph Architecture can be found in [2] and [3]. The other function of the module layer is to separate the higher layers from the operating system services by means of a well-defined system layer interface.

The **knowledge-based layer** supports symbolic computations. The implementation language is LISP, which is extended by the components of a high-level programming model described later.

DISTRIBUTED COMPUTING ENVIRONMENT

The prototype system has been implemented on a VAX network, running under the VMS/DECNET operating system. The structure and main components of the distributed computing environment are depicted in Figure 1.
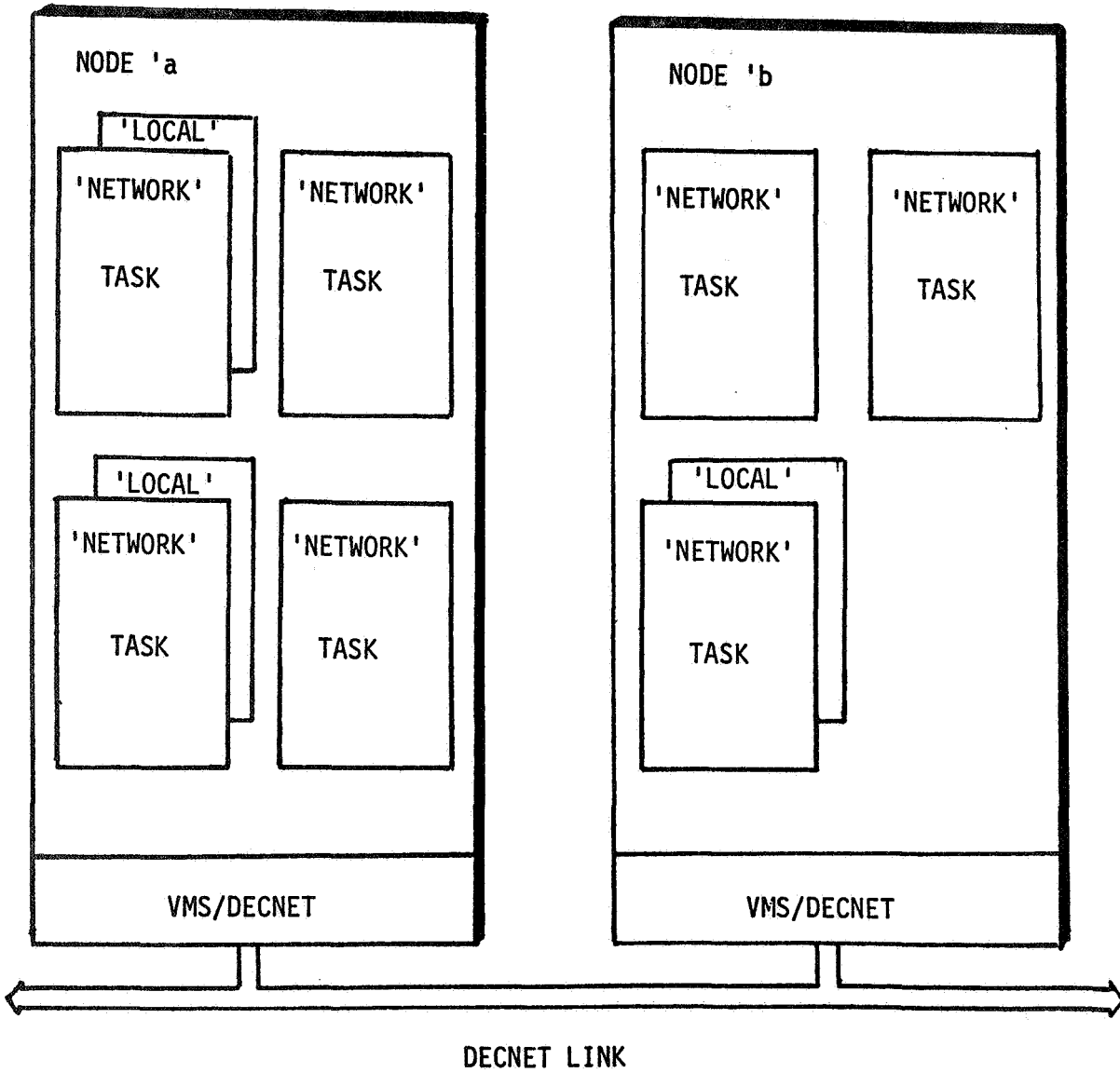
FIGURE 1. Distributed Environment

The different VAX systems represent a 'Node' of the network. The nodes run a copy of the VMS operating system and are linked through the DECNET services. The key DECNET services used in the implementation of the programming model are: 'remote task allocation and deallocation' and 'intertask communication' [4]. These system services are hidden in a LISP kernel (a modified version of FRANZ LISP) and are used for implementing the Communicating Lisp System (CLS) facility [5].

The basic concept of CLS is the 'Task' that represents a segment from the system resources of a particular Node. Each Task provides an execution environment that supports the basic concepts of the programming model and provides interface to the Multigraph Kernel. Tasks can be dynamically allocated and may have different attributes. The attributes determine whether a Task is able to communicate with an Operator through an attached terminal and limit the accessibility of the objects allocated in the Task environment. Tasks with 'NETWORK' attributes might have an attached terminal, and their objects can directly communicate with each other. 'LOCAL' tasks do not have terminal I/O and their objects are accessible only from the objects of their parent Task.

OVERVIEW OF THE CONTROL STRUCTURE

The basic concepts of the control structure and their relations are represented in Figure 2.

The central concept of the programming model is the Autonomous Communicating Object (ACO). ACO is the major system structurization principle for the knowledge-based layer and is an extension of the 'Object' concept of the object-oriented languages, such as Flavor [6]. The main features of ACO's from the aspect of control are the following:

1. ACO's are fully autonomous systems that can run virtually or physically parallel.

2. ACO's can be dynamically allocated and can compete for the same resources. In these cases, their relative priority will determine the resource allocation.

3. ACO's communicate with each other by means of a fully asynchronous message passing protocol. The message transfer and the allocation of objects and new Tasks are controlled by the Communication Managers. Each Task has its own Communication Manager, which is part of their default environment.

4. ACO's do not have global identifiers, the Communication Managers are responsible for maintaining the consistency of the name space.

In the programming model, ACO's serve as a standardized "object shell" around a variety of heterogeneous system components that are built by using various programming paradigms.

After creating an instance of an ACO, a control graph is built in the module layer by using calls to the Multigraph Kernel. The control graph
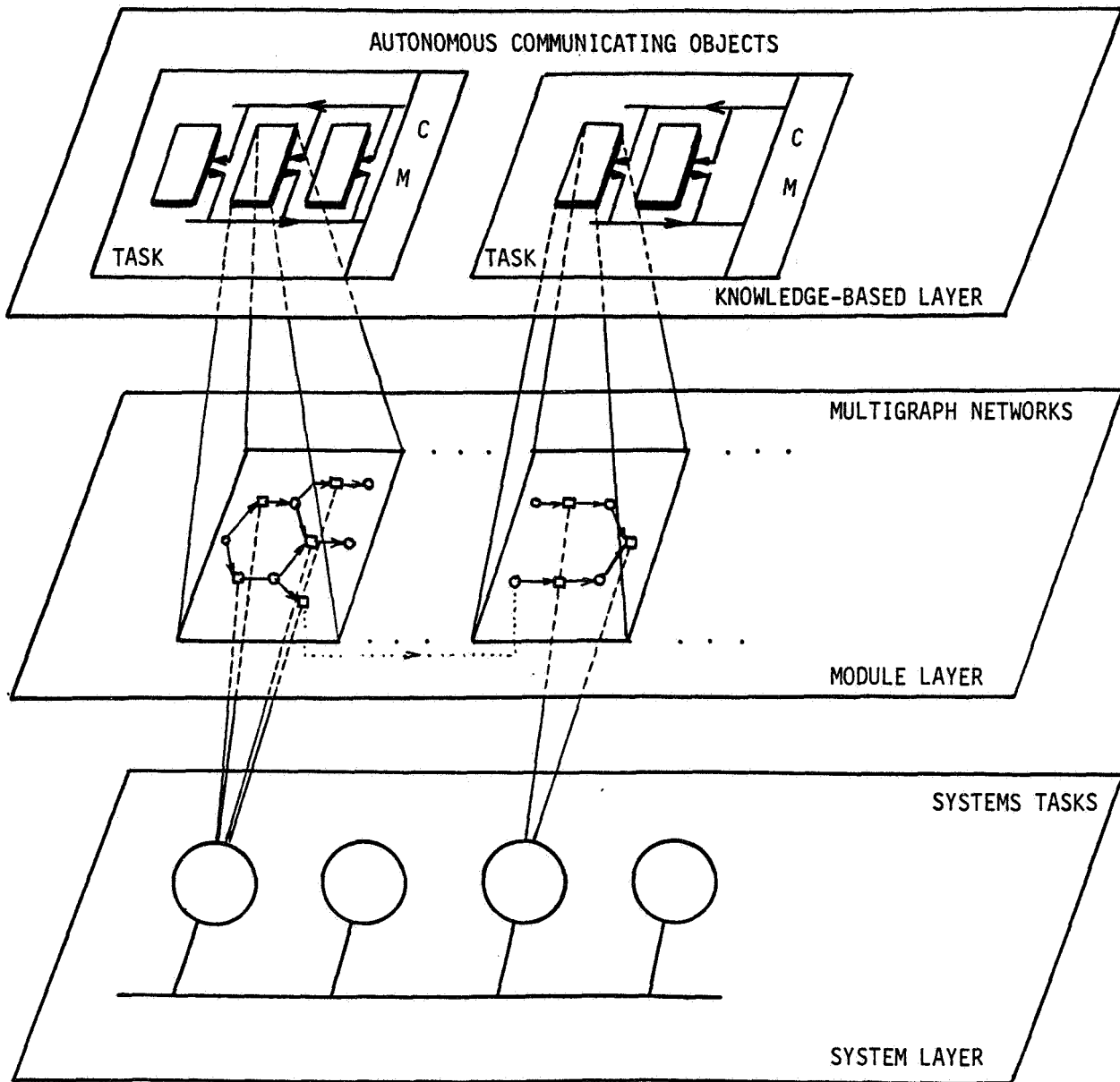
FIGURE 2. Layered Control Structure

represents the internal control structure of the object, and is described by using the concepts of the Multigraph Computational Model. MCM is a macro-dataflow model supporting data-driven and demand-driven control flow. The way in which the control graph is generated depends on the type of the object. Relatively simple incremental compilers build (and modify) the graph, according to the specific knowledge base of the objects.

While the ACO's form the "macro-structure" of the system, the internal control graphs of the objects constitute their "micro-structure." The macro-structure involves large computational granularity, therefore, the allocation of objects can be a relatively expensive operation, and their communication protocol might be robust and safe. The computational granularity on the level of the micro-structure can be significantly smaller. The minimum size of the elementary computations is limited by the switching overhead of the Multigraph Kernel (MK). (In the current implementation, the switching time is less then 200 microseconds on the VAX785.)

The third level of the control structure is provided by the system level. In the prototype system, the controller is the resource scheduler of the VMS/DECNET operating system. The system layer interface of the MK assigns system tasks to control graphs, and schedules the execution of the elementary computations. An important service of MK is to link control graphs that are allocated on different nodes. This possibility ensures that the micro-structure of certain ACO's can be built on multiple nodes, i.e., even a single object can form a complex, distributed system.

As a conclusion, the layered control model supports different levels of parallelism. ACO's are intended to form the macro-structure of the system, and they communicate by using the typical services of loosely coupled, distributed systems. The micro-structure of the objects, described by the terms of MCM, provides much lower computational granularity, and can take advantage of shared memory-based, multiprocessor architectures. The key element of the architecture is MK, which integrates these models in a common framework.

AUTONOMOUS COMMUNICATING OBJECTS

As we previously mentioned, the ACO's provide a unified shell around heterogeneous computational objects. The generic structure of the objects can be seen in Figure 3. Each type has a set of specific 'methods' implementing the communication protocol, and a set of 'methods' for building the control graph of the object. The core of the objects is the "knowledge-base," which is described by a well-specified representation language. Summarized below are the features of some of the basic object types.

Rule Network Object (RNO)

The structure of the RNO's can be seen in Figure 3. The behavior of a particular instance of an RNO object is primarily determined by the rule-set, which is associated with it. The rules are pattern-driven production rules, and are defined by a rule language described in [7]. After receiv-

AUTONOMOUS COMMUNICATING OBJECT
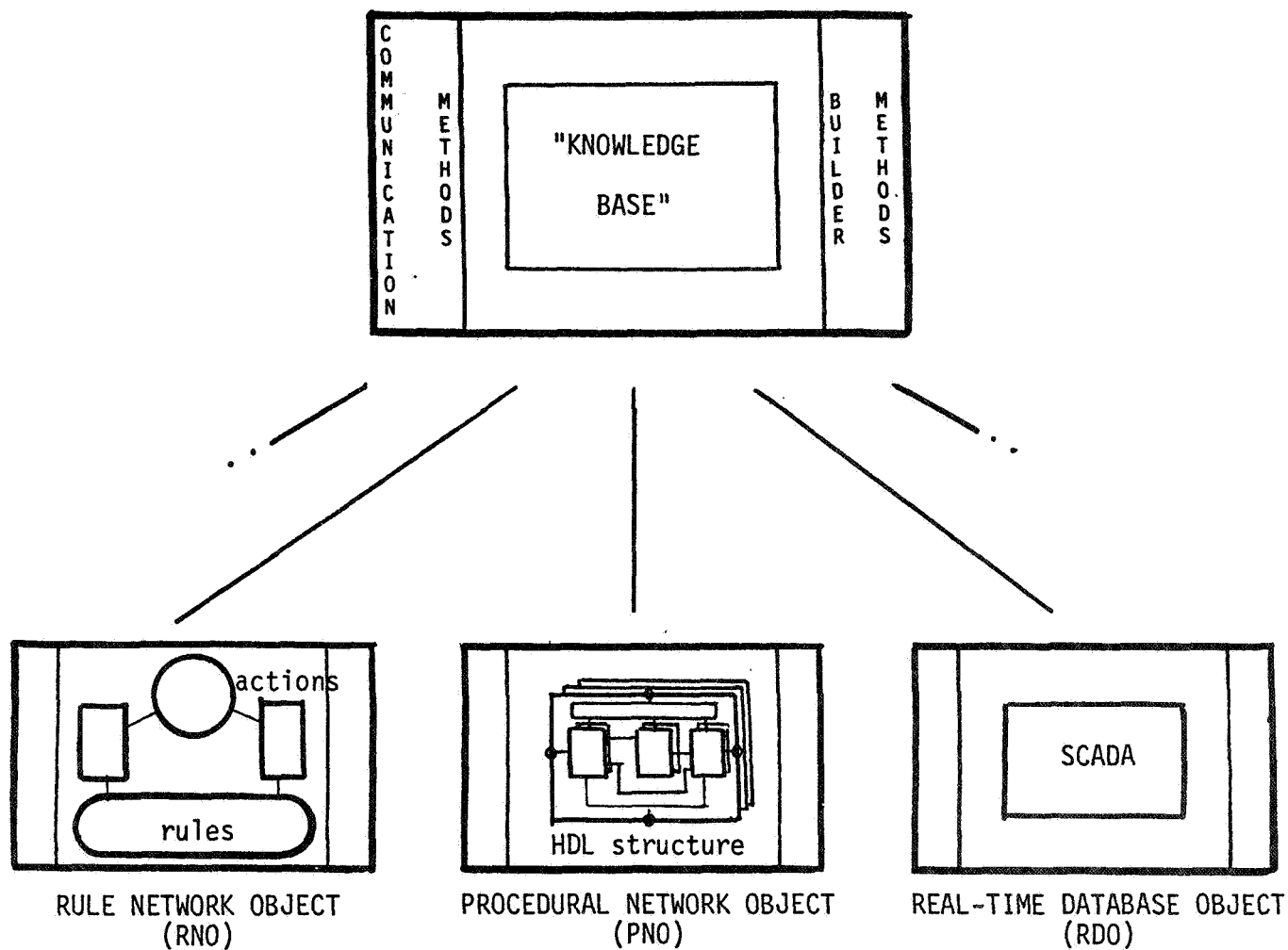
FIGURE 3. Autonomous Communicating Objects

ing a message, the RNO starts a reasoning process driven by the fact included in the message. The reasoning process is forward chaining, and might result in changing the internal state of the object. The state is represented by the factbase of the rule network, and by the instance variables of the object.

The 'builder' method of the RNO's generates the control graph of the object from the rules. In the current implementation, each rule is assigned to a separate node in the control graph, and the rule interpretation runs under the control of the MK using the dataflow control principle. The interpreter implements a modified version of the RETE algorithm. The average speed is 7-10 msec/inference on the VAX/785.

Procedural Network Object (PNO)

PNO's are intended for implementing signal processing systems. The structure of the signal processing system is represented in the Hierarchical Description Language (HDL), and this representation constitutes the knowledge base of a PNO. The HDL descriptions: (1) provide a well-structured, hierarchical representation of the signal flow, (2) support the specification of alternatives on any level of the hierarchy, and the description of the selection rules, and (3) describe the dynamic control interface of the procedural network.

The 'builder' method of PNO's generates the control graph of the procedural network, according to the top-level specifications required. The generation procedure can be considered as an "intelligent" interpretation of the HDL description: the top level specifications are propagated down the hierarchy, and decisions are made regarding the selection of alternatives and allocation of modules. The result of the building process is a network of modules written in any available languages. The network is linked by the control graph, and run under the control of the MK by using the dataflow control principle. An important feature of the PNO's is that they can build networks in the distributed environment. After generation, PNO provides interface to the procedural network, which makes it possible to inject signals in the network, to monitor its operation and to reconfigurate it. A more detailed description of the PNO's can be found in [8].

Real-Time Database Object (RDO)

RDO is an object shell around a real-time database provided by SCADA (Supervisory Control and Data Acquisition System). RDO can insert and retrieve data from the SCADA database, through the standard communication protocol. Since SCADA is a foreign component for the Multigraph programming model, RDO's do not have internal micro-structure, they are mapped into a single node in the control graph. The knowledge base for RDO is the configuration file of SCADA, the builder method is simply the activation of the SCADA builder system.

The prototype system uses the object types described above. A number of other object types have also been implemented, according to the needs of different application systems.

CONCLUSIONS

The Multigraph Programming Model has been developed for implementing complex, intelligent systems. The main focus of the research was to find an architecture and programming model that can integrate in a unified framework symbolic and numeric computations, different levels of parallel programming methods, and various high-level programming paradigms.

The basic concept of the high-level programming model is the Autonomous Communicating Object. The programming model consists of special object types that offer high-level support for the design of complex system components, such as procedural networks and rule-based systems. The prototype version of the programming model has been implemented on distributed VAX configuration, and has been tested in the context of the Intelligent Test Integration System.

REFERENCES

[1] Sztipanovits, J., et al., "Intelligent Test Integration System," Proc. of the NASA Conference on Artificial Intelligence for Space Applications, 1986, in press.

[2] Sztipanovits, J., "Multigraph: Parallel Architecture for Intelligent Systems," Department of Electrical Engineering, Center for Intelligent Systems, Vanderbilt University, Technical Report #86-01, 1986.

[3] Biegl, C., "Multigraph Kernel User's Manual," Department of Electrical Engineering, Vanderbilt University.

[4] Guide to Networking on VAX/VMS, Digital Equipment Corporation.

[5] Padalkar, S., "Communicating Lisp Systems Facility," Vanderbilt University, Master Thesis, 1986.

[6] Moon, D., Stallman, R., Weinreb, D., LISP Machine Manual, The MIT AI Lab, Cambridge, MA, 1984.

[7] Biegl, C., "New Florence User's Manual," Vanderbilt University, 1986 unpublished report.

[8] Sztipanovits, J., et. al., "Programming Model for Coupled Intelligent Systems in Distributed Execution Environment," Proc. of SPIE's Cambridge Symposium on Optical and Optoelectronic Engineering, 1986, in press.

AI and Simulation: what can they learn from each other?

Silvano P. Colombano Ph.D.

RECOM Software Inc.

Contract nr. NAS2-12172

NASA Ames Research Center, MS 244-7, Moffett Field CA 94035

October 16, 1986

Abstract

Simulation and Artificial Intelligence share a fertile common ground both from a practical and from a conceptual point of view.

Streghts and weaknesses of both Knowledge Based Systems and Modeling & Simulation are examined and three types of systems that combine the strenghts of both technologies are discussed: a Computer Aided Modeling system, an Intelligent User Interface and a Model Based Expert System (Embedded Simulation).

These types of systems are a practical starting point, however, the real strenghts of both technonologies will be exploited only when they are combined in a common knowledge representation paradigm.

From an even deeper conceptual point of view one might even argue that the ability to reason from a set of facts (i.e. Expert Systems) is less representative of human reasoning than the ability to make a model of the world, change it as required, and derive conclusions about the expected behavior of world entities. This is a fundamental problem in AI, and Modeling Theory can contribute to its solution.

The application of Knowledge Engineering technology to a Distributed Processing Network Simulator (DPNS) under development at Ames is discussed.

Introduction

In a very general sense the field of Modeling and Simulation is as old as science itself. Indeed, it has been at the very core of the scientific method: examine a phenomenon, make a quantitative model of it, derive predictions from the model, compare the predictions with the behavior of the real system. If necessary, modify the model and iterate until the behavior of the model and that of the real system match, according to some pre-defined criteria.

The advent of computers, until now, has expanded the role of Modeling and Simulation in two ways: 1) it has made possible to compute analytical models that would have been impossible or impractical to do by hand and 2) it has added the possibility of an algorithmic description of systems, such

373

as that typical of discrete event simulations, that would also have been impractical by hand. Just as mathematical languages such as Calculus have given models the precision required for quantitative results, programming languages provide an unambigous and reproducible description of models.

The coming of age of the field of Artificial Intelligence promises to expand the role of Modeling and Simulation even further. This paper explores the ways in which this expanded role may come about.

The field of AI has come about partly as a natural evolution of computer science, stemming from the realization that computers could be much more than powerful calculators, and partly from a desire to explore the ever more compelling analogies between brain and computer. Exploration of these analogies promises an increased understanding of brain function and shows potential new directions for the evolution of computer technology.

Some of the early experiments in computer representation of knowledge have given rise to specific techniques, now embodied in Expert Systems, for capturing the knowledge of experts and for deriving answers to queries in specific knowledge domains. Separation of the knowledge domain from the inference engine has been a powerful technique that has enabled people to make use of essentially the same inference engine for different applications by simply changing the knowledge domain. This has given rise to an exciting new industry, based on practical applications, useful in many areas of both business and industry.

While this industry is enjoying considerable attention and well deserved success, it should be kept in mind that its technological foundations are still rather primitive when compared with the original intent of exploring the realm of what we understand to be real intelligence.

It is not the intent of this paper to enter a controversy on what "real" intelligence is and on the limitations of present day AI technology. I take, first of all, the position that present day AI technology can both enhance and be enhanced by Modeling and Simulation techniques. At the same time, a deeper view of the science of Modeling and Simulation suggests that its goals, and those of fundamental AI research are, to a great extent, overlapping and complementary.

Intelligent simulation shell

Simulation languages have been developed in the same tradition and philosophy of procedural languages. Fundamentally they have provided constructs for describing systems in terms of differential equations, in the case of continuous simulation, and for filing and retrieving events or for coordinating competing processes, in the case of discrete event simulation.

The assumption made is that the system to be simulated is understood well enough to provide specifications for a complete algorithmic description. In this sense simulation is treated like any other kind of application software. But while the assumption of complete specifications is valid, at least in principle, for most application software and for some simulations, much simulation work is of an

exploratory nature, and done to reveal and understand the structure of the system to be simulated. For this type of work, simulation languages are helpful only in the sense that they speed up the programming process with high level constructs. It seems to me, however, that the tools provided are not especially suited for quick exploratory changes in the structure of the program.

Even in the case of general software engineering the suggestion is often made that languages should be based on the assumption that all software will continue to be modified. Much modern programming philosophy is oriented towards this fact. Indeed, most of the software cycle is spent maintaining (i.e. modifying) any given program. One approach taken in software engineering is preventive. The attempt is made to tighten and formalize the specification process so that the chances of errors, or that modifications will be required, is lessened. Unfortunately, cases where specifications can be completely fixed before the programming effort begins are more the exception than the rule. In simple terms, typically people do not know exactly what they want until they start seeing some concrete rendition of their idea. For this reason the concept of rapid prototyping is increasingly being favored over that of hard specifications.

While an iterative cycle is unavoidable for the production of most software, it is actually an essential part of the construction process when the software is a simulation designed to better understand a system under consideration. This is by no means the only type of simulation done, but it may be the most common. In this case, the very reason for creating a simulation is to experiment with changes in the underlying model. Some changes may be only parametric, but often they require restructuring of the model itself.

In order to make reasonable changes, and to understand their consequences, a good understanding of both the underlying system and of the assumptions made in the simulation are required. In addition, if the simulation is successful, this understanding will increase and will, most likely, lead to new experimentation and new changes.

Essentially, I am stating that a feature that distinguishes simulation from other kinds of software development is that, in simulation, the develpment process often turns out to be more important to a project than the final product. I believe that modern simulation environments should reflect this fact.

Let's call such environment an Intelligent Simulation Shell. The word intelligent refers to the fact that the shell will incorporate several types of knowledge, for example knowledge about

1. the process of modeling and simulation

2. the level of abstraction chosen for the model

3. the underlying assumptions of the model

4. the syntax of the underlying modeling language

5. hystory and relationships among the different models constructed to solve the problem

The representation of these types of knowledge will be based on models and meta-models, rules and

meta-rules embodying generally complex relationships among all these elements.

Both simulation and artificial intelligence deal separately with all these forms of knowledge representation, but coherent integration is still a long way off. One way to attack the problem is to look at simulation and AI as inherently separate tools and see how their respective strengths can be combined.

Strengths and weaknesses of the two technologies

Static objects

Objects whose attributes don't change in time are a strong feature of KBS. Typical examples might be genealogical relationships and high level decision rules.

Dynamic objects

Objects whose attributes do change in time have, until now, been mainly the province of Simulation.

Inference engine

The concept of an "inference engine" distinct from the knowledge domain has probably been the single most important factor in the growth of KBS technology. No counterpart exists in simulation. The model representation is typically interwoven with the algorithm that embodies the simulation, and this is precisely what makes simulation software more similar to ordinary application software than to KBS.

Validation

Validation is really a problem for both simulation and KBS, but dealing with the model of a system that is at least intuitively understood is a great help, even if the real system has not yet been constructed. The type of knowledge typically embodied in KBS is a set of rules that, occasionally can even be found to be in contradiction. The trustworthiness of decisions that may be based on these rule can only be assessed by comparison with a real system, but even in this case, lack of an underlying model makes the methodology very difficult.

Construction methodology

Whereas the process of modeling and simulation can be approached with some mathematical rigor (e.g [6]) building KBS is limited by the ability to extract an expert's high level knowledge and to codify it as a set of rules that can be used by the system to reach conclusions. Present day methodology requires the presence of a human expert whose knowledge the KBS is trying to capture. It can be argued that viewing this as a weakness of KBS is really unfair. Indeed, in cases where one wishes to retain knowledge that exists only at a high level in a human expert and which cannot be extracted from a suitable model, this becomes a strength, and this is really the reason for the relative success of KBS. There is, however, a problem of perception that must be addressed. It seems to me that the exciting notion of Artificial Intelligence and Expert System is promoting the view that this type of methodology is somehow superior to that of mod-

eling and simulation, and, for this reason, problems that should really be approached with M & S techniques are forced into a KBS mode. The attitude seems to be: You have a problem? Build an expert system.

Combining the two technologies

Ideas for a taxonomy of possibilities in combining the two technologies have been proposed (e.g. [4]). Let us consider some of these in light of the present strengths and weaknesses of Simulation and Knowledge Based Systems (KBS).

User interface

One obvious way to combine some of the strengths of both tools is to use a KBS as a user interface to a simulation. User interfaces to simulations range from simple data files that must be carefully written by the user by means of an ordinary editor, to sophisticated menu systems with sophisticated error checking. In setting up complex simulations it is generally easy to enter unacceptable parameters and obtain a meaningless run. Simple error checking of parameter ranges is often not sufficient, since the acceptable range of one parameter may depend on a previous choice of another parameter. In general one must implement a set of rules. Therefore, it appears that a KBS, with its separation of knowledge base from inference engine, would be an ideal tool for a user interface. This separation of knowledge base from inference engine implies the following: the ability to easily enter new rules or to modify old ones, and to present an explanation of the

result when so requested.

In addition, a KBS could be used to select from typical simulation reports only the items requested by the user in a high level conversational mode.

More sophysticated applications of a user interface would generally require more integration between the KBS and the Simulation, than the ones envisioned here. I will refer to some of these in the next section. But, even at this simple level, there are difficulties in utilizing present day Expert System shells. I will discuss some of these below.

Computer aided modeling and simulation (CAMS)

The idea of expert user interface can be carried much further if one includes the concept of modeling among the activities that can benefit from computerized aid. Often the words "modeling" and "simulation" are used as synonyms, but, strictly speaking, the simulation is the algorithmic implementation of a model the was specified without the aid of a computer. Typically, the modeling process requires skills in the art and science of Modeling and Simulation and a understanding of the knowledge domain pertaining to the system being modeled.

It is interesting to note that, whereas a clear separation of knowledge engineer and domain expert has been established in AI, the simulationist is typically a domain expert who picked up some modeling and simulation skills along the way, and often implements his/her model in a generic high level language, sometimes finding it advantageous to use a simulation language. This state of affairs is

377

reflected in, and may be a consequence of, the way these disciplines are taught: AI by computer science departments and Simulation by departments that view simulation as one of their tools, such as Engineering, Economics, Operations, etc. As a result, simulation is almost invariably taught from the point of view of a particular discipline, and uses existing tools instead of being a source of new ways of thinking in computer science.

In the CAMS concept, enough modeling expertise is built into a tool to guide a domain expert to build increasingly sophisticated models of a system under study. The distinction between this type of system an the user interface described above is that, in CAMS, the actual structure of the model is being changed in order to answer new types of questions, in the user interface paradigm, on the other hand, the model can only be changed parametrically. The former emphasyses, and takes advantage of, the learning that occurs as a result of the modeling process within a domain of interest; the latter is only concerned with running, and learning from, a particular model.

Examples of interesting attempts to implement the CAMS concept are KBS (which here stands for Knowledge Based Simulation) at Carnegie-Mellon [5] and two Expert Systems for design and simulation developed at Los Alamos National Laboratory [1]

Embedded simulation

Another extension of the idea of Expert User Interface is a system where one or more simulations run transparently, as required by the Expert System. Examples of this concept have been implemented (e.g [3]) and this type of system is often referred to as "model based expert system". Conceptually it is very simple: when a rule needs the value of a variable that is time dependent, a simulation that will produce the value of the variable at the required time is triggered. This simulation algorithm could be an analytical formula or a discrete event simulation.

This is an acceptable way to proceed, in the absence of a better paradigm for treating time independent knowledge (or rules) and time dependent knowledge (or models) on an equal footing. The difference, as pointed out above, is that the rules are separate from the inference engine while the model based knowledge is still built into the simulation algorithm.

Simulation models as a form of knowledge representation

I have pointed out some ways in which using both simulation and AI technology can enhance the solution of problems in both fields. This is a good way to start, but it doesn't get to the root of the real problem, or opportunity.

Expert Systems came about as a result of learning how to separate the knowledge base from the "inference engine". As long as the two remained inextricably interwoven Expert Systems were not impossible, only much harder to build. Having accomplished this separation, it appears that we have extracted something we can call "knowledge", in the form of rules or other ways in which some facts

relate to other facts. Conceptually this is very general, but, as implemented in present day KBS, it seems to refer only to relatively simple high level rules.

Even though the field of computer simulation is older, this separation has never occurred, perhaps because the need was not perceived. But now simulation can learn from the success of KBS. Models are a very rich form of knowledge representation, and it would be equally advantageous to the field of simulation to be able to separate the model representation from what, by analogy, could be called the simulation engine that exercises the model. The main advantage would be the ability to create environments where models are quickly changed, refined and exercised as needed.

To some extent, elements of this separation already exist in some high level constructs of simulation languages, where, for instance, devices and entities with their attributes are defined. But the systems do not have the "intelligence" to automatically propagate the consequences of any changes, or to warn or guide the user. These ideas can only be implemented if working from a common framework where knowledge is a mixture of time dependent models and of rules that can be both time dependent and time independent, and where AI and Simulation techniques are integrated.

Modeling and simulation as a fundamental problem in Artificial Intelligence

So far I have dealt with both AI and Simulation as a set of techniques which, when integrated, could yield new powerful tools. For my final point I want to go back to the statements made at the beginning of this paper, that the original purpose of AI research, at least in the mind of many, is to understand the phenomenon we call intelligence and to apply this understanding to computer technology. It seems to me that although AI technology has made great progress in creating systems that can capture human knowledge and deliver it in a way that is most natural to us (i.e. in the form of a dialog), it has not made much headway in increasing our understanding of intelligence. This is an area where a symbiosis of AI and Simulation can be most fruitful.

I would argue that ability to reason from a set of facts, as embodied in present day KBS is less representative of human reasoning than the ability to make a model of the world, change it as required, and derive conclusions about the expected behavior of world entities. Interestingly, this ability is of the kind envisioned in the CAMS concept. In CAMS, the modeling process would be guided by a user, but this would be a natural first step in trying to understand the thought processes and mechanisms required in creating and refining models.

It is really not surprising that "expert" thinking as been easier to model than "common sense". Knowledge in narrow domains is typically of the

kind that is amenable to representation in KBS. Common sense thinking requires a broad model of the world, and the flexibility to refine or generalize su^b-models as required. If we learn how to aid humans in this process we may also learn how to begin to make it automatic.

Experience with a simulator for distributed processing

At Ames, in the Information Sciences Division, we are dveloping a "Distributed Processing Network Simulator" (DPNS). Its intended use is to help in the design and evaluation of alternative configurations of the space station data system.

Our initial cut at the system is a simulation written in ECSS, a high level simulation language based on Simscript and taylored to computer systems. In this system, the user is given the ability to define interactively a configuration of several LANs, connected by bridges or gateways. For each LAN the user can define the number of nodes, and, for each node the type and number of devices and their characteristics. In addition, the user can run simulated jobs anywhere on the network. All choices are made by menu selection.

Definitions for each run are stored in a data base, and can be recalled and modified. To a very limited extent this system embodies some CAMS concepts, namely, the ability to modify the structure of the model and to keep track of past runs. This is state-of-the-arts for simulation, but structure changes are limited to a very specific class of models and components. Also, the number of components can change, as well as whatever characteristics have been parameterized, but the level of resolution considered for each components cannot be changed.

DPNS has brought to light another interesting problem: as the flexibility and generality of the system increases, so does the possibility of constructing structures that could not actually function in reality. The problem is similar to that of configuring a real system and forgetting to provide a cable or a correct interface. This may not be obvious for some configuration, in fact this might be one of the reasons for doing a simulation. But one would like to eliminate the possibility of making the kind of mistakes that are simply due to our innate inability to keep track of many things at once.

We looked at some common expert system shells with the idea of using "what's out there" in the expert user interface mode described above. Unfortunately, the expert system shells we have examined run in a consultation mode that is not appropriate for this type of problem.

Typically, in this mode, a knowledge engineer would have prepared a set of rules that will enable the system to answer a question of the type: can component A be connected to component B?. In order to answer the question the system would search its own data base and/or prompt the user for information on components A and B, and finally deliver the answer. This type of system is passive, in the sense that user has to make the enquiry. In addition, the system would gather information either from a prepared database (i.e. a fixed model) or would enquire of the user by means of a dialog.

380

The consultation mode appropriate for this type of problem is very different in the following ways:

1. a user would not want to keep asking "can I connect A to B?", rather s/he would want to simply connect A to B, B to C, etc.

2. a user would want to be interrupted if and only if one of the connections is not allowed

3. at this point a user would either recognize the mistake or would demand that the system provide an explanation

4. the user would certainly not wish to inform the system via a long dialog on what had been done up to that point, rather s/he would expect the system to know everything done until then

This type of system would need to be building its knowledge base as the user proceeds in constructing the model, would actively warn the user of problems with the model, and would be able to provide an explanation of the nature of the problem when so requested.

Conclusions

Even though AI and Simulation technologies are, at present, quite separate, both in terms of theoretical foundations and tools, there are practical ways of combining both to build highly synergistic modeling systems. Additional advantages to both fields will be gained from a common integrative approach to knowledge representation.

Bibliography

[1] Aldridge, J.; Cerutti, J.; Draisin, W.; and Steuerwalt, M.; *ExpertSystems for Design and Simulation* , AIAA/NASA Symposium on Automation, Robotics and Advanced Computing for the National Space Program, Los Alamos National Laboratory, Contract nr. LA-UR-85-2838, 1985

[2] Andrews, P.A.; *A System for Predicting the Behavior of Forest and Range Fires* In: SCS Conference on Computer Simulation in Emergency Planning; Simulation Councils Inc., 1983.

[3] Colombano, S; Doubek, S.; DuBois, D. and Weinstein, K.; *Distributed Processing Network Simulator (DPNS) - Version 1.0 User Documentation* , RECOM Software Inc., Contract nr. NAS2-12172, NASA-Ames Research Center, 1986.

[4] O'Keefe, R.; *Simulation and Expert Systems - A Taxonomy and some examples* , Simulation 46:1, 1986.

[5] Reddy, Y.V. and Fox, M.S.; *KBS: An Artificial Intelligence Approach to Flexible Simulation* , The Robotics Institute, Carnegie- Mellon University, Pittsburg PA, 1982.

[6] Zeigler, B.P.; *Theory of Modelling and Simulation* , John Wiley & Sons, New York, 1976.

SOFTWARE DEVELOPMENT WITHOUT LANGUAGES

Haywood S. Osborne
Teledyne Brown Engineering
Huntsville, Alabama

ABSTRACT

Automatic programming generally involves the construction of a formal specification; i.e., one which allows unambiguous interpretation by tools for the subsequent production of the corresponding software. Previous practical efforts in this direction have focused on the serious problems of:

o Designing the optimum specification language,

o Mapping (translating or compiling) from this specification language to the program itself.

The approach proposed in this paper bypasses the above problems. It postulates that the specification proper should be an intermediate form, with the sole function of containing information sufficient to facilitate construction of programs and also of matching documentation. Thus, the means of forming the intermediary becomes a human-factors task rather than a linguistic one; human users will read documents generated from the "specification", rather than the specification itself.

In the past few years, the Artificial Intelligence category known as "Automatic Programming" has begun to produce practical outputs, as typified by the advancement of expert-system shells such as ART and KEE, and in an intellectually separate vein, in ongoing improvements to the "Programmer's Apprentice" at MIT[1,5]. The sister field in the software engineering discipline, rather more appropriately labelled "Computer Aided Software Engineering" (CASE), is also beginning to sprout products, such as TAGS, Cadre, and IDE.

A common thread which runs through these disparate approaches is the idea that the form to be manipulated by the developer should be, not the software itself, but the "formal", specification for the software. One form of this paradigm is expressed particularly clearly by Balzer (1983)[2,3]. The mode of working resembles the current way of building microcircuitry, via CAD systems, and is so labelled in Figure 1.

THE C.A.D. APPROACH TO SOFTWARE DEVELOPMENT
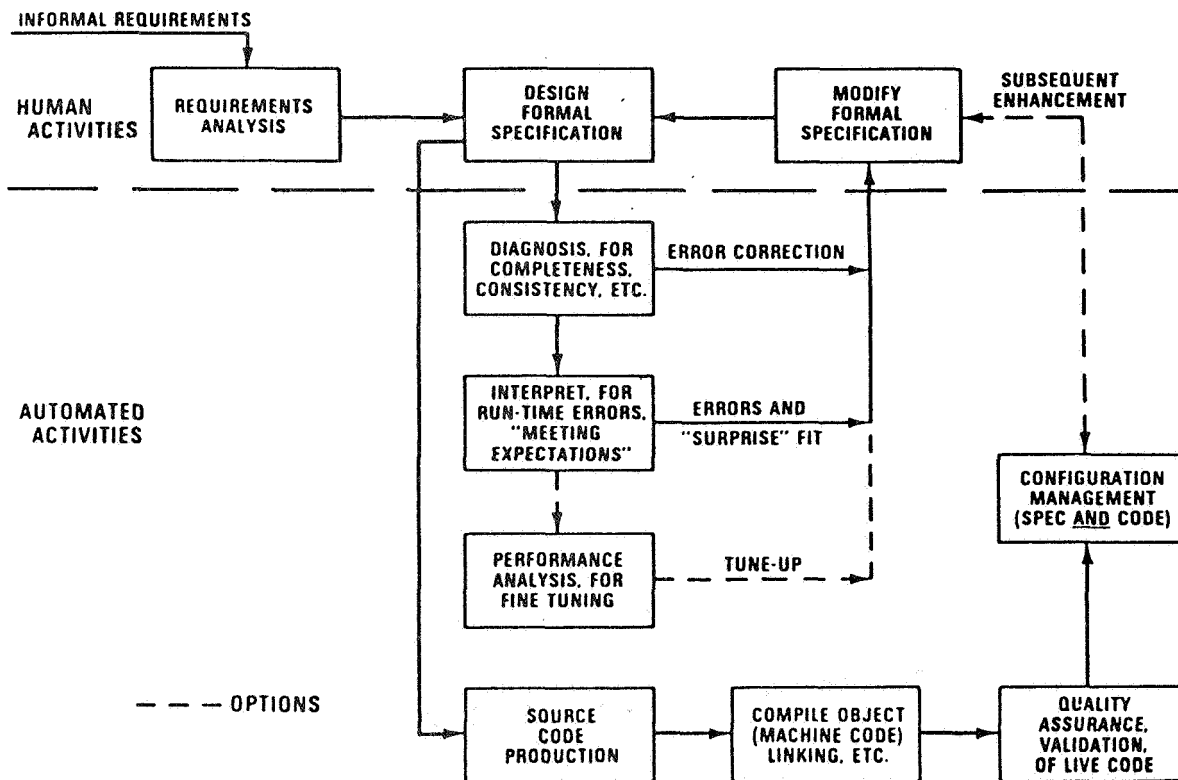
In the C.A.D. way of working, as illustrated in Figure 1, beyond the gathering and understanding of requirements, the specification is the major entity actually worked with.

o   The specification is composed;

o   Software tools diagnose for incompleteness and inconsistencies, and indicate such. The designer modifies the spec until formal criteria - satisfaction of the diagnostic tools - are met.

o   The tools may interpret the formal specification, pointing out errors of detail which could not be found earlier (e.g. values becoming out of range; timing errors). In response, the designer modifies the specification.

o   Once it "works", and the pieces work together in real or simulated modes, it is tuned to make it work well. Tools should measure simulated timings, "dead code" etc., in fashions analogous to a microelectronics C.A.D system's logic simulation, normally performed today in VLSI chip design. Again, the designer modifies the specification to meet requirements.

o   After the specification is satisfactory - and only then - source code is produced (by tools), and normal compilation and linking proceed; in early phases of C.A.D software development, programmers must still escort the deliverables through the process.

o   In later stages of C.A.D. system development, generation of test data may complete the picture.

A recurring problem in this approach is: what constitutes sufficient "formality", with respect to the language of the specification? In one line of reasoning, the ideal specification language would be English. From a human factors standpoint, that may well be true; but the tools which manipulate the specification must then deal with natural language problems as part of the man/machine interface, with respect to "understanding" the specification, and, also, the mapping of the specification to a programming language. Both efforts seem bound to remain in A.I. laboratories for some time to come.

The opposite line of reasoning characterizes most of the currently available products. A very high level language is provided, with the intention of gathering a knowledge base describing the target system in a sufficiently rigorous manner to facilitate "emission" of either matching code or an inference engine with an appropriate complement of constructed rules. In this approach, unfortunately, a human-factors problem begins to dominate: no matter how high the level of the language there comes the moment when the analyst or developer must transfer his attention from the problem domain to the medium of expression; and the activity performed from that moment is "programming" (Smoliar and Barstow, 1983)[4].

The technique proposed here is a bypass of the problem, rather than a theoretical solution. It may be summarized thusly:

o Commit tool-development resource to the construction of a syntax-directed editor, which would allow multiple varieties of input, as required by a human-factor analysis of the given specification language.

o Having sufficiently obscured the details of the language, with respect to the demands made on the user, tailor the language with the aim of allowing an internal representation which facilitates construction of tools with two major sets of functions.

a) Mapping to an existing programming language for execution-time efficiency;

b) Mapping to a diagrammatic scheme of graphic representation, to maximize feedback to the developer, in multiple formats for human efficiency.

Two important practical consequences arise from the approach:

o Facilitation of the Balzer (1983) C.A.D. paridigm becomes the focus of the language design effort, rather than the usual search for compromises between what a man can learn with some efficiency and what a tool can parse unambiguously.

FIGURE 2

START

TO __ DISK
CREATE
SPECIFICATIONS AND
MODIFY AS NEEDED

UTILIZE
THE MANY THINGS
WHICH CAN BE DONE,
MECHANICALLY, WITH
A SPECIFICATION ARE
DESCRIBED HEREIN

TO __ USER
DISPLAY
IN ANY OF MANY FORMATS,
CHOSEN BY THE USER
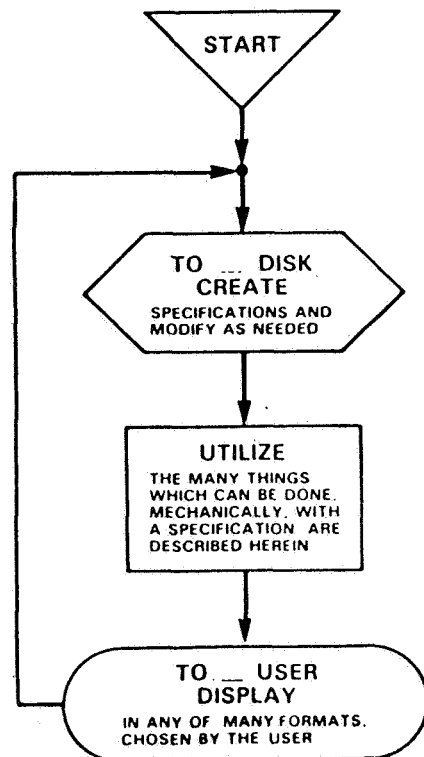
THE "INPUT/PROCESS/OUTPUT"
FLOWCHART, WHICH IS A GENERIC
DESCRIPTION OF ALMOST ANY-
THING, SAYS SOMETHING MEAN-
INGFUL IN THE CONTEXT OF AN
AUTOMATED DESIGN SYSTEM. IT
SAYS, "LOOK AT EACH STAGE AS A
SEPARATE PROBLEM, WITH A VERY
INDEPENDENT SET OF SOLUTIONS."

o The language "disappears" from the system, in the sense that the input medium becomes progressively detatched from the comprehensive - and user modifiable - choice of output displays. The quality of the C.A.D. system, therefore, is measured more by the human-factor design of its tools than the arguable attributes of the specification language it- self. This writer would propose that the creation of effective tools is a more tractable problem than the quest for the Perfect Representation[6].

In a "no-language" model of a C.A.D. system for software development, the role of human factors, rather than a specific representation language be- comes paramount. The illustrated system does utilize a "language", but it is buried in a syntax-directed editor which is part of the key to this kind of approach.

The heart of the package's concept (Figure 2) is that the stereotypical "input-processing-output" division actually reflects a meaningful split of necessary functions, each with very different requirements.

o Input has the primary requirement of rapidity, so that ideas can be captured before they are lost, and before the very act of inputting impedes the mental/logical flow of the next idea. Rapidity of input may take different forms. While one is becoming familiar with the system's editing, it should guide him without making him worry about "correct rules"; because if it does that, then no matter how high the level, the activity is still programming (Smoliar and Barstow, 1983). After becoming familiar, rapidity is, in general, fewest keystrokes possible (or, in a later in- carnation of tools, fewest utterances possible).

o Processing, on the other hand, should cater to the efficiency requirements of the surrounding environment. The internal layout of the data is never to be seen by humans; so, it is uncompromisingly tailored for tool performance. Speed counts. A wide variety of tool support counts. But, there is no direct human interaction with the internal format, so one caters to tools only.

o Output requirements of the package are also more closely bound to human factors than to the correctness of the internal language. Output displays in all forms are feedback to somebody: feedback to the designer may mean quickly recognizable error displays, to augment the "input" con- siderations described above; feedback to a user means the clearest possible depiction of the system's behavior, imply- ing a great variety of attractive displays (i.e., docu- mentation) which breakdown to a variety of levels, to the ground floor, if necessary. Each aspect of the system, and each aspect of its utilization, should have its own associated displays.

FIGURE 3

NOTICE THE ABSENCE OF A NORMAL
LANGUAGE. "SYNTAX DIRECTION"
COMPONENT CONTAINS A COMPILER, OF
COURSE; BUT IT MANIFESTS THE "LAN-
GUAGE" RULES AS MUCH BY PRESENTING
PULL-DOWN MENUS IN A GIVEN ORDER AS
BY SQUAWKING ABOUT ERRORS; SO THE
USER WON'T THINK OF IT AS A LANGUAGE
... OR NEED TO KNOW IT.

EDITOR
HUMAN FACTORED
FOR RAPID GENERATION
OF SPECIFICATION.

RAW _ SPEC

SYNTAX _ DIRECTION
BESIDES ERROR CHECKING,
CONTAINS RULES TO DICTATE
ORDERS AS DISPLAY

CORRECT _ SPEC

SPECIFICATION

CORRECT _ SPEC$_4$

CODE _ GENERATOR
LAST AND, IN SOME
WAYS, LEAST

ERROR _ ILLUMINATIONS

ALL _VISUALS   CORRECT _SPEC$_3$ CORRECT _ SPEC$_2$ RESULTS _BASED _ KNOWLEDGE

DISPLAY _ GRAPHICS
ONE OR MORE WINDOWS,
WITH DIAGRAMS, TABLES,
AND PERFORMANCE GRAPHS.

ILLUMINATES

INTERPRETOR
PROVIDES MACHINERY
TO DRIVE DISPLAYS OF
THE RUNNING SYSTEM.

RESULTS

PERFORMANCE _ ANALYSIS
USES DATA ACQUIRED AT RUN TIME
TO INFORM USER, AND, LATEST, TO
MODIFY SPEC.

GRAPHS _ AND _ LISTS

Note, in passing, that the illustrations in Figures 2 and 3 could be
legitmate outputs from a "no language" C.A.D. system. Figure 3, for
instance, is a high-level "schematic" style of diagram, showing six
separate components -- in this case, programs and a major data file;
the components would each have corresponding lower-level functional
diagrams, and the layout of the data file would appear in a tabular
description.

The implication of a no-language system, in this display context, is that
a variety of outputs could be user-tailored to express or repress details,
to cater to the particular audience, whereas a pure language-based system
would avoid any tailoring because the display is the language.

REFERENCES

1. C. Rich, H. Shrobe, "Design of a Programmer's Apprentice", in
    in Artificial Intelligence: An MIT Perspective, P. H. Winston
    and R. H. Brown (ed), Vol. I. MIT Press, 1979

2. R. Balzer, Automatic Programming, Institute Technical Memo,
    University of Southern California/Information Sciences Institute,
    Los Angeles, Cal. 1973.

3. R. Balzer, T. Cheatham, C. Green, "Software Technology in the 1990's:
    Using a New Paradigm". Compute, Vol. 16, No. 11, 1983.

4. S. Smoliar, D. Barstow, "Who Needs Languages and Why Do They Need
    Them?" Comm. ACM, 1983 (reprint: ACM 0-89791-108-2/83/006/0149).

5. H. Hindin, "Intelligent Tools Automate High-Level Language Programming",
    Computer Design, May, 1986.

6. H. Simon, "Whether Software Engineering Needs To Be Artificially
    Intelligent", IEEE Trans. Soft. Eng., Vol. SE-12, No. 7, 1986.

ORIGINAL PAGE IS
OF POOR QUALITY

KNOWLEDGE BASED PROGRAMMING ENVIRONMENTS - A PERSPECTIVE

Ashok T. Amin
Computer Science Department
University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT   Programming environments is an area of
recent origin and refers to an integrated set
of tools, such as program library, text edi-
tor, compiler, and debugger, in support of program
development. Understanding of programs and pro-
gramming has lead to automated techniques for
program development. Knowledge based programming
systems using program transformations offer
promise of automatic programming, and may have
significant impact on future program develop-
ment methodologies. A review of recent develop-
ments in the area of knowledge based
programming environments, from the perspective
of software enginnering, is presented.

1. INTRODUCTION

Software engineering is a discipline devoted to issues in
construction of large scale software systems. It was brought
about in late sixties to address the needs of software industry
for effective management and development of software systems
which were progressively getting larger and more complex [9].

It was recognized early that computers must be used to
facilitate the tasks of managers and developers. Widely used
model for software system life cycle identifies following phases:
requirements, design, implementation, testing, and operation and
maintenance. Early efforts for development of computer aided
tools has been directed towards support of a specific phase
of software system life cycle. However, the need for an in-
tegrated set of tools for the life cycle support, rather
than for a specific phase, soon became evident. Software
engineering environment refers to such an integrated support
for software development activity. The programming environ-
ment refers to integrated set of tools in support for pro-
gram development, namely, the program design and implementation
phase.

Basic elements for a programming environments include the
program library, text editor, compiler, and debugger. Some pro-
gramming environments, such as UNIX, provide support for pro-
totyping, symbolic debugging, performance analysis, and por-

tability checking. Primary objectives for a programming environment is to facilitate timely development of error free programs. As the area of programming environment continues to evolve, the emphasis is appropriately shifting from the support of low level activities to automation of low level activities and support of higher levels of development activities. At the higher level, a programming environment may be judged based on its support of spectrum of methods for design of programs [5]. Such environments must incorporate programming knowledge.

In syntax directed programming environment [10], the knowledge about syntax of the programming language is incorporated into the program editor which provides templates for various programming constructs. The programmer supplied text is checked for error for each construct as it is completed. This facilitates development of error free programs, and supports a higher level of programs than that of a set of characters.

Program library facilitates program development by enabling reuse of program code and thereby supports timely development of reliable programs. Program libraries are language dependent. However, program design information, suitably represented, can be reused across programming languages. A library of plans, where a plan is a representation of program design information, coupled with a plan editor provides for the support of program design in the Programmer Apprentice System at MIT [11].

Incorporation of programming knowledge in the form of program transformation rules leads to program transformation systems for implementation of programs [8]. The selection of transformation rule to be applied is generally done in interactive fashion to some extent. It is assumed that program specification is given in some program specification language.

What follows is a rather terse summary of excellent expositions in [7,8,10,11], the reader is referred to those articles for details. In the next section, we consider the acquisition of programming knowledge needed to support automatic program implementation. Section 3 provides a view of knowledge based programming environments. The last section provides a summary of the paper.

2. PROGRAMMING KNOWLEDGE

Knowledge of programming used in construction of a program is essential to transform a program specification into a program. Acquisition and codification of this knowledge as a set of rules for program synthesis is necessary to support automatic program implementation. This knowledge may be obtained by discovering programming paradigms used in construction of a program. Based on [7], we present knowledge that can be used to construct a sort program based on a class of sorting algorithms.

A unified view of a class of sorting algorithms leads to knowledge that can be used in synthesis of sorting algorithms.

390

It is based on the higher level transfer paradigm, recursive paradigm, divide-and-conquer paradigm which are also applicable outside the sorting context. A set S of elements are to be ordered according to some specified ordering relation to obtains the ordered set S'. The basic divide-and-conquer strategy involves splitting of the S into two smaller sets S1 and S2, sorting S1 and S2, and then joining them in a manner to generate S'. Thus,

$$SORT(S) \leftarrow JOIN(SORT(SPLIT_1(S)), SORT(SPLIT_2(S)))$$

A recursive approach based on this divide-and-conquer strategy is now evident. The proof of correctness of the algorithm is based on the fact that a set with single element is sorted, and requirement that JOIN of a two sorted subsets be sorted.

Depending on whether the split is singleton split or equal size split, and whether sorting is accomplished by JOIN or SPLIT leads to the following sorting algorithms.

	Singleton split	Equal size split
Work done by JOIN	INSERTION	MERGE
Work done by SPLIT	SELECTION	QUICK

An effective implementation of a sorting algorithm will require transformation of recursive to iterative transfer paradigm. In addition, additional implementation issues such as in-place sort etc. leads to a large number of programming rules for implementation of this class of sorting algorithms. These rules then form the basis for a system for automatic implementation of programs.

3. KNOWLEDGE BASED PROGRAMMING ENVIRONMENTS

The programming environments considered in this section differ in the type of knowledge it incorporates and the degree of automation it supports.

3.1 Syntax Directed Programming Environment

The knowledge about of the syntax of programming language is exploited in the syntax directed programming environments [10]. It provides for integrated support for program editing, execution, and debugging. The grammar of the programming language is embodied in a collection of templates predefined for all but the simplest statement types. Templates reinforce the view that as program is a hierarchical composition of syntactic objects, rather than a sequence of characters. Templates are generated by command, but expressions and assignment statements are inserted on per character basis.

391

Each time a template or a phrase is inserted, the code is generated. This allows the execution to follow editing without delay. Thus, incomplete programs are executable and hence program development and testing can go hand in hand. Since the templates are predefined, there is no possibility of error in templates. The user typed text is checked immediately since the parser is invoked by the editor on a phrase-by-phrase basis.

The templates correspond to abstract computational units; because they are both inserted and manipulated as units, the process of programming begins and continues at a high level of abstraction; the user is never mired in the syntactic detail. At runtime, templates provide a framework for the structured, single-step debugging facility. Thus templates, provide a unified view of both static and dynamic program structure.

3.2 Programmer's Apprentice

The programmer's apprentice project at MIT is involved in development of programming assistant system. The PA system assists the programmer in the program development. It allows the programmer to build up a program from prototypical fragments, and to modify a program in terms of its logical structure. The programmer does not have to work through the PA system. The PA when used serves truely as a assistant. In order to be helpful, the system must understand the programming process. The basis of this understanding is a representation for a program, called a plan, and knowledge about programming.

A plan contains data flow as well control flow. The basic unit of a plan is a segment which corresponds to a unit of computation. Each segment has a number of input ports and a number of output ports which specify the input values it receives and output values it produces. A set of specifications associated with each segment gives pre-conditions which must be true for the segment to execute properly, and post-conditions which hold true after the segment is executed.

The plan library is a collection of plans for common algorithmic fragments. These correspond to expressions, complex predicates, conditional expressions, and loops. In addition, more common programming cliche such as successive approximation, equality within epsilon, average, counting up, etc.. Each fragment has a name, and each has named roles which may be unfilled. The plan library is different from conventional program library in that it is independent of programming language, and a plan may be realized as code in many different ways.

The system consists of the analyzer - a program which constructs plan from a given program text, the coder - creates program from a given plan, the drawer - provides a graphical representation of a plan, the plan library - contains program fragments represented as plans, and the plan editor - allows modification of a program by modifying its plan.

3.3 Program Transformation Systems

An excellent overview of program transformation system is given in [8]. In the transformation approach to automatic programming development, it is assumed that the program specification is given in some suitable specification language. Hence the program specification itself may be viewed as a program in the specification language. A transformation T applied to a program P results in a new program P'. Program development is then viewed as application of appropriate set of transformations to the program specification. We note that P' may be a program in the same language as P, as would be the case for an optimizing transformation. And a transformation must preserve correctness in that it must faithfuly implement P.

Transformation systems typically use a catalogue of transformations. The selection of transformation may be automatic or may be made in interactive fashion. Transformation systems are used for program modification, program synthesis, program adaptation, or program description.

The transformation rules may be local and relate to language constructs, algebraic properties of these constructs, or application domain. Or these rules may be global and relate to flow analysis, consistency checks, or representation of programming paradigms. There are also rules that codify programming knowledge such as FOLD, UNFOLD rules, which are neither global nor local but represent implementation detail.

4. CONCLUSION

The goal of automatic programming still remains a distant one. General purpose systems provide flexibility while systems with restricted domain of applicability can be more powerful. Judicious incorporation of programming knowledge into programming environments can significantly enhance the capability for the development of error free programs and improve the programmer productivity.

REFERENCES

[1] R. Balzer, "A 15 year perspective on automatic programming," IEEE TSE, vol.SE-11, no.11, pp.1257-1267, November 1985.

[2] J. M. Boyle and M. N. Muralidharan, "Program Reusability through program transformation," IEEE TSE, vol.SE-10, no.5, 1981. pp.574-588, September 1984.

[3] J. Darlington, "An experimental program transformation and synthesis system," Artificial Intelligence, vol.16, pp.1-46,

[4] S. F. Fickas, "Automating the transformational development of software," IEEE TSE, vol.SE-11, no.11, pp.1268-1277, November 1985

[5] R. W. Floyd, "The paradigms of programming,", Comm. of ACM, vol.22, no.8, pp.455-460, August 1979.

[6] A. T. Goldberg, "Knowledge-based programming: A survey of program program design and construction techniques," IEEE TSE, vol.SE-12, no.7, pp.752-768, July 1986.

[7] C. Green and D. Barstow, "On program synthesis knowledge," Artificial Intelligence, vol.10, pp.241-279, 1978.

[8] H. Partsch and R. Steinbruggen, "Program transformation systems ACM Computing Surveys, vol.15, no.3, pp.199-236, September 1983.

[9] C. V. Ramamoorthy, V. Garg, and A. Prakash, "Programming in the large," IEEE TSE, vol.SE-12, no.7, pp.769-783, July 1986.

[10] T. Teitelbaum and T. Reps, "The Cornell program synthesizer: A syntax directed programming environment," Comm. of ACM, vol.14, no.9, pp.563-573, September 1981.

[11] R. C. Waters, "The programmer's apprentice: Knowledge based program editing," IEEE TSE, vol.SE-8, no.1, pp.1-11, January 1982.

Knowledge Acquisition and Rapid Prototyping of an Expert System: Dealing With "Real World" Problems

Patrick A. Bailey and Brett B. Doehr

Martin Marietta Denver Aerospace
Space Station Program
P.O. Box 179 (MS D1744)
Denver, CO 80201

Abstract

This paper addresses the knowledge engineering and rapid prototyping phases of an expert system that does fault handling for a Solid Amine, Water Desorbed (SAWD) CO2 removal assembly for the Environmental Control & Life Support System (ECLSS) for space-based platforms. The knowledge acquisition phase for this project was interesting because it could not follow the "textbook" examples. As a result of this, a variety of methods were used during the knowledge acquisition task. The use of rapid prototyping and the need for a flexible prototype suggested certain types of knowledge representation. By combining various techniques, a representative subset of faults and a method for handling those faults was achieved. Our experiences should prove useful for developing future fault handling expert systems under similar constraints.

Introduction

This paper describes the knowledge acquisition and rapid prototyping phases for the Atmosphere Revitalization Group Expert System (ARGES). The work was supported by Martin Marietta Denver Aerospace Independent Research and Development task D-47S. The expert system does fault detection and diagnosis for a CO_2 removal assembly within an Environmental Control & Life Support System (ECLSS) [17]. The project is interesting because we were not able to follow the usual knowledge acquistion requirements of dealing with a well understood knowledge domain and having experts in the domain readily available. Although this caused some problems, we were able to develop an expert system while gaining valuable experience in the knowledge acquisition and rapid prototyping processes. This paper explains why we were not able to follow the usual requirements, describes the knowledge acquisition and rapid prototyping process we did use, gives a summary of the lessons we learned, offers suggestions for similar projects, and discusses future directions for this work.

"Real-World" Problems

Two commonly agreed upon prerequisites for expert system development are that the knowledge domain be well understood and that the experts in the domain and the knowledge engineers work closely together [2, 4]. We were not able to strictly abide by either of these for the reasons described below.

1. The CO_2 removal assembly is a Solid Amine, Water Desorbed (SAWD) assembly developed by Hamilton Standard, Inc. The prototype was being developed by Hamilton Standard during the knowledge acquisition phase, resulting in several changes to the assembly design details during that phase.

2. The prototype we were using for the expert system, SAWD II, was being developed from an earlier version, SAWD I. Although there were experts on the operation of SAWD I and the theoretical operation of SAWD II, there were no experts on fault detection and diagnosis for SAWD II. We had to deduce most of this knowledge with the help of the experts.

3. The Hamilton Standard ECLSS experts, located in Windsor Locks, Connecticut, were busy developing SAWD II and the knowledge engineers were in Denver, Colorodo. Thus, the experts were not readily available to the knowledge engineers.

4. The operation of SAWD II would affect and be affected by its interaction with other ECLSS assemblies. We

needed to include some of these effects in the expert system but the interactions were not fully understood.

5. The expert system would have to process telemetry data from SAWD II. However, because SAWD II had not been built and tested, we didn't have any samples of that data. Thus, we didn't know what would be "typical" deviations for each SAWD II parameter during normal operation (e.g. the size and frequency of fluctuations, etc).

The following sections of the paper discuss the methods developed for dealing with these problems.

The Knowledge Acquisition and Rapid Prototyping Processes

This section describes the knowledge acquisition and rapid prototyping processes by giving a chronological summary of what was done and why it was done. The summary describes how we acquired our knowledge of SAWD II, how we chose an architecture for the prototype, and how it was revised based on feedback from the experts at Hamilton Standard.

Domain Selection

The initial intent of this project was to develop an expert system within ECLSS, with the Atmosphere Revitalization Group (ARG) identified as having a likely set of candidates. From this point, prior to our initial meeting with Hamilton Standard, we examined the literature available on various assemblies (CO_2 removal, CO_2 reduction and O_2 generation) within the ARG [3, 6, 11, 18]. This gave us initial insight about the domain candidates, making certain that the questions we then asked the experts were knowledgeable. Because of the effort and expense involved in setting up meetings between such widely separated groups, it was important to maximize the time that was available. In addition, we read about other expert systems done in the ECLSS area [10, 15] to identify some of their benefits and limitations, with the intention of being able to overcome the latter while retaining the former.

At that first meeting, we reviewed the ARG assemblies as well as an assembly for waste water reclamation. The project managers for each assembly gave a presentation on it, after which we discussed what might be an appropriate application for an expert system dealing with that assembly. Because of the relative complexity of their SAWD II CO_2 removal assembly, and after examining the capabilities of the assembly controller, we determined that we could enhance the fault handling capability for that assembly with the use of an expert system.

Initial Knowledge Acquisition

We acquired some additional information from Hamilton Standard specific to the SAWD II assembly, again preparing as much as possible ahead of time for the next face-to-face discussion. We then held a meeting with Hamilton Standard and were given a description of how the SAWD II assembly would operate based on its current design. At that meeting we requested some sample SAWD II faults and the effects (symptoms) of those faults on SAWD II. They gave us a summary of the effects of seven major faults and a Failure Mode and Effects Analysis (FMEA).

Various papers had given us a good idea of how SAWD I worked [3, 11] and how SAWD II was meant to operate [6], but SAWD II was to be sufficiently different from SAWD I that we couldn't rely on our understanding being exactly correct. The FMEA wasn't of much use as far as specific information because as hardware development proceeded FMEAs would quickly become outdated. However, it and the summary of seven faults did give us an idea of the type of reasoning the expert system would have to do; that is, it made us realize that the expert system could deduce faults from SAWD II sensor information using if-then type rules. This is something we had already realized from our knowledge of fault handling and the papers we read about fault handling expert systems [10, 12, 13, 16], but it was nice to see it confirmed. We had also read some papers on causal model reasoning and considered using it in the expert system. So, at this point we had two general ideas for the knowledge representation and the reasoning process for the expert system. The first was a rule-based approach in which if-then type rules would reason by forward-chaining from sensor information to symptoms and finally to faults. This is a fairly simple paradigm that is well understood and has proven useful. Rules infer symptoms from SAWD II sensor information and other rules infer faults from patterns of symptoms. We also thought it would be possible to make use of backward-chaining or

goal-directed reasoning. That is, rules would consider a fault as a possibility, reason backwards to the symptoms necessary for that fault to exist and then look for sensor information that would be necessary for the symptoms to exist. The second idea was a causal model approach in which a qualitative or quantitative model of SAWD II would be used. Although this approach is still an area of research, it has been used with success [1, 12, 14].

The Prototype Architecture

We decided to use a forward-chaining rule-based approach for the prototype. The reasons for this choice are as follows:

1. We understood it best and it seemed to fit the type of knowledge we had at that point ("if-then" rules are used to represent a wide variety of knowledge, probably because people can easily express their knowledge in that format). The FMEA and the fault summary indicated that the SAWD II fault handling knowledge could be formalized as essentially independent modules of knowledge each consisting of a fault and a set of symptoms that indicate the fault. Rules are good for representing this type of knowledge because they are modular and independent [8]. We chose forward-chaining [5] because all the available information would be present in ARGES; that information would be sent from SAWD II and other assemblies to ARGES via a data network.

2. SAWD II was going to undergo many changes before the final version so that any expert system we developed would have to be easily modifiable and expandable. A rule-based system would be much easier to modify and expand than a model-based system [8].

3. We wanted to get a prototype running quickly as a "proof of concept" and the rule-based approach would be the quickest of the two.

4. We could implement the model based reasoning later as an extension to the rule-based reasoning. This seems to make sense when a human expert's problem solving technique is considered. When faced with a problem to solve, humans usually attempt to use heuristics (rules of thumb or surface knowledge) first. These heuristics are rules they have learned from experience. When those rules fail, the human relies on their model of the situation (i.e., deep knowledge). The latter is usually a more complicated process taking more time.

Initial FMEA and Knowledge Base

After deciding on the rule-based approach for the prototype, we chose 12 typical SAWD II faults and wrote a document that contained those faults and the symptoms we thought would occur as a result of each fault. We deduced the symptoms from our mental models of how SAWD II should work. This is not the normal way that knowledge engineering is done; the experts usually have this knowledge and the knowledge acquisition process involves formalizing that knowledge. Because of the peculiarities of our situation we had to deduce the knowledge and submit it to the experts for review and confirmation.

After the first iteration of this FMEA we developed a prototype knowledge base and an experimental explanation facility for the faults in the FMEA. The knowledge base consists of rules that build a dependency network of the chain of reasoning used to diagnose a fault. This is a network of nodes, each node being connected to all the nodes on which it depends. The explanation facility we developed is a Lisp procedure that traverses the network backwards from the fault to each antecedent node, printing out text that explains the step in the reasoning represented by the node. The explanation was designed so the operator could choose the desired level of explanation; that is, the extent of the chain of inference to be explained.

We experimented with another method of doing the explanation. Instead of using a network, this method used rules to chain backwards from the fault to the antecedents; it was essentially the knowledge base in reverse. We used the network approach because it was more run-time efficient (though less space efficient) than the rule approach and we were more concerned with time efficiency than space efficiency. Also, the space efficiency advantage of the rule method was not significant because some of the information held in the network nodes would have to be held in memory by the rule method. Thus, the rule approach would require at least a partial network. Also, the ARGES explanation process was a procedural process and such things are better done in a procedural language rather than a rule-based language.

To set up the rule base we drew a map of the network the FMEA would build if it was compiled directly into rules. This was actually twelve maps; one for each fault. The first map we drew had two levels: sensor data recognized as symptoms by the left-hand sides of the rules and the faults asserted by the right-hand sides. Thus, some fault network nodes were connected to as many as seven symptom nodes. We decided the network would be easier to understand, when explained, if we could summarize some of the symptoms into a higher level description of the symptom. We called these intermediate symptoms.

As an example of an intermediate symptom, consider the effect of SAWD II on a CO_2 reduction assembly, the next process after SAWD II. The Bosch is an example: it combines H_2 and the CO_2 collected by SAWD II and reduces them to H_2O and Carbon. If a fault occurs in SAWD II which allows excess O_2 to be sent to the Bosch then two symptoms occur: the Bosch rate of expulsion of excess gas will increase and the Bosch rate of H_2 usage will decrease (other faults can cause one or the other of these symptoms to occur alone). We summarized these two symptoms as the intermediate symptom "the CO_2 reduction flow is impure".

The knowledge base makes inferences on three levels as follows:

1. If certain sensor readings are present then infer a particular symptom.

2. If certain symptoms are present then infer a particular intermediate symptom.

3. If certain symptoms and/or intermediate symptoms are present then infer a particular fault.

We could have done all the reasoning necessary by using only level 1 and 3 but the intermediate symptoms acted as a sort of cognitive summary; that is they grouped together symptoms that were related. This helps organize the explanation in a way that is easier to understand.

The First Review By Experts

After completing this first FMEA we held another meeting with the experts at Hamilton Standard. This meeting was interesting and informative for us because for the most part it was a discussion between two SAWD experts about our FMEA. They were trying to decide if each symptom set was valid for the particular fault. This was a fault-derived approach. To do this they made use of their knowledge of how SAWD I worked and their ideas of how SAWD II should work, so we learned a lot more about SAWD operation. During the discussions one of the experts asked if we shouldn't be using a symptom-derived FMEA; starting with a set of symptoms and deducing all the faults that could cause each symptom. We decided that a symptom-derived FMEA might miss some symptoms for a particular fault and might not be as efficient at generating a representative set of faults, which was our goal at that point. Therefore, we stayed with the fault-derived FMEA.

In addition to this meeting as a method of acquiring knowledge, we also had a Hamilton Standard computer scientist working with us in Denver for about three months. She had written most of the SAWD II controller software so she was able to answer questions about the controller operation and capabilities. Even though she was not an expert at SAWD II fault handling, she was helpful in our understanding of how SAWD II worked and in deducing the symptoms that would exist for each fault. However, the review by the SAWD experts was still essential.

As a result of these discussions we realized that most of the faults would be due to the slow drifting of sensors or the accumulated effects of the gradual failure of some other component. We grouped both of these fault sets together and called them drift faults. The faults that happened quickly (which we called step faults) would be detected by the SAWD II controller when the fault caused a parameter to reach a critical value. The drift faults would eventually cause some SAWD II parameter to reach a critical value and the SAWD II controller would shut the system down. So, ARGES would have to do the following:

1. Monitor the SAWD II data.

2. Diagnose the fault when a step fault caused SAWD II shutdown.

3. Trend the data and watch for drift faults.

4. Diagnose the drift faults.

We talked to several people about how we should do the trending in ARGES and had almost as many opinions as the number of people we talked to. It was difficult to know the correct mathematical method needed because we didn't know the form of the data, we were not statisticians and we had no data from SAWD II (it hadn't even been built at that time) so we had no good idea of the variations in the data the expert system would be getting (size and frequency of fluctuations, etc.). We just had some vague guidelines such as knowing that the size of the SAWD II environment would affect the form of the data by dampening fluctuations in the carbon dioxide level, but we didn't know how great the effect would be. We finally decided on doing a linear least-squares fit of the data points and then just comparing the slopes of the resulting fits.

Intermediate Knowledge Acquisition

This included another meeting between the knowledge engineers and the experts, as well as several rounds of questions and answers carried out via telefax and telephone calls. This process clearly identified some of the limitations of performing knowledge acquisition "remotely" as opposed to in actual meetings. When transmitting via telefax or using the mail to send questions and feedback, there is the delay in writing up the questions/information, the concern about trying to eliminate any ambiguity, and the problem that some ambiguity invariably remains. By not having "real-time" interaction, effort can be misdirected and serious delays result. Telephone calls, while providing real-time interaction, can involve difficulties in actually getting a hold of a person (the dreaded game of "phone tag"). In addition, it restricts one to the verbal medium for getting ideas across, which can often be limiting. Also, it provides scant time for mulling over a question or answer, resulting in the frustration of remembering questions or details five minutes or two hours after the conversation, which requires the process to begin anew.

Despite these difficulties, we were able to continually update our knowledge base to incorporate the current knowledge and philosophy of both normal and faulty SAWD II operation. This process demonstrated the necessity of using a flexible and easily modifiable representation of knowledge during the rapid prototyping phase. It also showed the utility of representing as much knowledge as possible, even when that knowledge is "best guess", so that the general amount and format of knowledge is known. It is easier to modify the specific value of a piece of knowledge once it becomes known, in the meantime using the best guess for preliminary testing and debugging, than to later on attempt to integrate a new and unforseen piece of knowledge which may alter the reasoning process and thus require major restructuring of the knowledge base.

Lessons Learned and Suggestions For Expert System Development

1. As a result of our knowledge acquisition experience, we conclude that the most effective and efficient method of knowledge acquisition that we tried was meeting with several experts at one time. This is especially true in a situation like ours because SAWD II was not finished, and having several people present should increase the chances of getting correct information (e.g., often one person will think of something that others miss). Getting feedback from the experts via telefax and/or mail suffices for minor feedback but we do not recommend it for the bulk of the interaction since it is possible that valuable time can be wasted waiting for responses to be written up and sent.

We suggest that there be time in between meetings so both sides are able to digest information and think; knowledge engineering can be an intense process and it is crucial that a correct understanding be gained by the knowledge engineers. When the experts are located a long distance from the knowledge engineers, the meetings should probably go for several days with at least the evenings free. Free time will allow each person to think privately about what has occurred in the meetings. This can get to be expensive but we think it will prove more cost effective in the long run because it will allow the problem to be more thoroughly thought out in the beginning so that many mistakes can be avoided. When the experts are located close to the knowledge engineers, we suggest several meetings with day-long breaks in between.

We also suggest that one knowledge engineer travels to the site a day or so ahead of the main meetings to brief the experts about what the knowledge engineers need or send a written briefing ahead of time (and make sure the experts read it). This shouldn't be done too far in advance because the experts are busy with their own work and will easily forget what they were told. This pre-meeting briefing will allow the experts to understand what is

needed from them and to formulate an approach to take in presenting their knowledge.

2. Make sure the people you are talking to are the real experts and talk to them directly. If the real experts are not available from the start of the knowledge acquisition process, significant changes may have to be made later. If you have to communicate with the experts through layers of management, miscommunication and time delays will most likely take their tolls.

3. In our situation, when the formalized expert knowledge (the FMEA) was reviewed by the experts, they not only changed the knowledge but also came up with new ways that the knowledge should be used (trend analysis of the data and the control of the reasoning process). This happened after the first iteration of the prototype and again after the second iteration. Because we chose a flexible rule-based approach to code the knowledge, neither change caused a major rewrite of the prototype. The lesson is *keep the prototype flexible*. The rule-based approach is a good way to do this.

4. It greatly facilitates things to find out as soon as possible at least the type and format of all the information that will be available to the expert system. This helps the reasoning process to plan for that information and also allows the knowledge base to incorporate "slots" for that knowledge in its structure. It also is helpful to get a best guess for any information still unknown, this enables reasonable testing and debugging to be done in early iterations of the expert system, with the ability to plug in "real" values as they become known.

5. There were at least two parts of the reasoning process that we didn't know for sure how to do correctly: the trending and the comparisons of the trend slopes. We thought about these things but were pressed for time so we implemented temporary solutions and deferred the real solutions until later. This allowed us to get a system up and running to evaluate the concept. Don't be afraid to defer parts of the problem.

6. We used a language developed internally at Martin Marietta for the rule base. It is called HAPS (Hierarchical, Augmentable Production System) and is an OPS-like language. We used it because we had access to the source code, it seemed to fit the problem as we saw it for our prototype and we didn't see any sense in spending a large amount of money on another language or expert system shell until we had a better idea of the final expert system architecture (e.g. would non-monotonic reasoning be needed, would the knowledge be expressed mainly in a rule base or would it be mainly model based, etc). The expert system shells available may be general enough to solve many problems but if a shell has a lot of features that a particular application does not need, then it is carrying a lot of overhead and cost. We think it is best not to make a premature choice of a language or shell. There are many public domain or inexpensive versions of OPS 5 that can be used in the same way as we used HAPS, i.e. for the prototype. When the problem is well understood, you can choose a language that better fits the problem.

7. We used Symbolics-Lisp for the control part of the expert system and the user interface. The program development was done on Symbolics 3670s and LMI Lambdas. These are computers dedicated to the Lisp language and have excellent AI development environments. We would highly recommend using these or similar machines if circumstances permit.

8. The architecture we used for the expert system may be useful for the prototype development of other fault handling expert sytems. It is a simple, flexible design that will allow the knowledge engineers to better understand the problem they are working with and allow them to have a working version done quickly to show the experts.

9. We spoke earlier of the concurrent development of the expert system and SAWD II as a problem. It is a problem when viewed from the viewpoint of "traditional" expert system development. However, concurrent development can be an advantage because the expert system development could influence the application design and development. In the future, such developments should probably be done as one.

10. The expert knowledge required for an expert system often contains proprietary information [7]. If the experts and the knowledge engineers are from separate companies, this could be a fatal problem. The issue should be discussed and a solution to the problem agreed on before knowledge engineering begins. Also, the decision should be written into a contract between the two companies.

Current Status and Future Directions

The control and the drift fault rule base have been written and partially tested. The final drift fault rule base has the same basic form as the prototype but the knowledge contained in the rules has changed and it reasons from trended data unlike the prototype (it reasoned from untrended data). A revised version of the same explanation is being used which incorporates the same basic idea as the original explanation but has a greatly improved text display.

When ARGES becomes more sophisticated it might have to handle faults requiring operator input; then we might need to use backward-chaining [5]. This might be the case in which the forward-chaining rules have failed so ARGES has to hypothesize possible faults based on the information it has and then backward chain to determine other information it might need in order to verify a possible fault or choose one of several. It would then query the operator for the information.

As mentioned earlier, another diagnosis technique that could be used when the forward-chaining rules fail is model-based reasoning. We may even be able to integrate rule- and model-based reasoning in a more effective way than this. Recent work in this field can be found in the references [1, 12, 14].

Our current method of trending is to do a linear least-squares fit of data points and comparing the slopes of the fits. We aren't convinced of the correctness of that method. However, we now have a contact at Carnegie Mellon University who has quite a bit of experience doing this sort of trend analysis and who has recommended a more correct method.

We plan to have a second rule base that diagnoses the step faults. ARGES would then watch for a message sent from the SAWD II controller that would indicate a step fault had occurred, put SAWD II data in working memory and invoke the step fault rule base. This will be very much like the drift fault rule base except it will recognize symptoms by comparing sensor values to the expected values; the drift fault rule base recognizes symptoms by comparing sensor trends. Also, the step fault rule base may have to be able to reason using partial data since SAWD II may be shut down before a full set of data is received.

If our drift fault rule base grows large, the time needed to do a linear least-squares fit of the data could become significant. In that case, we plan to modify the expert system control so that it narrows down the set of possible faults before trend analysis is done. This might be done by looking for symptoms that show up quickly but do not allow isolation of the fault.

REFERENCES

1. Bobrow, Daniel G., Editor, Qualitative Reasoning about Physical Systems. This book contains several excellent reprints. The most applicable are: "Diagnostic Reasoning Based on Structure and Behavior" by R. Davis and "The Use of Design Descriptions in Automated Diagnosis" by Michael R. Genesereth.

2. Bobrow, Daniel G., Mittal, Sanjay and Stefik, Mark J., "Expert Systems: Perils and Promise", Communications of the ACM, September 1986 Volume 29 Number 9 ppg 880 - 894.

3. Boehm, Albert M. and Cusick, Robert J., "A Regenerable Solid Amine CO_2 Concentrator for Space Station", SAE (Society of Automotive Engineers) Technical Paper Series, Number 820847.

4. Buchanan, B., Barstow, D., Bechtal, R., Bennett, J., Clancey, W., Kulikowsky, C., Mitchell, T., and Waterman, D. A., "Constructing an Expert System" Building Expert Systems, Hayes-Roth, F., Waterman, D. A. and Lenat, D. B., Eds. Addison-Wesley, Reading, Mass, 1983.

5. Buchanan, Bruce G. and Shortliffe, Edward H., Rule-based Expert Systems, pp. 3 - 6, Addison-Wesley.

6. Colling, Jr., Arthur K., Nalette, Timothy A., Cusick, Robert J., and Reysa, Richard P. "Development Status of Regenerable Solid Amine CO_2 Control Systems", SAE (Society of Automotive Engineers) Technical Paper Series, number 851340.

7. Cook, T. M., "AI Applications for the Space Station Program: Technology, Design and Integration Challenges", presented at the conference, AIAA Space Station in the Twenty-first Century, September 3 - 5, 1986 in Reno, Nevada.

8. Davis, R. and King, Jonathan, "The Origin of Rule-based Systems in AI", in Rule-based Expert Systems, Buchanan, Bruce G. and Shortliffe, Edward H., pp. 20 - 55, Addison-Wesley.

9. Dickey, Frederick J. and Toussaint, Amy L., "ECESIS: An Application of Expert Systems to Manned Space Stations." IEEE, 1984.

10. Doehr, Brett and Walsh, Rick, "MOSES: A Demonstration System for Handling Multiple Faults in Satellite Propulsion Subsystems", Proceedings from Artificial Intelligence - From Outer Space ... Down to Earth, October 1985, University of Alabama in Huntsville.

11. Dresser, Kennith J. and Cusick, Robert J., "Development of Solid Amine CO_2 Control Systems for Extended Duration Missions", SAE (Society of Automotive Engineers) Technical Paper Series, number 840937.

12. Fink, Pamela K., "Control and Integration of Diverse Knowledge in a Diagnostic Expert System", Proceedings of the Ninth International Joint Conference on Artificial Intelligence, (IJCAI '85), pp. 426-431.

13. Friedman, Leonard, "System for Automated Troubleshooting", internal publication at NASA's Jet Propulsion Laboratory.

14. Kuipers, Benjamin and Kassirer, Jerome, "Causal Reasoning in Medical Analysis of a Protocol", Cognitive Science 8, 1984, pp. 363 - 385.

15. Lance, Nick and Malin, Jane T.: "Development of a Prototype Expert System For Fault Diagnosis of a Regenerative Electrochemical CO_2 Removal Subsystem." Reference no. EC3/SR111, NASA/Johnson Space Center, Houston, Texas, May 13, 1985.

16. Maxion, Roy A. and Morgan, David E., "The Application of Artificial Intelligence Techniques to Reliable and Fault-tolerant Computing", IEEE, 1984 pp. 285 - 290.

17. Mendler, Andrew P., Pachura, David W. and Suleiman, Salem A. O., "ARGES: An Expert Sysytem for Fault Diagnosis within Space-Based ECLS Systems", to be presented at the Conference on Artificial Intelligence for SpaceApplications, November 1986, University of Alabama in Huntsville.

18. Spina, L., and Lee, M. C. "Comparison of CO_2 Reduction Process - Bosch and Sabatier", SAE (Society of Automotive Engineers) Technical Paper Series, number 851343.

REASONING ABOUT FAULT DIAGNOSIS FOR THE SPACE STATION
COMMON MODULE THERMAL CONTROL SYSTEM

G. Vachtsevanos and H. Hexmoor
School of Electrical Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332-0250

B. Purves
Boeing Aerospace Company
Space Station Program
Huntsville, Alabama 35801

ABSTRACT

The proposed common module thermal control system for the space station is designed to integrate thermal distribution and thermal control functions in order to transport heat and provide environmental temperature control through the common module. When the thermal system is operating in an off-normal state, due to component faults, an intelligent controller is called upon to diagnose the fault type, identify the fault location and determine the appropriate control action required to isolate the faulty component. This paper is introducing a methodology for fault diagnosis based upon a combination of signal redundancy techniques and fuzzy logic. An expert system utilizes parity space representation and analytic redundancy to derive fault symptoms the aggregate of which is assessed by a multivalued rule-based system. A subscale laboratory model of the thermal control system designed by Boeing Aerospace Company is used as the testbed for the study.

INTRODUCTION

The common module Thermal Control System (TCS) for the space station is designed to integrate thermal distribution and thermal control functions in order to transport heat and provide environmental temperature control through the common module.

A classical hierarchical controller has been designed to direct the fluid flow so as to remove excess heat and maintain an equilibrium such that the temperatures at certain specified points in the network take on prescribed values. To achieve this objective, the available control inputs are the overall fluid mass flow rate and the relative closures of certain bypass valves. The available outputs consist of flow rates and temperatures measured at several points in the network.

In addition to the optimum control strategy, algorithmic approaches have been developed, both conventional and AI-based, to diagnose select fault conditions in the TCS, determine best estimates of monitored process variable data, identify the type of fault and its location in the thermal loop, and provide appropriate status reports about means for isolating faulty components to the maintenance module with minimum disruption to the "normal" TCS functions.

In the operation of the TCS, interpreting and reporting to a large quantity of real time data is a challenging task, particularly under off-normal conditions. The timely detection of fault conditions and the isolation of faulty components is a task of paramount importance if the operational integrity of the system is to be maintained.

The expert system FDIC (Fault Diagnostics and Intelligent Control) has been developed to assist in determining the type of a fault, its precise location in the TCS configuration, and suggest means for isolating faulty components.

The Intelligent Controller (IC) interacts and works harmoniously with the hierarchical controller to maintain system integrity.

The goals of the IC design are to develop a fault detection and isolation scheme which

● Maximizes the sensitivity to component failure detection, and

● Minimizes the rate of failure detection-false alarms.

Usually these two design goals involve conflicting criteria and the IC design is called upon to optimize the trade-off.

Current practices in the area of sensor signal validation, fault detection, and isolation are limited to a few rather rudimentary techniques that rely, for the most part, on process symmetry characteristics and physical or analytic redundancy of observables. The proposed methodology incorporates novel analytical approaches and intelligent algorithms which guarantee its sensitivity to subtle failure modes that are transparent to most current techniques, offer, through analytic redundancy, a cost-effective alternative to additional sensor hardware, allow for detection of nonsensor components, provide acceptable fault isolation resolution, and assure robustness to false alarming.

Basic Methodology

Figure 1 depicts, in block-diagrammatic form, the overall intelligent control philosophy. The system or thermal control process is viewed to consist of the TCS and the hierarchical controller with their combined objective directed towards achieving specified control goals under normal operating conditions. Temperature, flow, and pressure sensors provide, through the data management system, status information to the IC for all important process variables. Controller command signals are also provided to the IC directly from the hierarchical controller. On the basis of available information, the IC calls upon a triggering module to initiate the data validation/fault detection routines; upon identification of a fault, it produces a status report detailing the fault type, its location, and such pertinent information as the degree of severity of the particular failure incident; it finally decides as to what appropriate measures must be taken to isolate effectively the fault component.

404

Figure 2 is a schematic representation of a subscale thermal control system (STCS) specially designed and constructed by Boeing Aerospace Co. to demonstrate proof-of-concept feasibility. Three pipe segments and four valves were identified as "components" that may be subjected to a failure mode--a leaky pipe segment and a sticking valve. They were chosen to represent typical piping and valving configurations in the STCS network. The detection methodology may be easily extended to include any number of components as failure candidates as long as a sufficient amount of either direct or analytic measurements, is available at the end nodes of each component.

Basic assumptions concerning the application of the proposed methodology to the STCS may be summarized as: it is assumed that all sensors are performing with zero probability of a faulty indication. Sensor accuracies, as specified by the manufacturers, have been considered in prioritizing and selecting analytical measurements for redundancy purposes, as well as in estimating measurement errors. Component failures are assumed to be of the single-point type and detection to the level of a component can be guaranteed only for the first failure in the system. All pertinent data are taken in the immediate vicinity of a component and, therefore, time delays and data sampling rates are assumed to be negligible. Finally, it is assumed that sensor noise probability density functions are uniform with zero bias errors.

The Intelligent Controller

Figure 3 is a block diagram of the major modules comprising the expert system FDIC.

It is of interest that the Fault Detection and Isolation program be initiated only when some evidence is present which implies the possible failure of a system component. Such triggering of the fault detection routines results in substantial computational savings and improves the reliability of the algorithm by reducing the frequency of the false alarms.

Triggering for a leaky pipe is initiated by two mechanisms: the first one is based upon the fact that controller command signals are presumed to remain unchanged whenever the STCS maintains a steady state status. The second trigger mechanism relies upon the detection of fluid flow through the accumulation (make-up water) tank. Under transient conditions, only the make-up tank scheme may be used as a triggering mechanism.

A motorized valve may remain at a certain position although a command signal is attempting to change its setting. An obvious triggering mechanism for a stuck valve involves initiation of the fault detection routine whenever a change in valve position command is issued. Other triggering mechanisms under consideration refer to identification of off-optimal settings for the loop flow rate whenever the thermal system is faulted.

SEVA (SEnsor VAlidation) uses signal redundancy in a modified version of the parity space algorithm for signal validation developed by the Charles Stark Draper Laboratories [1]. The validation procedure compares each reading to all other like readings. The "best" estimate from a set of good measurements is defined as that value which gives the minimum of a particular

405

function of the measurements. Where more than the minimum number of measurements is available, the "best" estimate is taken in the least squares sense, i.e., it is that value which minimizes the square of the length of the measurement error vector.

The tasks of the sensor validation program are to calculate the analytic measurement values, at each measurement point, which are needed to obtain the "best" estimate of the measured variable. A measurement point is defined to be the beginning or the end node of a leaky pipe segment and the immediate environment of a stuck valve. Furthermore, SEVA determines the normal error bounds for the measured variables and evaluates the final component measurement variables and their corresponding error bounds which enter the fault detection algorithm.

An analytic measurement is expressed as some functional dependence on values of variables, which are obtained from process measurements. The latter are always associated with measurement uncertainties and bias errors. In general, an analytic measurement may be expressed as $f(x_1, x_2, \ldots, x_n)$.

If $\varepsilon_{x_1}, \varepsilon_{x_2}, \ldots, \varepsilon_{x_n}$ are the error bounds corresponding to the direct measurements $x_1, x_2, \ldots, x_n$, then a Taylor's series expansion gives

$$f(x_1 \pm \varepsilon_{x_1}, \ldots, x_n \pm \varepsilon_{x_n})$$

$$= f(x_1, x_2, \ldots, x_n) \pm \varepsilon_{x_1} \frac{\partial f}{\partial x_1} \pm \varepsilon_{x_2} \frac{\partial f}{\partial x_2} \pm \ldots$$

$$\pm \varepsilon_{x_n} \frac{\partial f}{\partial x_n} \pm (0)^n$$

Thus, the analytic measurement error, ε_f, may be expressed as

$$\varepsilon_f = f(x_1 \pm \varepsilon_{x_1}, \ldots, x_n \pm \varepsilon_{x_n}) - f(x_1, x_2, \ldots, x_n)$$

$$= \pm \varepsilon_{x_1} \frac{\partial f}{\partial x_1} \pm \varepsilon_{x_2} \frac{\partial f}{\partial x_2} \pm \ldots \pm \varepsilon_{x_n} \frac{\partial f}{\partial x_n} \tag{1}$$

A conservative analytic estimate of the error threshold may be obtained by performing a standard error propagation calculation under a worst-case scenario. From Eq. (1), consider the worst case:

$$\varepsilon_f = \varepsilon_{x_1} \left| \frac{\partial f}{\partial x_1} \right| + \varepsilon_{x_2} \left| \frac{\partial f}{\partial x_2} \right| + \ldots + \varepsilon_{x_n} \left| \frac{\partial f}{\partial x_4} \right|$$

$$= \sum_{i=1}^{n} \varepsilon_{x_i} \left| \frac{\partial f}{\partial x_i} \right| \tag{2}$$

How small a leak or what minimum change in valve position can be detected is, of course, a function of the instrument error statistics and the error propagation properties when analytic measurements are considered.

Fault detection for a leaky pipe segment or a stuck valve is based upon determining discrepancies or inconsistencies in the measured values (direct or analytic) of certain variables which are most sensitive to the particular fault condition.

Consider first the case of a leaky pipe segment (see Fig. 2). Sensitivity analysis and consideration of instrument inaccuracies dictate that a reasonable measure of the fault condition may be arrived at by estimating the flow rate discrepancies between points a_1 and b_1, i.e., by monitoring the "measurement"

$$\dot{m}_1 = \dot{m}_{a_1} - \dot{m}_{b_1} \tag{3}$$

Similar flow rate measurements are considered for components (2) and (3), i.e.,

$$\dot{m}_2 = \dot{m}_{a_2} - \dot{m}_{b_2} \tag{4}$$

and

$$\dot{m}_3 = \dot{m}_{a_3} - \dot{m}_{b_3} \tag{5}$$

Equations (3), (4), and (5) form the measurement vector $\underline{\dot{m}} = [\dot{m}_1, \dot{m}_2, \dot{m}_3]^T$ that will be employed subsequently in the parity space algorithm.

The measurements $m_{a_1}, m_{b_1}, m_{a_2}, \ldots,$ could be either direct or analytic or a combination of both depending[2] upon the sensor availability at points $a_1, b_1, a_2, \ldots$. When more than one measurement are available at the beginning or end node of a given component, then the "best" estimate is used in Eqs. (3)-(5), in the sense that this estimate minimizes a least squares error criterion.

Next, consider the control valve fault analysis. The bypass control valves β, γ, δ, and σ do not incorporate a positioner and, therefore, a stuck valve cannot be detected by direct measurement of the valve position. In this case, it becomes necessary to consider some indirect quantity in order to express an inconsistency measure useful in the parity space analysis. The flow ratio C is defined as

$$C \equiv \frac{\dot{m}_1}{\dot{m}_0}$$

where $\dot{m}_1$ is the flow rate through the bypass valve and $\dot{m}_0$ is that portion of the flow through the heat exchanger. Then, a measurement for component Vi may be defined by the relation

$$m_{vi} = 1 - \left| \frac{c_i^1 - c_i^2}{c_i^{C1} - c_i^{C2}} \right|$$

where

c_i^1 = actual value of C_i before command is issued

c_i^2 = actual value of C_i after command is issued

c_i^{C1} = command signal before change in command is issued

c_i^{C2} = command signal after change in command is issued.

Thus,

$$m_{vi} = 1 - \left| \frac{\Delta C \ actual}{\Delta C \ command} \right| \tag{6}$$

The corresponding error bounds for each measurement variable are computed by SEVA on the basis of available instrument error statistics.

The fault detection module (FADE) gathers information from the Data Management System, SEVA, and any hierarchical controller setting discrepancies as symptoms for the possible existence of a component failure. Information about the measurement vectors which incorporate measures of flow discrepancies and valve positioning are provided to FADE from SEVA. Subsequently, all of the symptoms are normalized to a unique universe of discourse for diagnostic purposes. Normalized values of the symptoms are fed into a multivalued rule base to determine the type, location, and severity of a fault.

The FADE algorithm is initiated by prioritizing the available evidence for the existence of a faulty component. If the evidence strongly suggests that a possible fault condition is due to a leaky pipe segment, then it initiates the leak component detection part of the algorithm. If, on the other hand, the evidence is suggestive of a sticking valve, then the second part of the algorithm, involving the sticking valve detection, is initiated. Figure 4 depicts a block diagram of the principle functions of the fault detection module.

A major input to FADE is measurement discrepancies, as manifested by the parity vector in parity space [2-10]. The magnitude of the parity vector is a measure of the disparity between the various redundant measurements, and its direction is indicative of the identity of the failed system component. The algorithm proceeds as follows: Following [2], a technique based upon the concurrent checking of the relative consistency of smaller size subsets of measurements is employed to identify a faulty component and, at the same time, estimate the "best" value of the corresponding variable from measurements which remain fairly consistent. The existence and severity of the fault is checked by computing the length of the parity vector and comparing it directly with a measure of the permissible error bounds.

Rule-based algorithms have been developed in order to assess the "health" of a particular component (pipe segment or control valve) in the presence of

symptomatic evidence accumulated from various sources including the parity space algorithm, accumulation tank flow rate data, and hierarchical controller setting discrepancies. The algorithms are based on multivalued logic analysis and the compositional rules of inference.

The algorithms consist of a multivalued rule base containing a number of rules of the type:

IF symptom A_1 and symptom A_2, ..., and symptom A_4,

THEN fault certainty for component 1 is ___, or component 2 is ___, etc.

That is, the IF sides of the rules are information about the fault symptoms and the THEN sides are expert beliefs about the certainty or severity of the faults. Symptom magnitudes are considered in sets of linguistic terms such as HIGH and LOW where each member is given a degree of belongingness to the set. Elements of each set are spanned over the entire range of possible observations. This range is referred to as the universe of discourse. Operating experience and a large volume of STCS input-output data are required in order to define precisely the linguistic membership functions, the range of the universe of discourse, and any other appropriate feature of the intelligent controller for an on-line fault detection application.

Credibility of the diagnostic algorithm is enhanced by using a multiple consecutive disparity criterion. That is, the fault detection scheme is repeated consecutively at least three times before a final status report is issued.

Finally, historical data computed by the fault detection algorithm may be used as a tool for trend analysis and, therefore, for the possible identification of pending component failures or further degradation in component performance.

The fault isolation algorithm (FAIS) is provided with information about the fault type (sticking valve or leaky pipe segment) and the severity of the fault, and it decides as to the action to be taken to isolate the faulty component for repair and maintenance purposes. It issues a status report and appropriate commands to the maintenance module for further action.

Since there is no provision in the status for automatic shut-down or throttle valving, the severity of the fault will dictate the need for operator intervention. Thus, corrective measures can be taken by transmitting the FAIS report to the scheduling and maintenance modules.

CONCLUSIONS

An intelligent controller has been developed for STCS fault detection and identification, and fault isolation based upon a combination of signal redundancy techniques and multivalued logic.

The detailed algorithms provide valuable insight to the STCS control functions. They reveal the precise role and the desirable characteristics of sensors and transducers; they make clear the control functions of conventional means; they suggest ways for improving the selection of sensors and other system components; they guide the topological design of the STCS; they, finally, provide an indispensable tool for integrating normal and emergency control functions of the thermal loop.

The test cases examined illustrate the robustness, viability, and flexibility of the proposed approach.

Future work is required in order to incorporate real STCS test data into the intelligent controller structure, improve and refine the rule-based system, strengthen the interface between the conventional hierarchical controller and the fault diagnostics routines, and achieve a more effective integration of STCS and other subsystem functions.

ACKNOWLEDGEMENT

REFERENCES

1. D.C. Fraser et al., "First Quarterly Progress Report," Contract NAS9-4065, Task Order 41, Charles Stark Draper Lab, MIT, Cambridge, MA, Aug. 30, 1971.

2. M. Desai and A. Ray, "A Fault Detection and Isolation Methodology," Proc. of 20th IEEE Conf. on Decision and Control, San Diego, CA, pp. 1363-1369, Dec. 1981.

3. J.E. Potter and M.C. Suman, "Thresholdless Redundancy Management with Arrays of Skewed Instruments," NATA AGARODograph 224, April 1977.

4. A.S. Willsky, "A Survey of Failure Detection Design Methods in Dynamic Systems," Automatica, Vol. 12, pp. 601-611, 1976.

5. M.N. Desai, J.C. Deckert, and J.J. Deyst, "Dual Sensor Failure Identification Using Analytic Redundancy," AIAA Journal of Guidance and Control, Vol. 2, No. 3, pp. 213-220, May-June 1979.

6. K.C. Daly, E. Grai, and J.V. Harrison, "Generalized Likelihood Test for FD1 in Redundant Sensor Configurations," AIAA Journal of Guidance and Control, Vol. 2, No. 1, pp. 9-17, Jan.-Feb. 1979.

7. A. Ray, M. Desai, and J.J. Deyst, "Sensor Validation in a Nuclear Reactor," Proc. of the 12th Annual Pittsburgh Modeling and Simulation Conference, April-May 1981.

8. C.H. Meijer et al., "On-Line Power Plant Signal Validation Technique Utilizing Parity-Space Representation and Analytic Redundancy," EPRI NP-2110, Project 1541, Final Report, Nov. 1981.

9. P.B. Deegan and A.M. Christie, "An Expert System for Real-Time Diagnostics and Control," ANS International Topical Meeting on Computer Applications for Nuclear Power Plant Operation and Control, Pasco, WA, Sept. 1985.

10. J.E. Potter and M.C. Suman, "Extension of the Midualue Selection Technique for Redundancy Management of Inertial Sensors," Journal of Guidance, Vol. 9, No. 1, Jan.-Feb. 1986.
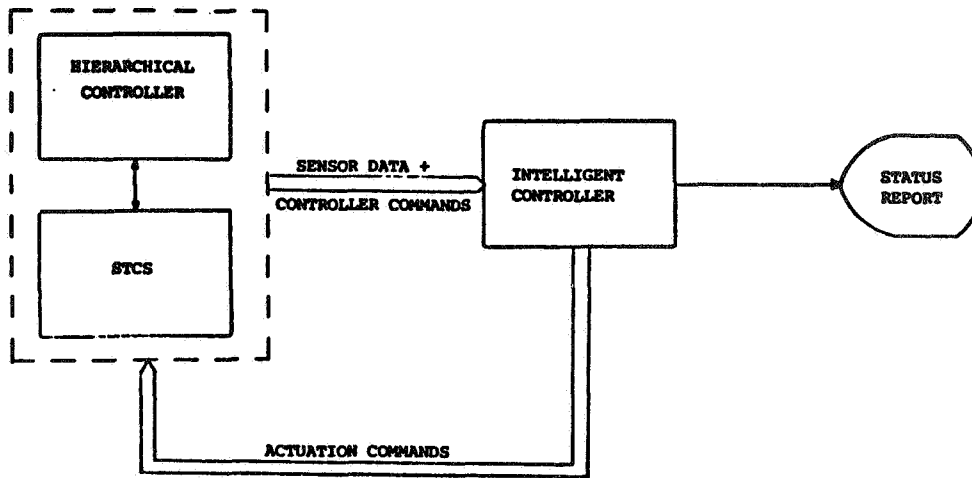
Figure 1. Block Diagram of the Intelligent Controller Interface

Figure 2. STCS Schematic

411
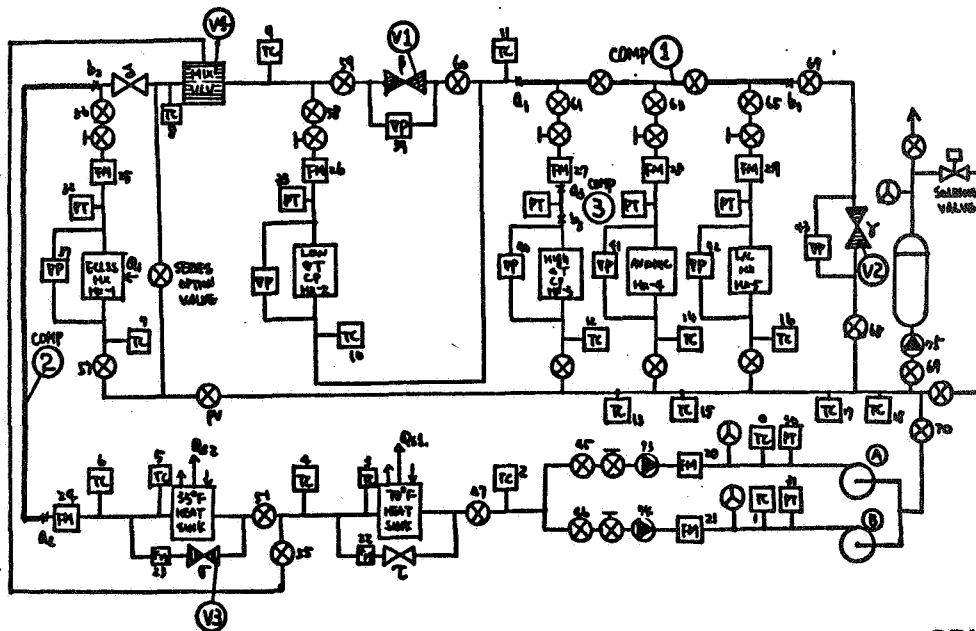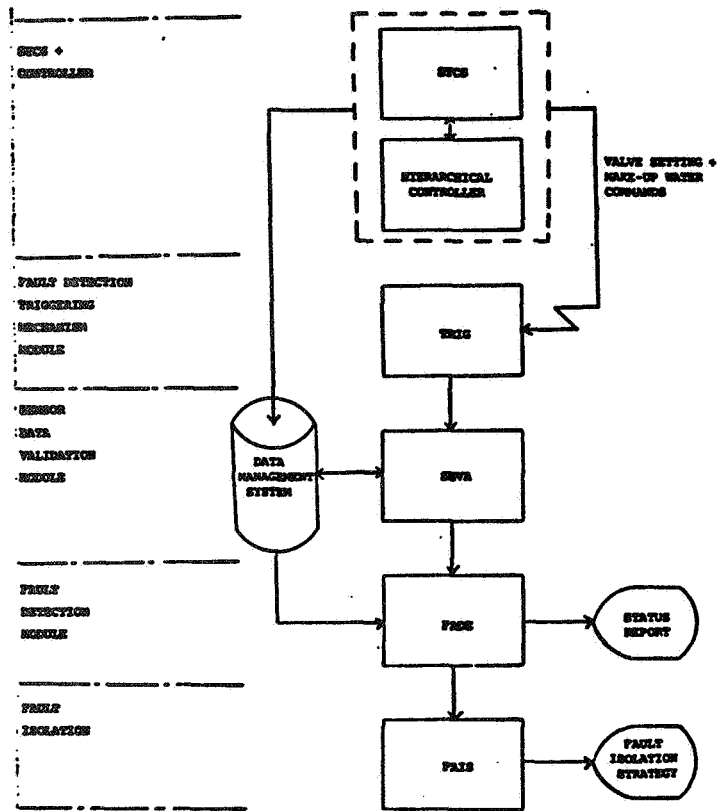
Figure 3. Fault Detection and Isolation Modules
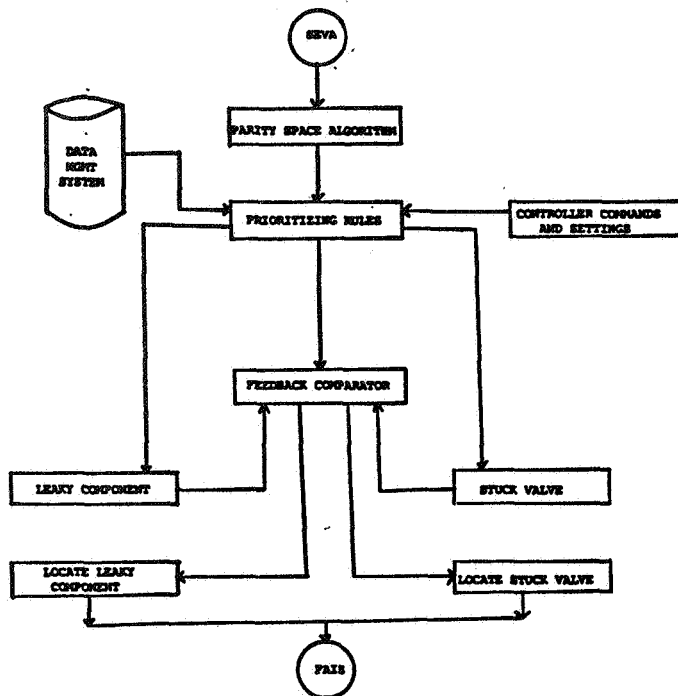
Figure 4. Block Diagram of the Major Fault
Detection Functions

412

Automated Generation of Spacecraft Activity Plans

13 November 1986

Donald A. Rosenthal
Sr. Computer Scientist
Space Telescope Science Institue
Homewood Campus, Baltimore, MD. 21218

Abstract

This paper describes an expert system currently in place in the ground support system of the Hubble Space Telescope (HST). Known as the "Transformation" system, it is used to process proposals submitted by the HST's world-wide user community, converting the scientific descriptions of the astronomical observations into detailed spacecraft activities which are linked in time. These detailed activities are formatted to be both readable by the telescope operations staff and directly schedulable by the planning and scheduling portion of the ground system. The implementation of the Transformation system is particularly interesting as the system was developed at the same time that its target system (the planning and scheduling system) was being built.

1   Introduction

The Hubble Space Telescope is a complex optical astronomical observatory which will be carried into low Earth orbit by the Space Shuttle in November of 1988. The HST incorporates five scientific instruments, each of which may be used with a large variety of filters, and in many different configurations. Additionally the fine guidance sensors, generally used for pointing the telescope, may be used to perform extremely precise positional measurements. A wide range of scientific investigations will be possible, including imaging, spectrometry, photometry, and astrometry, at distances as near as solar system objects, and as far away as the edge of the observable universe.

As the HST will be in a low orbit, there are many operational constraints that will interact with its scientific goals. For example, the Earth is still very large at that orbit, and any given astronomical object is typically occulted by the Earth for 40 minutes out of each 95 minute orbit. The telescope requires a relatively long time to slew to a new target and to settle and lock into position. The telescope will orbit beneath stationary communications satellites, but will only be in their footprints for short periods of time. Tape recorders are available, but as their lifetimes are limited, they must be used only when absolutely necessary. Power is limited, and the solar arrays must be kept close to their optimal position for collecting solar energy. The list of constraints and good practices is extensive.

The interaction between astronomical use of the telescope, and the details of its successful operation is important not only in efficiently scheduling observations, but also in the manner that those observations are planned and presented for scheduling. It takes considerable skill simply to correctly specify a series of spacecraft activities that perform the operation. In addition, a satellite as complex as the HST allows any particular goal to be accomplished in many ways. An operator knowledgeable in both the details of the satellite and its scheduling system can minimize the overhead involved in performing a set observations by finding an efficient plan.

The HST will be available to the worldwide astronomical community. As it is not reasonable to expect all astronomers to become experts in the operations of the spacecraft and its scheduling system, and it is costly to manually plan all observations, a system was developed to bridge the gap between a scientific proposal generated by an astronomer, and an efficient series of spacecraft operations which can be directly handled by the scheduling system.

2   Manual Transformation

The HST ground support system is being delivered to the Space Telescope Science Institute in a series of software "builds". As each update is delivered, the operations personnel generate procedures for using the new capabilities. For the scheduling subsystem, the manual procedures involve generating a short, handwritten "script" which maps observers' proposals into the data structures used by the scheduler to create activity timelines. Using these scripts, and

a "cookbook" of suggested practices, values for the fields of the data structures are derived and entered into the scheduling database (known as the PMDB). "Scripting" is performed by senior Operations Astronomers, who are knowledgeable both in observational astronomy, and the specifics of the HST. This process can take as short as a few hours or as long as a day. Deriving the values for each database relation and entering them field by field is the job of more junior Console Operators, and typically takes about as long as the script generation. Most of the derived operational details—when to turn on the tape recorders, which telemetry link (high or low bandwidth) is to be used, etc., are stored in these database relations and an expansion of data by a factor of 50 is typical. That is, for a single page of exposure descriptions on the proposal forms, about 50 pages of PMDB data is generated.

Specifically, the scheduling system utilizes the following hierarchy of data structures:

- exposure—a single data collection activity; most instrument parameters are stored at this level.

- alignment—a set of exposures that can be performed without moving the telescope; positional, pointing control, and certain special requirements are stored here.

- observation set—a set of alignments that can be performed while using the same guide stars for pointing; critical timing data is stored here.

- scheduling unit—the smallest entity upon which the scheduler operates. It is a set of one or more observation sets, and is the level at which timing relationships between exposures and any real-time decision points may be specified.

The documentation of the manual procedures varied. Details of the scripting process required considerable dialogue with the operations staff. The cookbook, however, was fairly complete, and served as a very good reference for generating rules. Our domain experts were extremely cooperative, as they were anxious to be automated out of the tedious job of manual transformation—we experienced very few problems in extracting techniques from our experts.

On the other hand, the manual procedures were in constant flux. As the ground system was being delivered piecemeal, the operations staff had to change their procedures as time went on. Each new delivery meant changes to old procedures, new database relations to be populated, as well as potential false starts for using new capabilities. The rulebased transformation system needed to be extremely flexible in order to reflect its constantly changing requirements.

3   Rulebased Transformation

The basic strategy behind the Transformation system was to emulate the manual procedures. This capability is a major advantage of a rulebased implementation. Paralleling the manual techniques also made it easier to change the program when operational procedures changed. In addition, the data structures of the scheduling system are somewhat arbitrary and poorly

conceived. In many cases, the operations staff had developed techniques for doing what was needed "in spite of" the scheduling system. Once again, because the code closely imitated their general procedures, it was fairly easy to incorporate these tricks as well.

In the automated system, the analog of a script is an "assignment record", an internal structure used to record the assignment of an exposure on a proposal form to an exposure in the PMDB, and the position of that exposure in an alignment, the alignment in an observation set, etc. Once all the assignments have been made, the derivation of values for all the fields in the database relations can be performed by rules which essentially implement the "cookbook".

The technique utilized for "merging" a set of structures of one level into a single structure on the next level is fairly straightforward. There are typically a few merging rules corresponding to general situations when it is useful to merge structures. These are backed up by a fairly large number of special case rules which undo the merge if it is impossible, inefficient, or a violation of a constraint.

For example, it is generally useful to place two exposures of a single proposal into the same alignment if they use the same instrument, observe the same target, and either the proposer specifies that they be performed sequentially, or they are simply listed consecutively on the exposure form (often an implied connection between two exposures). There are separate merging rules for each of these cases, and about a dozen exposure merging rules in all. There are approximately 80 exposure unmerging rules, however. For example, there is one instrument on board the telescope (the High Speed Photometer) which has no moving parts; each of its filters is permanently attached to a separate aperture. In order to change filters, while using the HSP, the telescope must actually be moved a very small amount. Moving the telescope contradicts the definition of an alignment, so there is one rule which removes merges between two otherwise mergeable exposures if they are HSP exposures which use different filters.

The output of the Transformation system is a single "assignment file" which is a complete list of all generated database relations and the derived values of each field in each relation. This file is produced in a machine editable, human readable format. A summarizing program can be invoked to generate a synopsis of the assignment file which is modelled after the "scripts" that the operations astronomers themselves generate when transforming by hand. This provides the operations staff with a comfortable and familiar format for checking the results of transformation. The psychologically important feature of an editable assignment file was a considerable aid to acceptance of the system by the staff, even though it is, in fact, rarely utilized. Once the output is verified, a batch program may be initiated to reformat the output file into a series of database commands which loads the data into the database.

The current transformation rule base has approximately 400 rules, but there have been a considerably larger number written over the course of its development. Many rules have been superceded by changed operational procedures, or, as typically happens, a large number of special case rules have been collapsed into a single more general rule. The system was delivered almost a year ago, and has been used in end-to-end ground system exercises. As the delivery preceded the final release of the scheduling system by 15 months, the rule base is still being updated to comply with changing operational needs.

ORIGINAL PAGE IS
OF POOR QUALITY

As OPS5 provides few constraints on program format, the rulebase is very readable to the domain experts. The rule names themselves provide a "table of contents" of the program's capabilities, but the experts have also been given the full program source for verification. They are able to suggest additions and changes simply by reading the rules, and in doing so have even come up with procedures that were not encountered during manual transformation—helping us create a rulebase in some ways more robust and complete than the manual methods. But the best proof of the readability of the rules, and the experts' trust in the system is that the few notes for manual scripting now say, "see cross-reference table for config./mode, or Don's OPS5 rules."

4 Summary

The Transformation system is a rulebased subsystem for ground support of the Hubble Space Telescope which bridges the gap between astronomical descriptions of observations by the telescope and detailed spacecraft plans which can be scheduled on a timeline. The system was built while its target system was still under development, and therefore had to adapt to constantly changing functional specifications, Transformation closely emulates the manual procedures and is able to generate output which resembles what would be produced by hand. The system has been successfully integrated into the ground support system, and has been used during end-to-end operational dress rehearsals.

Intelligent Interfaces for Expert Systems

by
James A. Villarreal and Lui Wang

Artificial Intelligence Section/FM72
Mission Planning and Analysis Division
Johnson Space Center
NASA

ABSTRACT

Vital to the success of an expert system, is an interface to the user which performs intelligently. The Artificial Intelligence Section of the Mission Planning and Analysis Division is developing a generic intelligent interface for expert systems. This intelligent interface has been developed around the in-house developed Expert System for the Flight Analysis System (ESFAS).

The Flight Analysis System (FAS), an MPAD developed product, is comprised of 84 configuration controlled Fortran subroutines that are used in the pre-flight analysis of the space shuttle. In order to use FAS proficiently, a person must be knowledgeable in the areas of flight mechanics, the procedures involved in deploying a certain payload, and an overall understanding of the FAS. ESFAS, still in its developmental stage, is taking into account much of this knowledge.

The generic intelligent interface involves the integration of a speech recognizer and synthesizer, a pre-parser, and a natural language parser to ESFAS. The speech recognizer being used is capable of recognizing 1000 words of connected speech. The natural language parser is a commercial software package which uses caseframe instantiation in processing the stream of words from the speech recognizer or the keyboard.

This paper will describe how the different systems are configured, the problems encountered in integrating these systems, the capabilities and drawbacks of the system, and future implementations.

INTRODUCTION

Aside from the developing of expert systems, the Artificial Intelligence Section is also interested in various man-machine interfaces. These include speech recognition and synthesis, natural language, and the use of graphics systems for information retrieval. The testbed being constructed

to demonstrate the various implementations of these man-machine interfaces is illustrated in figure 1.0. The intelligent man-machine interface workstation is not tailored specifically for the interfacing to ESFAS. Instead, in order that this configuration could be used for other expert systems, generality is kept a key consideration in the developing of this system. This paper describes the various components of the generic man-machine interface and what part they play in the overall system.

FAS AND ESFAS

Before describing the various components of the intelligent man-machine workstation a brief introduction to the Flight Analysis System (FAS) and the Expert System for the Flight Analysis System (ESFAS) is provided.

FAS is an MPAD developed product used by flight designers for pre-mission planning and analysis of space shuttle flights. The FAS consists of 84 configuration controlled Fortran subroutines which reside on the HP9000. To operate FAS to any degree of proficiency requires extensive training or knowledge in the areas of flight mechanics, procedures relevant to shuttle operations (such as the deployment of payloads), use of the UNIX VI editor, and a thorough understanding of FAS itself.

Even if a user is well versed in the areas of flight mechanics and the procedures involved with space shuttle operations, the analysis and planning via the use of the FAS is still quite extensive. A user must have considerable training pertaining to the functions and the inter-relations between each processor, and knowledge of the FAS executive. ESFAS, an expert system developed in-house, is an attempt aid the user intelligently. At its inner-most level, the recent version of ESFAS, has developed a set of rules for a few selected number of processors. These rules provide ESFAS with syntactical information on how these processors can be arranged within a sequence and the required inputs for a particular processor. Generally, this level of the expert system warns the user of syntactical incorrect sequences of processors or invalid data types.

An outer level of ESFAS, provides the user with the associated sequences of FAS processors for several typical developed and qualified manuevers for the space shuttle. A type of manuever may be the deployment of a booster for a satellite. Typical payload deployments could range anywhere from the highly complex Inertial Upper Stage (IUS) booster for TDRSS satellites to the relatively less complex Payload Auxilary Module (PAM). Each procedure has its own set of groundrules and constraints. This level, then provides the user with different type of typical manuevers thst the users can then modify to their needs. The ESFAS knowledge base checks for

SPEECH IN

KURZWEIL
KVS-3000
VOICE RECOGNIZER

IBM PC AT

SYMBOLICS LISP MACHINE

NATURAL LANGUAGE (LANGUAGE CRAFT)

LEXICON

SPEECH OUT

SPEECH SYNTHESIZER (DECTALK)

ESFAS ART-BASED

DOMAIN RULES

MULTIBUS

VSP-1100 (VOTAN) SPEECH RECOGNIZER

TOUCH SCREEN

GRAPHICS SYSTEM (RASTER)
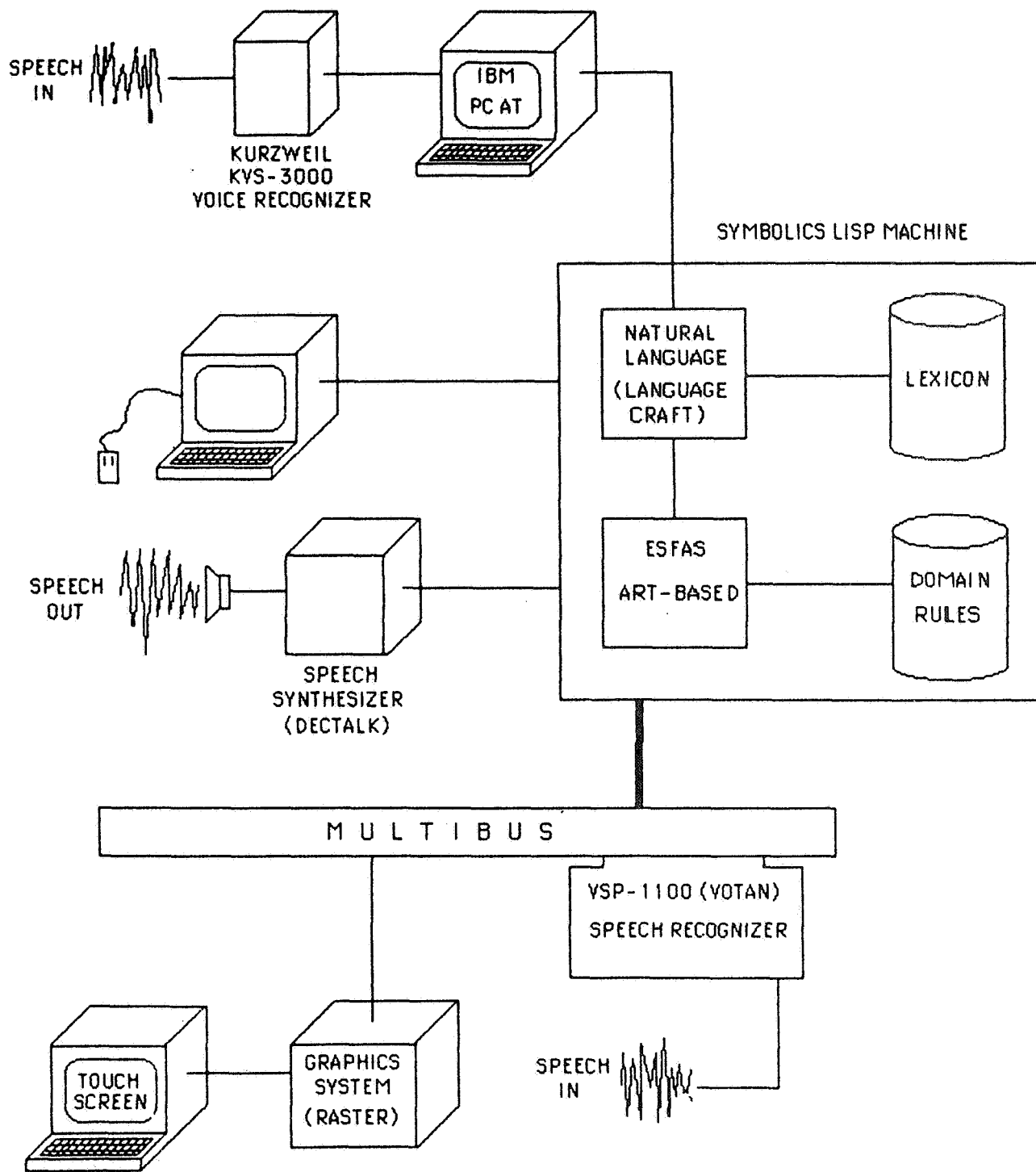
SPEECH IN

FIGURE 1. GENERIC INTELLIGENT INTERFACE WORKSTATION
TESTBED CONFIGURATION

421

and attempts to correct constraint violations while the user is constructing a particular manuever.

SPEECH RECOGNITION AND SYNTHESIS

Speech recognizers can be classified into belonging to one of four classes. The class I system recognizes isolated speech; i.e. speech which single word commands are suitable for the application. These type of recognizers can further be characterized as recognizing a small number of words (10 - 300). Because of the small number of words in their active vocabulary, they usually have the capabilty of recognizing commands from several speakers with little or no training.

The class II recognizer is either a connected connected or continous speech recognizer. For connected speech, a user must pause at least 1/6 of a second between words. Whereas, for continous speech, the user is allowed to run words together as in natural speech. The general tradeoff being that the continous speech system is restricted to managing a smaller number words than the connected speech system. The vocabulary size of this class varies from 50 to 1000 words.

The class III recognizer is the speaker identification system. These systems are usually intended for verification of personnel, via voice identification, before allowing access into secure areas. Because of the intensive training and screening required of the user, this system has not gained much propularity.

The fourth class of recognizer is the speech understanding system. Unlike others, this class of recognizer does not rely wholly on the acoustic signal, but makes use of concepts from the field of natural language, which is discussed in greater detail below.

The speech recognition systems currently in use by the intelligent man-machine interface testbed are the Kurzweil KVS-3000 and the Votan VSP-1100. The Kurzweil KVS-3000 is capable of recognizing 1000 words of connected speech. For this particular system, the user must undergo an "enrollment" as well as a training period. The enrollment phase is a method by which the system builds an individualized voice model. The data acquired from this phase is later used in the training phase to construct voice patterns for the application vocabulary. The ESFAS project is currently using a 350 word vocabulary. Also, the vocabulary has been partitioned into subsets. A subset is a scheme in which only a certain subset of words is active at any given time.

The Votan VSP-1100 is capable of recognizing 75 doubly trained words of

continous speech. At the writing of this paper, this recognizer has not been implemented into the ESFAS interface. But, plans are underway to have the VSP-1100 recognizer and the KVS-3000 alternate dynamically during the use of the ESFAS. The use of the VSP-1100 will be reserved for times when a need for continous speech is necessary or desireable (such as the entry of numeric data).

To query or direct input from the user, the Dectalk speech synthesizer is used. This is a phoneme based voice system; i.e., speech output is constructed by concatenating a series of phonemes. The constructing of a phrase usually goes through a trial and error process until the desired response is achieved.

NATURAL LANGUAGE

The most common and natural method which humans use to communicate is with language. These languages may be in the form of English, Spanish, Russian, French, etc. The communication of humans with computers, on the other hand, may seem quite awkward or inflexible. This rigidity is greatly due to the computer's need to *parse* the *grammar* into some form it can understand.

The ability for a computer to understand unconstrained grammar or Natural Language (NL) is an area of AI research which is receiving much interest. Present day systems are capable of processing a somewhat constrained grammar. But the free use of language into a computer, which is so natural and seemingly unconcious to humans, is still beyond grasp. Usually, NL systems are limited in that the user is restricted to a specific structure, *syntax*, or in the various different meanings that a word may have, *semantics*.

Although different schemes have been used in developing NL systems, the most popular present day methods are the knowledge based sytems. Generally, these systems rely on a large amount information about the specific domain. This knowledge is represented in the form of logic, procedural semantics, semantic networks, or caseframes.

The NL system currently in use by the generic intelligent interface testbed uses caseframe instantiation to process the input from the speech recognizers or the keyboard. The commercial name of the package is Language Craft and is developed and distributed by Carnegie Group. Basically, the programmar is supplied with a workbench of various tools necessary to develop a grammar. The major tools consist of the rewrite rules, caseframes, mapper rules, and lispifier rules. Basically, a caseframe is a representational structure of the language appropriate to the domain.

The template for a caseframe is shown below:

```
[ *name*
        :type sentential or nominal
        :header header pattern
        :nlg-token word or phrase
        :semantic-class person, place,
                thing or time
        :label pattern
        :predictable yes or no
        :cases
                (casename-1
                field-a value-a
                field-b value-b...)
                (casename-2
                field-a value-a
                field-b value-b...)
                ...]
```

On a successful parse, the parse is then passed to the mapper rules. Mapper rules are generally a method by which many caseframes may be coalesced into one. After this pass, the parse is then sent to the lispifier rules. At this point, the parse is matched against the lispifier rules. On a successful match, the appropriate lisp function calls are made.

GRAPHICS

At present, a graphics computer has not been installed into the system. In general, a graphics system will be used during the retrieval and manipulation of structural objects. As an example of how this could be useful, consider an astronaut at a space station workstation. Suppose that, due to failure, she needs to replace a certain camera. Of course, a crew activity planner would have to be consulted, before she can perform this task. Having scheduled a time for this EVA manuever, there is then the need of comparing the old camera with the new camera. Now, this comparing process could be accomplished by traditional methods; i.e., getting all this information from books or manuals. A more effective method would be to work with graphical representations with the use of a graphics system, because working with paper documents in a space environment is difficult.

A major stipulation expected of the graphics computer is that it be controllable through its bus directly. This is important because graphics systems inherently process large amounts of data in order to display images. Since the lisp machine will be the central controller, data

424

transfers must be as quick as possible.   These data transfers are achievable only by accessing the bus directly.

CONCLUSION

The impetus on this project is placed on developing a command/control workstation testbed functionally similar to that being proposed for use on the space station. From the proposed workstation, astronauts should be able to access or control the various subsystems on board the space station. These subsystems may be one of the spacecraft sustaining systems, such as the guidance navigation and control, or the crew monitoring systems, such as the crew health monitor or crew activity planner, or a payload experiment. Whatever the system a crewmeber is communicating with, it is critical that the interface be flexible, intelligent, and extremely user-friendly. These attributes of the system are a key importance because the ability for a crewmember to work efficiently will rest greatly on the user interface. For these reasons, it became apparent, that a feasibilty study of the various state-of-the-art user interface tools be researched and implemented.

The generic intelligent interface, as originally conceived, is presently not complete. To complete the system, a graphics interface or computer is still necessary. The graphics computer is especially needed during information retrieval. Therefore we are unable, at this time, able to make complete judgements on overall system performance.

However, for the systems which have been implemented, several deductions can be made. Considering the capabilities of present day NL systems, it would be reasonable to expect systems to operate without anaphora within the forseeable near future. Anaphora, the need to repeat a phrase or word, in order that the NL system can understand the meaning, is time consuming and a burden. As an example of anaphora, consider the editing of a document completely through the use of NL. Suppose the first command to the editor is to center the title. If the second command is to capitalize the first letter in each word, the editor should be aware that since an object has not been given, then this action is intended for the title and not the entire document. A straightforward scheme simply stores the noun of the sentence onto a stack for later retrieval. This scheme works for a few cases,' but restricted for full use. Another drawback of NL systems, are their dependence of grammar to a specific domain.

Presently, the speech recognizers are operating in an open loop fashion; i.e., output from the recognizer is going directly to the NL system. Therefore, the recognizer is not exploiting knowledge from the NL system which may be   useful in selecting or overriding a particular word. Most speech

recognizers can be set to output the words which scored the highest during the matching process. A pre-parser expert system is being developed which will select a word based on its grammatical context. No known commercial NL systems have inherent interfacing methods for use with voice systems.

Requirements for a Tool to aid in the Development of Expert Systems for Space Station Applications

Chris Culbert
NASA/Jonhson Space Center
Mail Code FM7
Houston, TX 77058
713-483-3921

ABSTRACT

The Space Station Program is currently in the early stages of design. The extended life of the program and the wide range of planned capabilities present NASA with a major challenge to make effective use of all available resources. Expert systems may represent one of the most important techniques NASA can use to improve performance and functionality of the Space Station. The recent growth in expert system applications has been spurred by the availability of commercial expert system development tools. These state-of-the-art tools provide multiple representation techniques, sophisticated user interfaces, and powerful development tools; allowing programmers to develop expert systems without requiring years of training in knowledge engineering. This document describes the requirements for a software tool to aid in the development and delivery of expert systems intended for use in the Space Station Program. The intent is to define a tool that would be useful for a very broad range of expert system applications.

INTRODUCTION

This document describes the high level requirements for a software tool to aid in the development and delivery of expert systems intended for use in the Space Station Program. The intent is to define a tool that would be useful for a very broad range of expert system applications. These requirements have been gathered by the Artificial Intelligence Section, Technology Development Branch of the Mission Planning and Analysis Division (NASA/JSC) with inputs from most NASA centers through the Expert Systems Technology Working Group.

The Space Station is being designed to accomodate growth and evolution. The tools used to develop expert systems for the Space Station must also be prepared to evolve and meet the needs for advanced expert system technology. Some of those capabilities can be envisioned already, although they may not be readily avilable in current expert system tools. Therefore, this document will outline a set of initial requirements, those capabilities which are currently available and should be included in the tool

immediately, and a set of future requirements, those capabilties which should be included as the technology matures.

REQUIRED CAPABILITIES

The following characteristics are required by any general purpose expert system tool used for Space Station applications. All of these capabilities are available in current, state-of-the-art tools and should be included in the initial release of the tool. Although they may not all be used for every application, the developer must be able to choose those capabilities that are needed and put them to use to solve a problem, like choosing a tool from an available tool set.

Knowledge Representation Requirements

This section describes the capabilities the tool must have to represent knowledge (e.g. the methods provided for describing a problem).

Rules

The tool must provide a rule system for representing knowledge. The rule system will provide a robust and powerful rule syntax with multiple variable bindings, predicate functions, and free form patterns on the left hand side of a rule. There will be no limitations on the number or type of patterns, or on the number of rules.

Object Description

The tool must provide an object description language. The tool will provide a robust frame system with user-defined relations between objects. The object description language will provide full inheritance of properties (attributes, default values and procedural attachments) between objects. The user must be able to control or redefine inheritance as needed. Classes of objects will be able to inherit from several superclasses that are not necessarily descendants of the same class (multiple inheritance). Procedures will be able to be local to several classes that do not necessarily have any common superclass (multi-methods).

Hypothetical Reasoning

The tool will provide a method for hypothetical reasoning. Hypothetical reasoning allows an expert system to consider multiple solutions to a problem. It is important in problems that require consideration of numerous alternatives to find the best solution.

Uncertainty Management

The tool will have the capability to reason about uncertain data. Information isn't always known with complete certainty, many problems that require expertise deal with uncertain knowledge. An expert system built to solve

428

such a problem must have a way of handling this uncertainty.

Truth Maintenance
The tool will have the capability to define logically dependent information and automatically maintain the consistency of the fact database. Truth maintenance allows the development of complex logical relationships between objects and aids the verification of such systems. The truth maintenance system must also be able to handle logical contradictions under appropriate circumstances.

Knowledge Sources or Rule Sets
The tool must support organization of the knowledge base into distinctly modular sets of rules or knowledge sources. The tool must allow rule sets (or objects or other representation units) to be activated only as they apply to a problem. This will support incremental development by multiple experts, improve performance by assuring only appropriate knowledge is considered, and improve modular testing and verification.

Blackboard Mechanism
The tool must support the integration of an underlying "blackboard" system to provide central storage of data for communication between distributed expert systems. A blackboard could also be used to support transfer of information between user applications and expert systems.

Constraint Propagation
The tool must support constaint propagation. Given a set of initial values, constraint propagation assigns a value to each cell that satisfies the constraints. The basic inference step consists of finding a constraint that allows it to infer a previously unknown cell value. Potentially, other unknown cells could be inferred by continuing the inference process as more cell values are determined. Constraint propagation is useful in situations where there is incomplete information about the values of state variables and where the relationships among them can be expressed as constraints.

Meta-Reasoning
The tool must provide the ability to include meta-knowledge about the application. This would include such features as: rule priorities to control rule firings, meta-rules to load or choose applicable rule sets, and potentially, the knowledge of where an application is valid and what the boundries of the knowledge base are.

Inference Engine Requirements
This section describes the capabilities the tool must have to utilize or apply the knowledge stored in the knowledge base.

Forward Chaining
The inference engine will allow forward chaining of rules. Forward chaining allows an expert system to reason from known data to a conclusion. It is important for problems that are data driven, i.e. those where the solution is not necessarily chosen from a small set of potential solutions.

Backward Chaining
The inference engine will allow backward chaining of rules. Backward chaining allows an expert system to assume a solution and reason back to the data that supports the assumption. It is important for goal-driven problems, i.e. those with a limited number of solutions, or where it is preferable to prove a particular solution than to analyze all the available data.

Procedural Attachments
The tool must also allow local procedures to be "attached" to an object to provide full object oriented programming capability. Procedures will be written in any procedural language available to the software development environment. The tool must also allow the automatic execution of procedural code when a value in an object's slot is changed or accessed in any way (i.e. "active values").

User Defined Inference Engine
The tool must allow users to define their own inference engine to be used in place of the default inference engine provided by the tool.

Integration of Features
The tool must not only have all the previously described characteristics, they must be fully integrated with each other. Users must be able to freely mix forward and backward chaining rules and reason about information stored in objects. Hypothetical solution techniques must be able to access both objects and rules. Objects must be able to represent a group or set of rules. Both object attribute values and free form facts must allow certainty factors to be assigned to them. Rules must be able to reason about the certainty of a fact. Both objects and rules must be able to take advantage of the truth maintenance system. Procedural atttchments must be able to trigger rule firing and rules must be able to cause execution of procedural attachments. Overall, complete integration of features is required to provide a robust environment for the development and execution of expert systems.

Delivery Environment Requirements
This section describes the capabilities the tool must have for delivery and use in the expected Space Station computing environments.

Integration with other Languages

The expert system must be able to call functions or programs developed in other languages. The tool will be able to call functions developed in any language which will run on the host hardware. Developers must be able to call predicate functions on the left hand side of a rule, and any kind of function from the right hand side of the rule. Procedural attachments to an object (methods) will also be available in any external language used in the Space Station Software Support Environment (SSE).

Embedded Expert Systems

The tool will allow expert system applications to be fully embedded in a larger system. For most Space Station applications, an expert system will merely be a part of a larger system, improving the "intelligence" of that system. For this to occur, the entire expert system, it's inference engine, knowledge base, etc., must be completely embeddable within a larger system. A call to the expert system will be similar to any other subfunction call. This capability is also required to allow the use of expert systems in a distributed processing environment.

High Performance

The tool will provide sufficient performance to allow it's use for near real-time system monitoring and control functions. This means that the rule system should be fully optimized with a rule compiler and efficient pattern matching techniques. Also the tool must be capable of handling large object databases with potentially thousands of objects.

Hardware Independence

The Space Station may not provide specialized hardware for the execution of automated functions. To facilitate the use of automation, the tool will provide a version of the inference engine (and other necessary internals) capable of running on conventional computing hardware.

Interface Support

The tool will support all display and interface capabilities provided as a part of the standard Space Station configuration. At the minimum, this includes high resolution displays, bit mapped graphics, multiple-windows, pointing devices (mouse, light pen, touch sensitive, etc), natural language, and speech recognizers. The tool must provide "hooks" to allow the use of any of these capabilities in a delivered system, and for advanced interface technologies as they become available.

Explanation Facility

The tool must provide a standard framework for the an explanation subsystem which could be incorporated in any expert system application. The explanation subsystem should be able to explain the operation of the

expert system at all stages of a logical activity and must be able to handle activities which span many sessions. Depth and level of explanation must be user controllable. The tool should allow explanations all the way down to the level of identifying specific rules.

Development Environment Requirements

Even with an expert system tool, the development of an expert system is still a complicated process. This section describes the capabilities the tool must provide to aid the development of an expert system application.

Intelligent Editors

The tool will include an intelligent text editor that can detect syntax or typographical errors prior to compilation. The editor should also aid documentation of the development effort and provide tools to improve the readability of the code. Graphical editors can also be useful for the development of object hierarchies and should be provided where possible. The graphical editors will create readable, maintainable code.

Graphical Interface

The tool will make extensive use of graphical interfaces including features such as menus, multiple windows and pointing devices.

Debugging and Tracing Aids

The tool will provide features to aid the debugging of an expert system application. The features will include capabilities such as: tracing rule execution, single step rule execution, examine current state of the system, modify current state, explanation of the expert system decisions and graphical depictions of object hierarchies.

Incremental Compilation

The tool will provide incremental compilation of rules to speed the prototyping and debugging process.

On Line Help

The tool will provide extensive on-line help with full documentation of features including examples. The tool should also provide the framework for building on-line help systems for applications.

Interface Development Aids

The SSE will provide the capability for the development of advanced interfaces. The tool should allow use of these capabilities to their fullest extent.

FUTURE CAPABILITIES

The following characteristics features that will be important for Space

Station applications when the technology becomes available. These capabilities should be planned for in future releases of the tool. As with the required capabilities, they may not all be used for every application.

Access Control/Multi-user Support

The tool shall be capable of producing expert systems which can support multiple users either sequentially or simultaneously. This includes the maintenance of an integrated knowlededge base available to all users, and communications mechanism for coordinating multiple sessions.

Rule Induction

The tool should include a means of inducing rules from an example set of problem solutions. There are tools currently available which can do this for limited problems, but none which are able to work within the framework of a robust, hybrid knowledge representation system.

Parallel Processing

Evolution of the Space Station should provide parallel processing for very high performance systems. The tool must be able to take advantage of this capability as it develops.

BENCHMARKING EXPERT SYSTEM TOOLS

Gary Riley
NASA/Johnson Space Center
Mail Code FM7
Houston TX 77058
713-483-3921

As part of its evaluation of new technologies, the Artificial Intelligence Section of the Mission Planning and Analysis Division at Johnson Space Center has made timing tests of several expert system building tools. Among the production systems tested were ART, several versions of OPS5, and CLIPS (C Language Integrated Production System), an expert system builder developed by the AI section. Also included in the test were a Zetalisp version of the benchmark along with four versions of the benchmark written in KEE, an object oriented, frame based expert system tool.

The benchmark used for testing was a modified version of a planning problem known as the monkey and bananas problem in which a monkey must plan how to overcome a series of obstacles in order to eat a bunch of bananas. The monkeys and bananas problem is prototypical of certain planning problems. In this kind of problem a final objective or goal is desired (e.g. the monkey wants to eat the bananas). Typically the initial goal cannot be satisfied, therefore subgoals must be generated and accomplished. For example, the monkey must be holding the bananas to eat them and he is not holding them, therefore a goal must be generated for the monkey to hold the bananas. The monkey must be at the same location as the bananas in order to hold them and he is not, therefore a goal must be generated for the monkey to walk to the location of the bananas. And so on. Although this problem does not represent a "real world" problem, it is prototypical of such problems. An expert system tool does not know whether or not the rules it is executing are rules for a "real world" problem. Thus, in an expert system benchmark it is necessary only to create a set of rules that is typical of "real world" expert systems. The monkey and bananas problem meets this criterion.

The benchmark for the production systems consisted of 30 rules and it required 81 to 86 rule firings to obtain the solution (depending upon the production system used). The discrepency in the number of rule firings was attributed to differences in the inference engines. In ART, for example, attempting to place an already existing fact into the knowledge base is a null operation, whereas in OPS5 the fact would be placed in the

knowledge base but with a distinct timetag. Also, the conflict resolution schemes (the mechanism which determines which rule will fire next) are different in ART, CLIPS, and OPS5. Because of the similarities in the production systems, there was a one-for-one translation for each of the benchmark rules.

Four versions of the KEE benchmark were done, two by the AI section and two by Intellicorp. All of the versions of the KEE benchmark used the object description facilities of KEE to describe the problem. In two of the versions, reasoning was accomplished through the use of lisp methods (message passing between objects). In the other two versions, reasoning was accomplished through the use of rules: forward chaining in one version and backward chaining in the other version. The Zetalisp version of the benchmark was very similiar to the KEE version using lisp methods, however, instead of using KEE to describe the objects, the flavors capability of Zetalisp was used. The ZetaLisp flavors version was primarily provided to give an indication of the cost of high order languages.

The comparison of object oriented systems with production systems is subject to a lot of interpretation. The speed of execution of each of the systems is heavily dependent upon the efficiency of the implementation, and it is extremely difficult to insure that the implementation of the benchmark is the most efficient implementation for both kinds of systems. One cannot draw hard conclusions about object oriented vs. rule based paradigms directly from the run time speeds of a benchmark . For the typical user, however, such a benchmark provides extremely useful information. The average user is only able to make a "best try" when solving a problem. He does not necessarily have the inside information that the designer of the system would have when solving the problem and will probably not be able to code it as efficiently as the designer. With this in mind, the benchmark can provide information on the expected speed of a typical attempt to solve the problem. One must consider that this benchmark does not necessarily represent the speed of the most efficient implementation, but of a typical implementation, and weigh the results accordingly. Because of the inherent differences between rule based tools and object oriented tools, separate tables have been used to show the results of the benchmark.

Some of the benchmark results were quite surprising. Not unexpectedly the production systems running on the lisp machines did the best. Vax OPS5 did very well compared to the lisp machines, but among all the expert system tools it was the only one which did not provide a full interactive programming environment (specifically the ability to incrementally compile rules). Running benchmarks on lisp machines can be

somewhat tricky and misleading. Due to the small size of this problem, the amount of memory available on the lisp machines had little effect on the final run time speed. That is, running the problem on a 4 megabyte machine was not any faster than running the problem on the same machine with 2 megabytes. However, because the lisp machines use virtual memory, the first run of a program is usually slower because the program must be paged into memory. To illustrate, the first run of the Zetalisp flavors version on the LMI took 15 seconds, while all subsequent runs took less than 0.4 seconds. The benchmark results indicate the most optimistic (that is the fastest) speed seen over several runs This is a slight disadvantage for computers such as the Vax 11/780, the HP 9000, and the PC's because the run times on these computers tended to be fairly constant. This benchmark, however, was not intended to provide hard and fast numbers on execution speed, but rather provide a relative scale and give a general idea of expectations for different expert system tools running on different machines.

The language the tool was developed in affected its performance. OPS5+ outperformed ExperOPS5 by about a three to one ratio on the Macintosh. This is probably attributable to the fact that OPS5+ is written in C and ExperOPS5 is written in ExperLisp. ART running on the Vax did not run as fast as ART running on the Symbolics. Lisp on the Vax runs about 5 times slower than lisp on the Symbolics and Vax ART was written in DEC's common lisp, so this may be the major cause of the poorer performance. Also the Vax ART was a beta test version. Lisp is well-suited as the design language for expert system tools, however, tools written in lisp will most likely suffer a performance penalty on machines with architectures that are not optimized for running lisp.

The KEE benchmarks done by the AI section did not do well on this test when compared to the other expert system tools. Intellicorp was given the opportunity to improve the benchmarks results. The Intellicorp object oriented version was a modified version of the benchmark done by the AI section and obtained most of its speed improvement by reusing units. The AI section implementation created objects dynamically. Intellicorp changed that logic to limit the amount of objects created dynamically. Apparently, the creation of objects in KEE is an "expensive" operation. Limiting the dynamic creation of objects, as done in Intellicorp's version, is a technique that should be applicable in many, but certainly not all, problems. Neither of the KEE rule versions ran extremely fast. Of the rule versions, the backward chaining solution (which was done by Intellicorp) did the best and seemed to be the most natural implementation of the problem using KEE.

While the appropriateness of a particular paradigm (e.g. rule based, object

oriented, or procedural) for a particular problem is subjective, the AI section believes that this benchmark is most naturally solved using rules. Therefore, while the KEE Lisp methods benchmark and the ZetaLisp flavors benchmark had fast execution times, we believe that the use of Lisp to accomplish reasoning for this particular problem does not allow for all the advantages associated with higher order reasoning methodologies such as rules.

While benchmarks provide useful information, one should avoid drawing far reaching conclusions from them. This benchmark is just one of many possible benchmarks and does not fully test the capabilities of the expert system tools. One might be fairly safe in drawing the conclusion that OPS5+ is a better optimized implementation than ExperOPS since the design and purpose of the tools are nearly identical. However, to draw the conclusion that ART is a better expert system tool than KEE or that KEE is a better expert system tool than ART is beyond the scope of this study. ART and KEE have completely different design philosophies and their strengths lie with different types of problems.

BENCHMARKING EXPERT SYSTEM TOOLS

* Hardware tested

    * SYMBOLICS 3640 with 4 Megabytes of memory

    * LMI with 2 Megabytes of memory

    * TI EXPLORER with 4 Megabytes of memory

    * VAX 11/780

    * HP 9000

    * IBM PC with 1/2 Megabyte of memory

    * IBM AT with 1/2 Megabyte of memory

    * Macintosh with 1/2 Megabyte of memory

* Software tested

    * ART - Inference Corporation

    * CLIPS - NASA MPAD AI Section

    * OPS5 - Charles L. Forgy

    * OPS5 - DEC

    * OPS5+ - Artelligence Inc.

    * ExperOPS5 - ExperTelligence

    * KEE - IntelliCorp

BENCHMARKING EXPERT SYSTEM TOOLS
(RULE-BASED APPROACHES)

TOOL (version) [1]	MACHINE	TIME (sec)	RPS[2]
ART (V2.0)	SYMBOLICS	1.2	86 : 71.7
CLIPS (V3.0)	SUN	1.2	86 : 71.7
OPS5 (VAX V2.0)	VAX	1.3	81 : 62.3
OPS5 (Forgy VPS2)	SYMBOLICS	1.7	81 : 47.6
ART (V2.0)	TI EXPLORER	2.4	86 : 35.8
CLIPS (V3.0)	VAX	2.5	86 : 34.4
ART (V2.0 Beta)	LMI	3	86 : 28.7
CLIPS (V3.0)	HP 9000	4.0	86 : 21.5
OPS5+ (V2.0003)	IBM AT	5.2	81 : 15.8
CLIPS (V3.0)	IBM AT	7.0	86 : 12.3
ART[3,4]	SYMBOLICS	7.6	N/A
OPS5+ (V2.0002)	Macintosh	14	81 : 5.8
ART (V Beta 3)	VAX	17	86 : 5.1
KEE (V2.1.66)[4,5]	SYMBOLICS	17.8	N/A
OPS5+ (V2.0003)	IBM PC	19	81 : 4.3
CLIPS (V3.0)	IBM PC	21.1	86 : 4.1
ExperOPS5 (V1.04)	Macintosh	55	81 : 1.5
KEE (V2.1.66)	SYMBOLICS	165	84 : 0.5

[1] While benchmarks provide useful information, one should avoid drawing far reaching conclusions from them. This benchmark is just one of many possible benchmarks and does not fully test the capabilities of the expert system tools. For example, to draw the conclusion that ART is a better expert system tool than KEE or that KEE is a better expert system tool that ART is beyond the scope of this study. This table by itself may present information out of context. It is necessary to read the text of the paper which accompanies this table for a fuller explanation. This table does not contain all of the benchmark results.

[2] Total number of rule firings : Rules per second

[3] Benchmark implemented by Inference.

[4] Benchmark implemented using backward chaining rules. All other benchmarks on this table were implemented using forward chaining rules unless noted otherwise.

[5] Benchmark implemented by Intellicorp.

BENCHMARKING EXPERT SYSTEM TOOLS
(OBJECT ORIENTED APPROACHES)

TOOL (version) [1]	MACHINE	TIME (sec)
Zetalisp Flavors	SYMBOLICS	0.067
Zetalisp Flavors	LMI	0.3
KEE (V2.1.66) (Lisp)[2]	SYMBOLICS	0.44
KEE (V2.1.66) (Lisp)	SYMBOLICS	6

[1] While benchmarks provide useful information, one should avoid drawing far reaching conclusions from them. This benchmark is just one of many possible benchmarks and does not fully test the capabilities of the expert system tools. For example, to draw the conclusion that ART is a better expert system tool than KEE or that KEE is a better expert system tool that ART is beyond the scope of this study. This table by itself may present information out of context. It is necessary to read the text of the paper which accompanies this table for a fuller explanation. This table does not contain all of the benchmark results.

[2] Benchmark implemented by Intellicorp.

TWO IMPLEMENTATIONS OF THE ESFAS PROJECT

by
Lui Wang
NASA/Johson Space Center, FM-72
Houston, Tx 77058
713-483-3921

ABSTRACT

The purpose of this paper is to make a comparison between the two most sophisticated expert system building tools ,the Automated Reasoning Tool (ART) and the Knowledge Engineering Environment (KEE). The same problem domain (ESFAS) was used in making the comparison. The expert system for the Flight Analysis System (ESFAS) acts as an intelligent front end for the Flight Analysis System (FAS). FAS is a complex configuration controlled set of interrelated processors (FORTRAN routines) which will be used by the Mission Planning and Analysis Division (MPAD) to design and analyze Shuttle and potential Space Station missions.

This paper describes implementations of ESFAS. The two versions represent very different programming paradigms; ART uses rules and KEE uses objects. Due to each of the tools' philosophical differences, KEE is implemented using a depth first traversal algorithm, whereas, ART uses a user directed traversal method. In conclusion, either tool could be used to solve this particular problem.

INTRODUCTION

The Flight Analysis System (FAS) is an executive environment, which controls a set of processors. There are two major components of FAS: the executive (EXEC) and the processors. The FAS executive has a set of global commands and provides an active work area (AWA) for data storage. The processors are a set of individual standalone programs that perform specific functions. In order to execute a processor, the FAS user has to set up an interface table, which is a text file that specifies input/output conditions. The result of execution of a processor is usually in the form of a display with an output data file, which is written in the AWA. Users typically set up a FAS run by stringing the FAS commands together in a sequence table and filling in an interface table for each of the processors they wish to execute.

Although the FAS environment is well implemented and many features are

PRECEDING PAGE BLANK NOT FILMED

included to provide on-line help and aid, the system still requires users to have a detailed understanding about the executive and the processors. This is where expert system technology offers its benefits. First, the expert system should handle the bookkeeping and special syntax which is imposed by the executive and the processors. Furthermore, the expert system should catalog all the well understood procedures along with their constraints and input requirements. Finally, the expert system should present its capabilities to the user or synthesize the input requirements from the user to select a set of required processors for the simulation.

PROTOTYPE DESCRIPTION

The current goal of this expert system (Expert System for the Flight Analysis System, ESFAS) is to derive a suitable knowledge representation, which would help the users do "house keeping" chores (ie, special commands and syntax) and would establish an organization to store the well understood procedural knowledge. Later, rules can be written to combine one standard procedure to another standard procedure to form hybrid procedures which may be used to solve a complex problem.

In time, FAS will have over 90 processors available to users, which can be used in various combinations to solve hundreds of problems. However, the number of processors is too large for the prototype development. Thus, the problem domain is greatly restricted as to a small subset of the FAS capabilities:

ESFAS should construct a basic Trajectory Event Timeline for a shuttle mission, from OMS-2 cutoff to Deorbit ignition, and allow for any number of simple orbit adjustment maneuvers.

This restricted problem requires the use of six FAS processors; BASTM, PHYDM, TFSV, INVAR, ACOAST, and GPMP. The last two processors are used multiple times if there are multiple maneuvers. The prototype should produce a sequence table and a set of interface tables for each of the processors that are used by the sequence table.

A goal driven technique is used to solve this problem. Because experts usually divide the problem into many sub-tasks (ie; well understood procedures). When all the proposed sub-goals (or sub-tasks) are solved, the original problem is automatically fulfilled. Here is an example of how an expert logically divides this particular problem into the following sub-tasks:

(1) Creating a trajectory-event-timeline entails creating an ephemeris or creating a TET-array.

(2) Creating a TET-array entails executing the INVAR processor and specifying the number of maneuvers.

(3) Executing the INVAR processor entails initializing the state vectors, and the time and specifying the input/output units.

(4) Specifying the number of maneuvers entails executing the number of pairs of ACOAST and GPMP processors.

(5) Initializing the state vectors entails executing the TFSV processor.

(6) Initializing the time entails executing the BASTM processor.

(7) specifying input output units entails executing the PHYDM processor.

THE TWO IMPLEMENTATIONS OF ESFAS

Since the basic program strategy was proposed by the last section of this paper, the implementation techniques and the appropriate knowledge representation for each tool are left to be discussed.

The Automated Reasoning Tool (ART) is a rule-based production language. It is similar in philosophy to OPS5, the classic example of a forward chaining, data driven programming language. There are three basic components in a production language: the rules, the facts, and the inference engine. Rules are usually used to capture heuristics and an expert's experiences. Facts are used to represent the current state of the system. Finally, the interactions between the rules and facts are driven by the inference engine, and rule execution is based on the existence (or non-existence) of specific facts in the database.

With this basic notion of a production language, ART uses a user directed traversal methodology to solve this problem. For example, a FAS user specifies that he would like to create a trajectory event timeline; then the system responds with a set of sub-tasks which he must do. At this time the user has a choice of picking which tasks he would like to do first. Then the system will prompt the user with a set of case specific questions. Other sub-tasks may be generated based upon the previous responses. Strict data type checking is done along the way to ensure "good" user inputs. Eventually, when all the sub-tasks are satisfied, the user is left sequence and interface tables free of syntax errors.

445

However, run time logic errors may not be completely detected at this point in time.

This program has not been implemented to its full extent. So far, all processor information is initially stored in LISP structures. Only the necessary processors (determined by rules) are brought into the database for doing reasoning. There are basically two rule sets in the system; one is the inter-processor relation rule set and the other is the intra-processor relation rule set. These two rule sets are the main governing mechanism to create or delete sub-tasks. Finally, the interface is written mostly in LISP and FLAVORS, since the feature required for the system was not fully supported by ART at the time of the initial prototyping.

The Knowledge Engineering Environment (KEE) is a frame based object oriented language which is very different from rule-based languages. Information is described in terms of objects or units. Each unit is associated with many attributes or slots. Slot values can be another object or a procedural function called a method. In addition, units can be related to other units either by the sub-class or the member-class relationship. KEE has a special slot called an active value, which is like a method, except the active value is invoked automatically upon slot access. This feature is desirable, for example, when a slot has changed its value. The screen can then automatically receive an update message. Thus, the user does not have to call the screen updating function explicitly. KEE has a rule system, but, it is slow and difficult to used. The rule system is used only when one needs to describe a complex relationship which could not be modelled by the hierarchy or attached methods to some units.

In this version of the implementation, the expert knowledge is being stored in three different places. First, the FAS processor information and hierarchical relationships are modeled by memberclass relations. Next, other inter-processor relationships are modelled through the dependency slots. Finally, the intra-processor relationships are modelled by the rule system, because the relationships are too complex to express in terms of methods. In general, there is a top level control loop, which guides the user traversing through the knowledge base and generates a new knowledge base along the way. Upon completion, the system extracts information from the new knowledge base and creates error free sequence and interface tables.

The initial prototype for this version is finished. There is a large amount of LISP code for directing the program flow. Since the system is not required to have a very fancy interface, one can use some of the interfacing tool kits provided by the KEE system.

446

FINAL DISCUSSION

Since an objected oriented language provides a strong notion of organization and the FAS system has a natural sense of structure, one would presume KEE is the ideal solution for the problem. However, rigid knowledge representation encroaches upon the user's flexibility. In a rule-based system, one has the flexibility to pick and choose, as long as all the constraints are satisfied at the end. In an object oriented system the program is controlled by a top-level function and all messages are passed explicitly through the program control. When the system generate a sub-goal, the user has to answer the system's request or it can not process any other information. In spite of this disadvantage, objected oriented languages do provide a nice organization and many problems fit well within this programming paradigm. Most of all, the ESFAS project can be solved in either tool.

REFERENCES:

(1) Culbert C., NASA Memorandums "ART Evaluation" and "KEE Evaluation" FM7 (86-9) and FM7 (86-10) respectively.
(2) Culbert C. Wang L. and Flinn H. "ESFAS An Intelligent Interface for the Flight Analysis System" ROBEXS '85 ISA, 1985 pp 215- 220.

Life-Span Knowledge Engineering for Space Operations

Dan Hays
The University of Alabama in Huntsville

Ordinarily we think of "knowledge engineering" as the process of translating the knowledge and problem-solving strategies of a human expert into rules and procedures incorporated into a machine based "expert system" which can, given adequate input, solve the same sorts of problems as the expert. This process is suggestively called "engineering" because it involves some art as well as science, and requires certain adjustments, approximations, and fine-tuning to represent the information and skill. By extension, some uses of "knowledge engineering" and "expert system" have come to refer to the building of other kinds of discretionary, information-bearing computer systems.

One appeal of these knowledge-based systems is their ability to take care of problems without having a human expert present. For work in Space, being independent of humans is especially important both for situations where devices will be in remote or dangerous locales where people are not present; and for situations such as space stations where human resources are limited and schedules are tight so that machines to a large extent must care for themselves. Another appeal seems to be that once a system is designed, the expense and bother of human care can be avoided.

In qualification of the above ideas, this paper argues that *our notion of knowledge engineering, and our expectations for its application, should be extended beyond the period of construction of unit expert systems, to the entire "knowledge-system management" associated with one or another real system,* whether it is a piece of hardware or an entire human-machine operation such as a lunar factory. Humans will in fact be involved not just as occasional users of the computer-resident systems, but will have to be involved from time to time in maintaining and updating them, in some cases repairing them, and in retailoring them when the machine systems or functional contexts change.

We are undergoing a critical transition in the nature of many automated knowledge systems. The change is from special-purpose systems acting more or less in isolation, to large configurations of knowledge-base processing units. This transition is being caused probably more by requirements of Space programs than any other source. The pressure to design heavily automated environments for a near-range Space Station in particular is causing us to shift our thinking from building and working with one expert system at a time, to preparing many machine-resident intelligent systems that share facilities and map the interaction of complex external events and internal interpretations. (Similar multiply interactive systems will come to be important in some industrial and military settings, also, as the consequences are worked out in industry of linking automated production facilities, and as military systems provide for really complex device/human interaction.) It is not clear that we have adequately confronted the implications of

having to build such systems. It is certainly not too soon to discuss *desiderata* for them.

Three key ideas, then, are involved here.

First is recognizing that we should *extend our period of concern for knowledge-base systems from construction and initial validation to their entire life-span of use.*

Second is a curiosity about *the useful forms of intelligence-bearing machine systems for the next decades of work in Space (and, indeed, elsewhere).*

Finally—really a part of making the life-span notion workable—is the realization that *the knowledge engineer of Space systems may have to be concerned not just with the original experts whose ideas were represented in machine rules and information, but with all the various users, builders, subcontractors, managers and so on who will directly affect or be affected by the system.* Explicit concern with, if not direct management of all of these is necessary if the knowledge-base systems and the object-systems they refer to are to be used effectively. We have thus gone from knowledge engineering to knowledge-system management, but have yet to work out the details of how best to bring this about.

"Knowledge Engineering" in the Early 1980's

The term "knowledge engineering" originated among artificial intelligence researchers to describe what they were doing, according to Hayes-Roth, Waterman and Lenat (1983, p 12). The theme of basic versus applied research in artificial intelligence was discussed in April, 1981, by Nils Nilsson in a talk at Carnegie-Mellon University, titled "Artificial Intelligence: Science, Engineering, or Slogan?", printed in the Winter 1981-82 issue of *AI Magazine.* What came to be called "expert systems" would have been classified by Nilsson as "second-level applications", those that were primarily concerned with applying expertise in a field of application rather than "first-level applications", such as MYCIN, which were undertaken largely to elucidate "basic AI knowledge about the core topics" of the field (p. 7).

By the time of the 1983 publication of *Building Expert Systems,* edited by Hayes-Roth, Waterman, and Lenat, the term "knowledge engineering" was widely used. Indeed, the appearance of this book probably solidified the acceptance of the term, along with "expert system", among the wider audience for information technology.

In their introductory essay, Hayes-Roth and fellow editors identified knowledge engineering with the construction of expert systems. "Extracting, articulating, and computerizing the expert's knowledge constitute the essential tasks in this area." (1983, p. 12). "Knowledge engineering addresses the problem of building skilled computer systems, aiming first at extracting the expert's knowledge and then organizing it in an effective implementation." (p. 13).

Ten "generic categories" of knowledge engineering applications were identified: Interpretation (inferring descriptions from sensor data), Prediction, Diagnosis, Design, Planning, Monitoring, Debugging (defined as "prescribing remedies for

450

malfunctions"), Repair, Instruction ("diagnosing, debugging, and repairing student behavior"), and Control in the broad sense of monitoring, diagnosing and correcting a referent system. Of these, it would appear that most applications counted as expert systems by mid-decade (for example, as listed in Waterman, 1986) have focused on diagnosis and remediation, with these subsuming monitoring and debugging and sometimes being undertaken in a systems control context.

(By contrast, general studies in artificial intelligence have dealt more broadly with intelligent functions, judging from papers included in the last several proceedings of the American Association for Artificial Intelligence and the International Joint Conference on Artificial Intelligence.)

The Life-Span Concept

That researchers in artificial intelligence during these years did not seem to look much beyond the construction and testing of knowledge-base systems is natural enough. Some of the systems were avowedly experimental. Others that were tested (notably medical diagnosis systems of the Stanford Heuristic Programming Project, e.g., Buchanan and Shortliffe, 1984) generated enough information bearing on revisions of the systems, or related matters such as tutorial and explanatory facilities (Clancey, 1979), that there may have been little reason to look ahead. The notion of maintaining a developed system was considered, though, from time to time, for example in an article by Randall Davis (1982, p. 10).

Perhaps it is a measure of the youth of the expert systems field—or the short time that functioning systems have been in use—that today the maintenance of the systems receives little discussion. For example, little or no mention is made of this issue in several recent general works: Harmon and King, 1985; Jackson, 1986; and Waterman, 1986.

Even so, I believe that it is important to extend the business of the knowledge engineer, or whatever we may choose to call this postion, beyond formulating a system and testing it. To be sure, some expert systems refer to small areas of external phenomena, where there will be no discoveries or new insights, and no environmental instabilities to affect what goes on. In these cases, revising the system is pointless.

Other systems may be affected by new discoveries, though, or may need to adjust to certain changes in the phenomena they deal with. For example, a medical diagnosis system would need to be changed as new treatments were developed, or as new ailments occurred in the population. In these cases, unless the machine-resident knowledge systems are themselves sensitive to the new developments, or can come to have verifiable insights about new ways of handling problems, humans will need to update the systems. In fact, even self-modifying systems may need careful attention, including cross-validation from human efforts, until we understand them much better than we are likely to for some time, or unless their domain of self-modification is very restricted and predictable.

In thinking about how far the knowledge engineer's concern should extend, *life-span engineering* as a general concept is suggestive. There are several senses in which an engineer in traditional fields would be concerned with life-span phenomena. One is the very common concern for failure rate of components and assemblies at various stages of use. The fitting of curves showing expected infant mortality and length of service of components is a basic operation. Another concern for life-cycle is in matching the expected longevity of components, given expected conditions of interaction and wear, and cost considerations, in designing a given device or material system.

The above are both predictive concerns for life-span events of a system, where the engineer plans ahead for expected longevity. In other cases, engineers and other technical persons work with the systems past the design stage. (Different engineers may be involved at different stages.) There may be one or more stages of testing and certification, then technical supervision and quality checking during manufacture, and (perhaps less frequently overall) analysis of device wear, and malfunction after use. At some point, engineered devices, structures, and systems may attract the attention of technical persons when the devices (etc.) "die" and need to be disposed of, scavenged for parts, secured from polluting or defacing the environment, and so on. Engineering attention to terminal stages is perhaps least likely, for various reasons.

Devices of continuing value to the people who instigated their design and manufacture are more likely to be attended past the design and testing phases. Devices that are themselves valuable because of being used to make other devices (machines in manufacturing plants), or that are for some reason valuable and owned by someone who can afford maintenance (vintage sports cars, waterworks, or missile systems) will also receive attention. Consumer-owned devices will be variously maintained.

The intelligent machine systems of a Space Station or a Lunar Base are important candidates for continuing maintenance, if not improvement, not so much because they are owned by someone who can afford the repairs (assuming that appropriations come along) but because devices and systems that are valuable depend on them. Certainly not every system in extra-terrestrial locales will have intelligent automated systems associated with them, but it appears that many will.

It is something of a paradox that though these systems will need to be rugged over many conditions of use and many of them must be prepared to function fairly autonomously, some will almost certainly need some on-site tending and most should be open to revision after use or as goals of the facility change. Human specialists in automated knowledge systems will perform at least some of these tasks.

Knowledge systems differ in some respects from the object systems that they may model—for example, an object system may corrode but a knowledge system might just get out of date—but relative to upkeep they may be similar. A classification scheme for the maintenance of knowledge-base systems could well parallel that for devices. It would include:

452

> *user level maintenance,* including routine checks, flagging items for annotation or eventual deeper augmentation, and so on.

> *shop level maintenance,* performed by knowledge-system specialists, involving a somewhat deeper level of diagnosis, and the installation or removal of subject-matter modules but not major control arrangements.

> *depot or even factory maintenance,* where extensive or deep changes to a system are made and tested.

Modular Knowledge Systems

In planning the expert systems of tomorrow, we can think ahead to desirable forms for the systems, and experiment with various design features that we believe to be important. In doing so, I believe that it is important to look at likely external requirements of the systems and to think of ways to satisfy these, rather than just working to develop the systems of today, which may have been prompted by different design assumptions.

Leanness of resource usage and *effective interfacing with object machines and human users* are characteristics that virtually all Space-resident systems should have. Together with *adjustive autonomy* these are important design goals for systems used in remote bases.

Because of the density of informational requirements and the changing nature of tasks, Space Stations of the next decades will need to have *knowledge-base systems that are modular* in certain senses. First, some will have to be replaceable, as new object-systems are added to or removed from the Station. These replaceable modules will need in most cases to be integrated at least for support purposes with other knowledge-referent facilities. A second kind of modularity would apply as well to the structure of standing systems. Here, facilities would be cross-referenced to make effective use of basic semantic and procedural resources, and enter into priority systems for effectance and display.

I've discussed one view of modular knowledge systems, whose real problem is not their modularity but their integration with other modules and basic systems processing, in a technical memo (Hays, 1986). The topic is one that should receive increasing attention as the demands of densely packed action environments such as Space Stations are closer at hand.

Working with Others

The knowledge engineer's role has from the start involved both epistemic and social processes. Knowledge is obtained, observed, shaped, formed into rules and descriptions, refined. Much of this process involves interacting with others: mainly the experts, but increasingly, as systems are designed for non-technical persons, with the users.

A person who is charged with developing an intelligent system for a complex project may have to interact with a number of people, not all of whom will be agreeable to all suggestions. The complexity of organizational interaction should

increase as the need to integrate component systems from diverse sources is more pressing.

To some extent, these problems are the familiar ones of working in bureaucratic organizations with persons who have mixed motives, desires for self-protection and self-enhancement, and so on. But they are not trivial.

I would argue for a very central role for those concerned with knowledge system development, because of the importance of these systems in the action organizations of the future. Whether or not this centrality comes to exist, persons who see to the development and integration of knowledge systems will have to have not only face-to-face skills but also good understanding of how human organizations work.

For example, one problem area for knowledge systems management is in incorporating information from design and testing phases of object systems. Advice and information from designers and testers can be voluminous, and can trace many design and implementation revisions. It will reflect both the close knowledge of the designers about what they've conceived and worked out, but also their blind spots owing to inadequate concepts of conditions of use or egoism about the ruggedness of their creations in unpredictable interactions with other systems. It will ordinarily cross organizational boundaries, so that protectiveness of ideas and concern for liability in case something should go wrong will be factors.

Integrating diverse systems always brings problems. It is something that people do not do well, in general. They are apt to, for example, over-ritualize the process with numerical techniques that are not quite relevant and associate the process with prestigious organizational positions that increase managerial overhead without corresponding increases in effectiveness of actual functioning. The integration of systems is just what the knowledge engineer/manager of the next decades will be strongly concerned with. Two suggestions are offered here. One is to study peculiarities of the integrations of knowledge systems with one another and with object systems, so that the processes may be better understood. *Probably at least as much should be invested in studying interfaces and interactions of system components as in studying individual knowledge system characteristics.* The second suggestion is to keep the work of integrating the systems primarily a technical matter and out of the arena of nominal organizational responsibility.

Training for Knowledge Systems Management

This issue can only be mentioned briefly here.

If jobs with new combinations of skills and desirable background knowledge come into existence, it may be efficient to provide academic training programs for them. This may be so, even if the job roles are in something of a state of flux and evolution, since academic programs can provide useful background knowledge and not just specific job information.

454

The academic background for automated-knowledge workers of the next years should certainly cross departmental boundaries of the universities. All these jobs require facility in

> social and personal skills.
> knowledge processing skills of several sorts.
> familiarity with technical details of symbolic programming, and possibly with the nature of the related object devices and systems (including social and organizational systems that might be the subject of automated advice or coordination in some particulars).

Depending on the particular goals of an individual, coursework might be drawn from fields such as computer and information science, psychology, philosophy, general arts and science background (probably the most important kind of academic preparation), organizational and management studies, and particular subject-matter areas.

One group of people who could probably benefit now from specially designed academic programs are people already working at a professional level in agencies and companies, who are specialized in subject matter fields, but who could profit from training, probably at the Master's level, in knowledge system development. Such training should focus on knowledge processing first, I would argue, then on technical familiarization with matters such as programming systems. These people are often involved in developing or monitoring the development of intelligent systems in companies where there is little expertise or experience in either artificial intelligence or more abstract knowledge management.

As time goes on, training at other levels may be developed, for example, undergraduate training for "knowledge technicians" and special higher level training for integrative knowlege management. (Again, I would recommend a predominance of arts and science training rather than technical training for these people. This is an important point, that may be disputed by persons in technical departments.)

Another trend we can expect is the eventual incorporation into technical fields of course content relating to epistemic systems. This should come about naturally as effectance systems or devices become more closely linked to knowledge-base processing,

References

Buchanan, B. G. and Shortliffe, E. H. *Rule-based Expert Systems: the MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1984.

Clancey, W. J. "Tutoring Rules for Guiding a Case Method Dialogue," The *International Journal of Man-Machine Studies,* 1979, 11, 25-49.

Davis, R. "Expert Systems: Where Are We? And Where Do We Go From Here?" *AI Magazine,* Vol.3 No. 2, 1982, 3-22.

Harmon, P. and King, D. *Expert Systems.* Wiley, 1985.

Hayes-Roth, F., Waterman, D. and Lenat, D. (Eds.) *Building Expert Systems.* Addison-Wesley, 1983.

Hays, D. "Modular Automated Knowledge Systems." Technical Memo, Cognitive Systems Laboratory, U. Alabama in Huntsville, 1986.

Jackson, P. *Introduction to Expert Systems.* Addison-Wesley, 1986.

Nilsson, N. J. "Artificial Intelligence: Engineering, Science or Slogan?" *AI Magazine,* Vol. 3 No. 1, 1981-82, 2-8.

A Space Station Onboard Scheduling Assistant

A.F. Brindle and B.H. Anderson
Boeing Aerospace Company
P.O. Box 3999 MS 82-58
Seattle, WA 98124

Abstract

One of the goals for the Space Station is to achieve greater autonomy, and have less reliance on ground commanding than previous space missions. This means that the crew will have to take an active role in scheduling and rescheduling their activities onboard, perhaps working from preliminary schedules generated on the ground. Scheduling is a time-intensive task, whether performed manually or automatically, so the best approach to solving onboard scheduling problems may involve crew members working with an interactive software scheduling package. This report describes a project to investigate such a system, which uses knowledge-based techniques for the rescheduling of experiments within the Materials Technology Laboratory of the Space Station. Particular attention is paid to 1) methods for rapid response rescheduling to accommodate unplanned changes in resource availability, 2) the nature of the interface to the crew, 3) the representation of the many types of data within the knowledge base: crew, resources such as power, experiments, schedules, and constraints, and 4) the possibility of applying rule-based and constraint-based reasoning methods to onboard activity scheduling.

1. Introduction

This paper presents the preliminary design of an Onboard Scheduling Assistant (OSA) for the Space Station. A more detailed description of the issues involved and the existing demonstration system may be found in [1].

The Space Station activity scheduling problem has a number of interesting characteristics. A major feature is that scheduling of crew activities and other autonomous onboard tasks will occur both on the ground and onboard. A full schedule including orbit maneuvers, housekeeping and maintenance tasks, and payload experiments will be developed on the ground and periodically transmitted to the Station, where it may undergo some modifications. This implies that a data format for scheduling will be shared between ground and station; in fact, it will be advantageous for the two to have a consistent knowledge representation scheme, as described below. Therefore, the ground based scheduling problem is described briefly, the onboard rescheduling problem is described, and a constraint based representation is proposed as a suitable data organization approach for both ground and Station scheduling. A description is then given of a prototype onboard rescheduling tool which uses a constraint based knowledge base to implement several limited rescheduling algorithms.

Ground based planning and scheduling will have the goal of a highly optimized, detailed schedule. A large variety of constraints will be involved in the scheduling, such as precedence constraints on activities, hard timing constraints, and resource usage constraints.

Multiple, and scarce, resources will be allocated as part of the scheduling. These resources may be logistics elements, such as laboratory equipment, which are allocated in fixed units and not consumed. They may be consumables, such as liquid fuel, or generated consumables which may be stored, such as electrical power. Re-

sources may be accompanied by a matrix of attributes, for example, crew members with associated skills. Resources may even be created by execution of tasks within the schedule, as is the case with recycled water. The algorithms used to aid in scheduling are often determined by the nature of the resources involved. For example, a bin packing approach [5] to electrical power load management is not feasible, because with a generated, stored resource, the amount of resource available at any time is not independent of the schedule. The amount of resource available at time t depends upon the amount used by tasks scheduled for a period before t.

The nature of task requirements for various resources also influences the applicability of scheduling algorithms. Individual Space Station activities have resource needs that vary over the duration of the activity, and in some cases, such as power, the resource is so tightly constrained and fully utilized that constant approximations for task resource requirements may be undesirable.

Existing scheduling algorithms can manage some portions of the ground based scheduling. These include algorithms arising from project scheduling (e.g. CPM and Pert [8]), job shop scheduling (see Coffman [2]), and especially project scheduling over multiple resources (see the surveys of Davis [3] and Herroelen [6]). However, this optimizing scheduling would benefit from a broad, flexible knowledge base, which would permit the representation of the diverse constraints and resources, and more heuristic data for a wider variety of scheduling algorithms. It would also allow the multiple scheduling efforts now performed on the ground (using tools such as CAPS [10] and the system of Jaap [7]) to be better coordinated.

Onboard Space Station scheduling will consist of rescheduling in response to changes in the operating environment, changes such as unanticipated reductions in resource availability. Such rescheduling will be deliberately limited in scope, on the assumption that the time and computing resources available for rescheduling onboard will be strictly limited, precluding a full scheduling effort. More important, it is assumed that the scheduling data available onboard will be a subset of that employed during ground based optimizing scheduling. This leads to the conclusion that making gross alterations to the existing schedule onboard would probably create more problems than it would solve, as constraints that are not understood onboard would be violated.

The questions which arise, then, in the creation of an onboard rescheduling tool, are 1) what simple alterations to an existing, highly optimized schedule will best respond to the changing onboard environment without violating presumed constraints on timing, resource availability and precedences, and 2) what types of knowledge must be represented onboard for such rescheduling. To these can be added 3) what simple alterations to the schedule could enhance crew job satisfaction by giving them control over their daily activities (again without violating constraints).

It turns out that minimum perturbation rescheduling in some cases involves the manipulation of constraints and allowable alternatives that have not been represented explicitly up until now. For example, if a task requires multiple resources including a crew member with a particular skills matrix, and that person becomes unavailable, then the substitution of another available crew member with similar skills is desirable. The rescheduling of the activity to another time is not a good option, since that might upset the use of the other resources or violate other constraints.

The constraint based knowledge representation of Fox [4,9] for job shop and project scheduling permits the building of a knowledge base for both ground and Station scheduling. It also provides a vocabulary for the description of the scheduling problem. Tasks, resources, and schedules are objects in the representation. Limitations that define a valid schedule are represented explicitly as constraints, including task requirements for resources. Since a major part of most scheduling efforts involve deciding how to make do when all of the constraints cannot be met, each constraint may be associated with relaxations, which describe alternate, possibly less desirable, constraints for consideration. Each relaxation has an associated utility, or desirability metric, and the constraints themselves may be

458

rated as to which ones should be relaxed first in the search for a reasonable schedule.

The suitability of a constraint based representation for ground scheduling is only hypothesized here. However, the development of the Onboard Scheduling Assistant illustrates its utility for onboard processing of schedules, resource allocations, resource attributes, and resource requirements.

2. The Onboard Scheduling Assistant

The OSA is a demonstration system written in Zetalisp and running on Symbolics machines. It employs a menu driven, graphical interface to allow the user (supposedly a crew member) to view scheduling information along many different lines. Displays include all tasks in the timeline of one schedule (see Figure 1), a task's requirement for a resource plotted over time (Figure 2), and the total use of one resource by one schedule plotted over time (Figure 3). As many as four of these displays may be viewed simultaneously (Figure 4). The user may edit resource availability, reschedule individual tasks in a schedule, request a summary of all points at which any resource has been overallocated by a schedule, and request that new schedules be created by any of several simple, fast rescheduling algorithms. The ability of the crew to amend their own availabilities and to make small movements of tasks in time, as well as the "what if" capability resulting from these features, should increase crew acceptance of the schedule and the scheduling process.

The knowledge base is object oriented, with explicit treatment of constraints. Currently constraints are limited to task requirements for resources or resource attributes. Relaxations on constraints, with their utilities, are permitted in the form of alternative resources, alternative attributes, or requirements for any resource within a set. Resources may be either generated consumables or logistics items, and may have discrete attributes.

Other object types included are schedules, re-

source allocations within schedules, resource utilization summaries, resource overutilization summaries, task types, meta-task types, and tasks. Meta task definitions allow individual activities, such as steps in an experiment, to be joined into large, goal oriented procedures. These definitions can be hierarchical. A task is an instantiation of task or meta-task type, and thus may have a number of sub-tasks. The sub-tasks are assumed to be independently schedulable, subject to constraints, but decisions on whether to add or delete an activity are made relative to the entire task only.

Each task has a ground assigned priority, a static number indicating its original importance in the health of the station and the achievement of payload goals. Each also has crew assigned priority, which allows the crew to reassess criticality of tasks, if necessary. The knowledge base has the capability to represent time passing, so that the start and end times of tasks can be compared to the "current" time. These three features are employed in the computation of task priorities during scheduling.

Some of the knowledge in a constraint based scheduling representation, such as notions of state, causality, and revision, are required for full optimizing scheduling, but have been postponed in the implementation of the OSA. Nevertheless, the information available in the OSA permits the following rescheduling approaches:

- Resource Substitution. The summaries of resource overallocation are analyzed to determine which tasks are involved in problem areas. Constraints are not ranked in the system, so it is necessary to decide on an order for examining them. Therefore, the problem tasks are ranked by dynamic priority. This is the weighted sum of terms which reflects the importance of 1) the ground assigned priority, 2) the crew assigned priority, 3) whether or not a task has already begun, and 4) how much a task is contributing to the problem areas as a whole. This last is measured as the proportion of use by the task averaged over all problem areas. It reflects the general goal of keeping as many tasks as possible

459

in the schedule; many other goals, priority heuristics, and methods of measuring terms are possible.

Once the tasks are ranked, processing proceeds from lowest ranking to highest. The constraints (here, resource requirements) of the task are considered in random order. For each requirement for a resource within a problem area, an effort is made to replace the allocation of that resource to the task by satisfying a relaxed requirement which utilizes another, available resource. This is done once for all task requirements which are pertinent to the problem areas, resolving as many problems as possible.

- Task Deletion. All tasks involved in problem areas of resource utilization are ranked by dynamic priority, as described under Resource Substitution. Then the problem areas are processed in random order, and the lowest priority tasks involved in each problem are deleted from the schedule until all problems of overutilization are resolved. This is intended only for tasks that are known by the crew to be involved in very few constraints which are not represented in the system, since deletion of a task can easily lead to violations of precedence constraints.

- Task Insertion. One task not currently scheduled is selected by the user. An attempt is made to schedule the task, without creating any problems in resource usage, and without moving any scheduled tasks in time. This is intended only for tasks that are known by the crew to be involved in very few constraints which are not represented in the system, since addition of a task can easily lead to violations of resource use.

These reschedulers could be combined into a full backtracking scheduler, but it would be far too slow in its exhaustive search of a combinatorially large space. More realistically, all three could be used as routines within ground based scheduling which made extensive use of search-limiting constraints and other heuristics. Onboard, it seems preferable to provide an automatic scheduling

option which attempts resource substitution initially and then falls back upon task deletion, but which avoids a more comprehensive search for combinations of relaxations or reassignments of start times.

3. Conclusions

The OSA is a running demonstration which illustrates the viability of constraint based representation and limited heuristic based rescheduling for the onboard Space Station scheduling problem. The investigation into the onboard scheduling environment has emphasized the need for consistency between ground and Station scheduling representations. Furthermore, it has revealed that advantages could be gained for crew satisfaction and adaptive response to environment changes if the polished schedule is transmitted to the Station along with a small amount of the knowledge underlying it, such as resource requirements and relaxations. Additionally, it has provided a mechanism for some experimentation with user interfaces for the display of the very complex, multidimensional body of knowledge that is required for scheduling.

Much more work is needed on the full representation of knowledge for ground scheduling, along with the acquisition and analysis of heuristics for optimizing scheduling as it is currently performed for manned space missions.

References

[1] A.F. Brindle and B.H. Anderson. *A Space Station Onboard Scheduling Assistant Project Report.* Technical Report D180-29768-1, Boeing Aerospace Company, 1986.

[2] E.G. Coffman, editor. *Computer and Job-Shop Scheduling Theory.* J. Wiley and Sons, 1976.

[3] E.W. Davis. Project scheduling under resource constraints – historical review and categorization of procedures. *AIIE Transactions,* 5(4):297–313, 1973.

[4] M.S. Fox. *Constraint-Directed Search: A Case Study of Job-Shop Scheduling.* Technical Report CMU-RI-TR-83-22 (Ph.D. Thesis), Carnegie-Mellon University Robotics Institute, 1983.

[5] M.R. Garey, R.L. Graham, and D.S. Johnson. Resource constrained scheduling as generalized bin packing. *Journal of Combinatorial Theory (A)*, 21:257–298, 1976.

[6] W.S. Herroelen. Resource-constrained project scheduling – the state of the art. *Operational Research Quarterly*, 23(3):261–275, 1972.

[7] J. Jaap and E. Davis. Experiment scheduling for spacelab missions. In *The Conference on Artificial Intelligence for Space Applications 1986*, co-published in these proceedings.

[8] J.J. Moder and C.R. Phillips. *Project Management with CPM and Pert.* Van Nostrand Reinhold, New York, NY, second edition, 1970.

[9] A. Sathi and M.S. Fox. Representation of activity knowledge for project management. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-7(5):531–552, 1985.

[10] K.L. Scott. *Crew Activity Planning System II Users Guide.* McDonnell Douglas Astronautics Company, for NASA Johnson Space Center, February 1986.
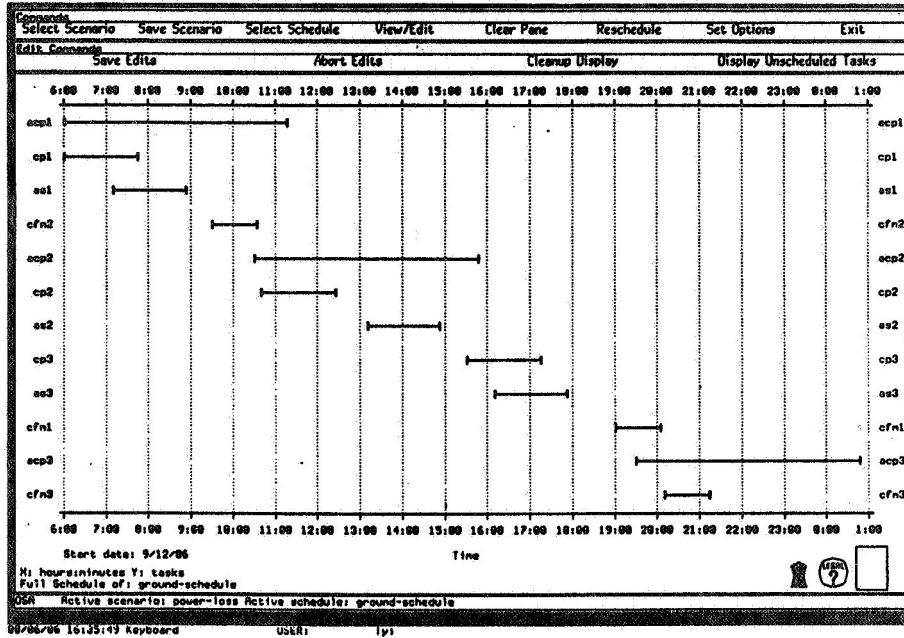
Figure 1: The Tasks in a Materials Technology Laboratory Schedule.
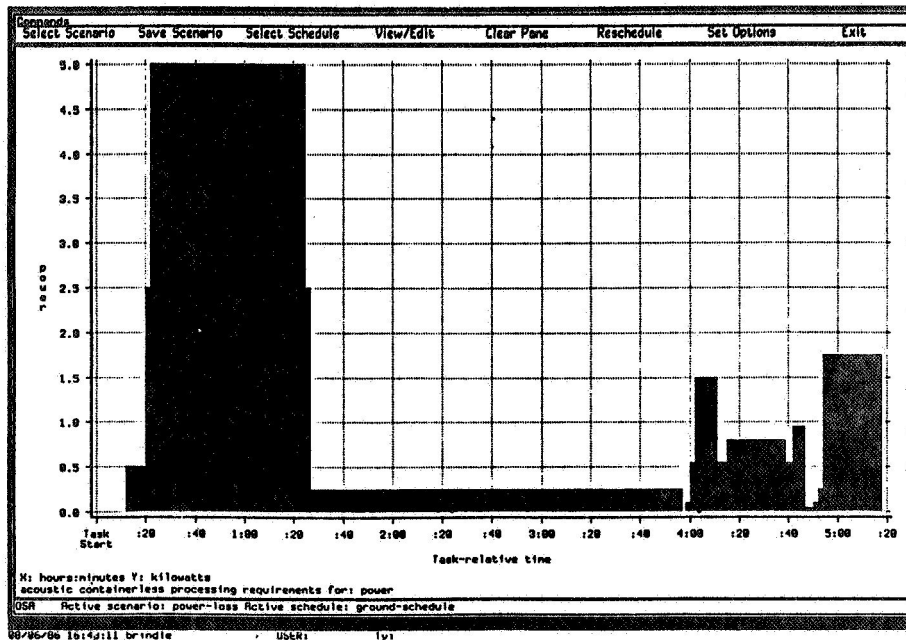
Figure 2: The Requirement of an Acoustic Containerless Processing Experiment for Power.

462
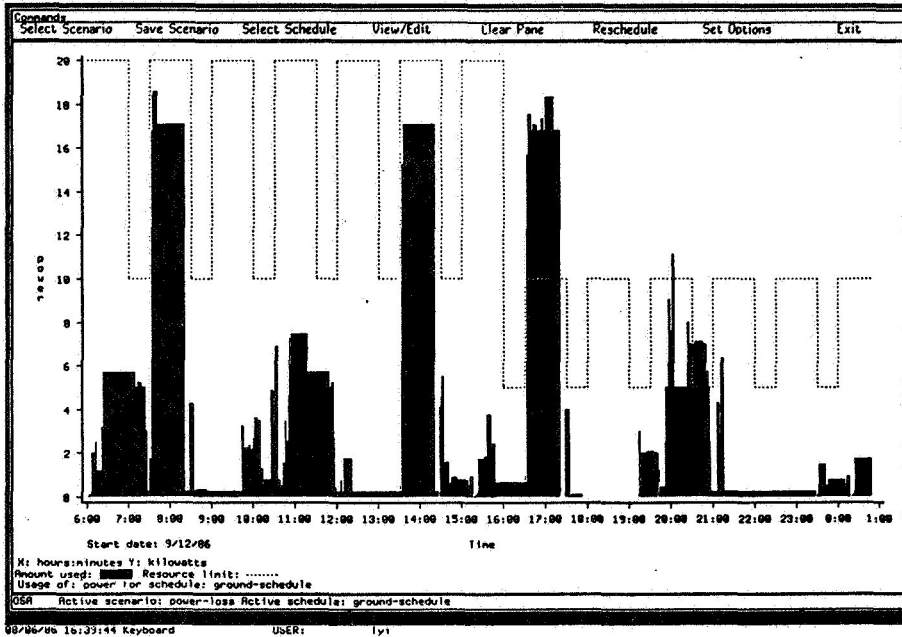
Figure 3: The Summary of Power Usage for the Schedule in Figure 1.
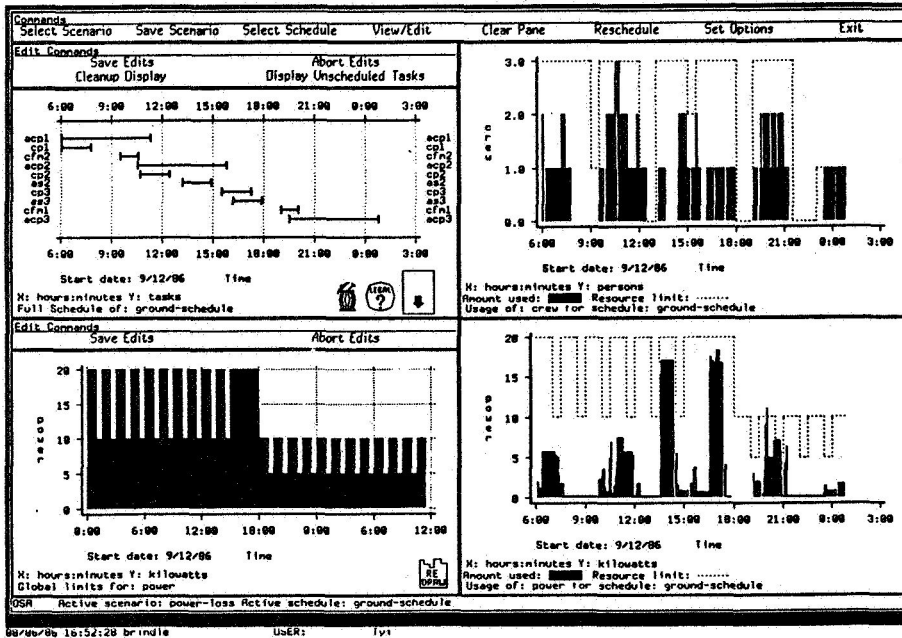
Figure 4: Four Displays as Selected by the User.

NOA
A Network Operator Assistant for Scheduling TDRSS

ABSTRACT

Network Operator Assistant (NOA) is a prototype expert system developed at Computer Sciences Corporation (CSC). NOA uses detailed scheduling knowledge and problem solving heuristics to assist Network Control Center (NCC) operators schedule the NASA Space Network (SN) in time critical situations. This paper presents the current status of NOA and its future directions.

NOA was developed on a Xerox 1108 in LOOPS, a multiple paradigm programming environment that integrates object-oriented, access-oriented and rule-based programming in Interlisp-D. NOA can reschedule network users and critique the current scheduling status.

NOA has proven successful in the 'Active Period' of scheduling which is crisis intensive and more dynamic in nature than the initial schedule generation period. In the Active Period addition, deletion, and change of requests are made for which NOA performs heuristic scheduling such that disruption of the existing schedule is minimal.

NOA has a powerful user interface. The entire schedule can be sequenced in a time-line and can be displayed in various ways using multiple windows and menu driven functions. This aids the operator in looking at the schedule from different perspectives, that is, from the point of view of the users or of resources, individually or in any combination.

Currently NOA is being tested in a scenario where 300 requests per day from 20 user satellites are scheduled along with 50 percent changes and 20 percent new requests being added during the active period. In such a complex scheduling situation where many variables are involved, reliance on human experts may not always be appropriate. Human decisions are local, whereas computers do not have problems dealing with many variables and can provide global strategies. NOA provides global strategies, by providing an optimal mix of traditional computation in scheduling, such as linear programming, and heuristic rules.

NOA's knowledge base has several layers of abstraction providing better understanding of the problem domain. Any level of the knowledge base can be accessed whenever required. Currently, this knowledge base is being deepened by adding a layer of user specific knowledge. This will allow NOA to do opportunity scheduling, assess impacts of either unusual or crisis events, and evaluate all scheduled and unscheduled activities.

TYPE & TOPIC

Type: Unclassified

Topic: Space and Satellite Applications

AUTHORS
Bikas K. Das
  Computer Sciences Corporation
  8728 Colesville Road
  Silver Spring, MD   20910

Richard A. Berg
  Computer Sciences Corporation
  8728 Colesville Road
  Silver Spring, MD   20910

Terry Janssen
  Computer Sciences Corporation
  8229 Boone Blvd.
  Vienna, VA   22180

INTRODUCTION

NOA is designed to assist Network Control Center (NCC) Operators schedule Tracking and Data Relay Satellite System (TDRSS) resources at NASA's Goddard Space Flight Center. The development of NOA was spurred by the increasing complexity and demanding requirements of the NCC's scheduling. A number of expert systems[9,10,11,12,13] have been developed in the area of scheduling. Although quite successful in their respective domains, many of these systems are limited in one way or another. Usually they are slow and often do not address dynamic environments. Hence, real time application is not possible. Often they employ naive heuristics and must rely heavily on human experts for application of results. Some of them operate on a shallow knowledge level. These types of limitations have been apparent in the construction of NOA and are discussed later in the paper.

It has been argued[1] that to get a reliable program to solve a problem one must study the problem, not the way people say they solve it. This principle has been used extensively in the development of NOA. Individual problem solving methods, such as heuristic rules, frames or procedures are important in the development of an expert system. However, knowledge based reasoning at a higher level of abstraction is also useful, especially for a complex system. This approach is discussed in the literature[8] and has been adopted in NOA. It is helpful not only in capturing the essence of the task but also in organizing both knowledge and problem solving behavior for more focussed problem solving.

TASK DOMAIN

The NCC schedules and monitors the NASA Space Network (SN) resources[4], primarily TDRSS. Once a schedule is generated during the schedule generation period, it can be easily changed up to a short time prior to the use of the schedule information (the active period). The basic objectives of schedule generation are:

1) Provide increased service to the user by increasing his probability of being scheduled.
2) Reduce the need for the NCC operator's manual intervention through increased automation.
3) Use the NCC experience base in developing a flexible, effective scheduling activity.

The load of requests are rapidly increasing along with the number of participating users. In 1987, 300 requests per day are anticipated during the schedule generation period along with 50 percent changes and 20 percent new requests being added during the active period.

Due to the increasing volume of requests and the flexibility needed at different levels of scheduling, the following scheduling requirements are emerging :

467

O A more dynamic environment requiring the ability to generate alternative support, to reschedule by priority, and to evaluate scheduled activities.

O The ability to handle a crisis or emergency situation that needs opportunity scheduling, identification of critical requests, handling of scarce resources, and impact assessment.

To incorporate these scenarios, CSC has developed a methodology for efficient optimization of a number of requests for single resource scheduling and its heuristic extension to multiple resources.[5,6,7] Such deterministic optimization using linear programming methods is acceptable during schedule generation but does not fully address 'active period' scenarios. Furthermore, traditional software to automate the procedures in the 'active period' become unwieldy and inefficient where judgment and heuristics are an integral part of the decision-making. NOA has been developed to address this aspect of scheduling.

DESIGN & IMPLEMENTATION
NOA is developed in Loops[2], a multiple paradigm programming environment that integrates object oriented, access-oriented and rule-based programming in Interlisp-D and is running on a Xerox 1108. The rich programming environment of Loops for building with different knowledge representations and problem solving schemes is described elsewhere[12], and will not be further discussed here.

The primary resources to be scheduled are TDRSS antennas. Each user normally has several requested activities. Each activity, called a mission, needs to be scheduled. Each TDRSS has two single access (SA) antennas and one multiple access antenna (MA). A mission has information regarding the view periods of all the TDRSS's and the request for the antenna such as SA, MA, or both. A user may or may not be flexible about choices. LOOPS has been used to represent three basic objects; the user, the mission, and the antenna.

Figure 1 symbolizes an unscheduled activity while Figure 2 indicates a scheduled one. The dotted line in Figure 2 implies an indirect connectivity whenever a direct connectivity exists between an antenna and a mission. This representation allows flexibility in dynamically manipulating the objects when triggered by an external method (a method is the procedure that sends a message to an object in Loops) . NOA's primary knowledge base is built in this framework and consists of scheduled and unscheduled missions obtained from a batch optimization procedure.

NOA has a powerful user interface and uses multiple windows and pop up menus. The entire schedule can be sequenced in a time-line and can be displayed in various ways using multiple windows and menu driven functions. This aids the operator in looking at the schedule from different perspectives, that is, from the point

of view of the users or of resources, individually or in any combination. Moreover, this allows the operator to critique the current optimization and use methods to further optimize the available slots, if possible, in an incremental fashion.

Figure 3 illustrates another layer of the knowledge base consisting of rules. The rules are organized and connected to the object space of the primary knowledge base and are derived from general scheduling principles used during the Active Period. In the event of additional requests and/or changes in requests, NOA will try to find an alternate schedule for any changes requested.

Scheduling a new mission is usually done in one of the following three ways. First look for an open slot that is within mission requirements. If a slot is not available, try to shuffle the new mission with already existing missions, if possible, and then de-assign and assign missions to fit the new mission in. If the second approach fails, assign the new mission by a priority scheme and bump the appropriate lower priority missions. These rules are in turn be used to critique the current schedule and further optimize it.

A model of resource constraints is also being built. Certain resources in the Space Network, separate from the TDRSS, are in high demand but are scarce. These resources represent a constraint class in the constraint knowledge base. Other resources will likely break down, need repair, and remain under repair for certain periods of time. They represent a second class of constraints. This constraint information will be used in the process of resolving conflicts when the existing schedule becomes invalid. Consider the following example.

If a resource breaks down, appropriate changes are made in the constraint knowledge base. The effect of the breakage is then determined using the constraint knowledge base. The affected missions during the period of repair are rescheduled to other time periods or other resources. The knowledge of constraints is a part of the third knowledge base.

AI ISSUES
Since multiple paradigms were used in the implementation of NOA, it is not possible to give a quantitative estimate of size of the knowledge base by counting the number of rules alone. The model prototype currently has more than 200 objects, 15 rule-sets containing approximately 75 rules, and 10 active values. The model derives knowledge by representing its objects in a multiple inheritance lattice, by triggering other computation due to access oriented programming, and by the way the rules are structured in this multiple paradigm environment.

Situations exist in this problem domain where a single mission is scheduled to multiple antennas for different time durations. Also in theory an antenna may be accessed at different band widths at the same time by more than one mission. The notion of

composite objects[3] can be applied in these situations. In the current model a set of antennas can be suitably combined, where an antenna itself has parts such as bandwidth. This entity can be defined as a composite resource. This was experimentally tried in NOA. However, although the use of composite objects allows the automatic initialization of all the parts of the composite object, no additional advantage was found using these objects in NOA.

NOA is adaptable and enhancable. Unlike traditional software it can easily evolve to conform with the working environment. For example, rules are easily modified to reassign or de-assign a mission on multiple antennas. Rules are easily changed to consider available time periods on both sides of a schedule period. Unlike other systems in scheduling, NOA's knowledge base has several layers of abstractions. This provides better understanding of the problem domain. Furthermore, one can be confined to shallow levels if it is not important to probe the deeper levels of knowledge. Thus all levels of knowledge bases are independent, but connections can be made whenever required.

NOA employs common sense reasoning. For example, while rescheduling a mission, it is common sense to look into other activities only at the time of the mission's view-periods. Thus search procedures guided by a time window are used, an approach that is not only computationally efficient but also very natural to the problem being solved. This common sense reasoning is most useful in organizing the major tasks of an application rather than within the tasks themselves.

In a complex scheduling situation where many variables are involved, reliance on human experts is not always appropriate. Human decisions are local, whereas computers do not have problems dealing with many variables and can provide global strategies. NOA provides global strategies by an optimal mix of traditional computation in scheduling, such as linear programming, and the use of heuristic rules. One example is that the results of the traditional optimization are represented in the primary knowledge base of NOA (see figure 3). Depending on the nature and number of changes in the request, NOA will decide the objectives of further optimization using traditional procedures if needed and subsequently will update the primary knowledge base.

CONCLUSION
The NOA project is being used to explore AI technology at CSC. An extensive search was made for an appropriate problem domain. The one identified was based on its need in the real world and available expertise within CSC. Initial success with NOA has led to the development of the following improvements.

NOA's knowledge base is currently being expanded. A planned deep model of this scheduling domain will enhance its scheduling ability and provide more insight in any particular problem. This will be achieved by incorporating a layer of knowledge containing both user specific knowledge (knowledge about user satellites,

their interrelations with TDRSS resources, orbit characteristics and so on) and also the constraint knowledge below the knowledge base of general scheduling principles. In this context, a framework will be built that will allow NOA to do opportunity scheduling, asses impact either due to any unusual event or crisis and evaluate all scheduled and unscheduled activities. Because of NOA's architecture and design approach, its potential as a scheduler's assistant in a complex and dynamic scheduling environment is promising.

REFERENCES

1) Denning, Peter J. (1986), Toward a Science of Expert Systems. *IEEE Expert*, Vol. 1, No. 2.

2) Stefik, M., et al (1983) Knowledge Programming in LOOPS. *AI Magazine*. Vol. 3, No. 3.

3) Stefik, M. and Bobrow, D. G. (1986) Object-Oriented Programming: Themes and Variations. *AI Magazine*. Vol. 6, No. 6.

4) Goddard Space Flight Center (1984) *Network Control Center Full Operational Capability Detailed Requirements*.

5) Reddy S. and Brown W.: Private Communication.

6) Long, A. (1985) *Network Control Center (NCC) Schedule Generation Concepts*. CSC Preliminary Report.

7) Reddy S. (1986) *A Methodology for Scheduling TDRSS Resources*. CSC Report.

8) Chandrasekaran, B. (1986) Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design, *IEEE Expert*, Vol. 1 No. 3.

9) Bruno, Georgio, Elia, Antonio and Laface, Pietro (1986) A Rule-Based System to Schedule Production, *Computer*, Vol. 19 No. 7.

10) Fox, M. S. and Smith, S. F. (1984) Isis-A Knowledge Based System for Factory Scheduling, *Expert Systems, the International Journal Knowledge Engineering*, Vol. 1.

11) Omar, Angelo M. (1986) School Time Table Scheduling in Prolog, *SIGART Newsletter*, No. 96.

12) Hennard, Janet (1986) ESRA: An Expert System for Resource Allocation *Proceedings of Robotics and Expert Systems*, pp. 179-188.

13) Gargan Jr., Robert A. and Kovarik Jr., Vincent (1986) An Expert System for Mission Scheduling and Conflict Resolution, *Proceedings of Robotics and Expert Systems*, pp. 189-200.

```
 ---------------                    ---------------
|               |                  |               |
|   U S E R     |------------------|  M I S S I O N |
|               |                  |               |
 ---------------                    ---------------
```

figure 1. Unscheduled Activity

```
                    ---------------
                   |               |
          - - - - -|  A N T E N N A |------
         |         |               |      |
         |          ---------------       |
         |                                |
         |                                |
 ---------------                   ---------------
|               |                 |               |
|   U S E R     |-----------------|  M I S S I O N |
|               |                 |               |
 ---------------                   ---------------
```

figure 2. Scheduled Activity

```
 --------------      ------------------      ---------
| USER'S       |    |                  |    |         |
| REQUESTS     |<-->|    OPERATOR      |<-->| DISPLAY |
|              |    |                  |    |         |
 ------ | -----      --------A---------      ----A----
        |                    |                   |
        |                    |                   |
        |           ------------------------     |
        |           |                       .    |
        V                                        V
 --------------      ------------------      ----V----
|              |    |     PRIMARY      |    |         |
|OPTIMIZATION  |--->|   KNOWLEDGE      |<-->|INFERENCE|
|              |    |  BASE (SCHEDULES)|    |         |
 --------------      ---------A--------     |         |
                             |             |         |
                             V             |         |
                    ----------V---------    |         |
                   | GENERAL KNOWLEDGE |   |         |
            --->|   OF SCHEDULING IN   |<-->|   AND   |
                   | THE ACTIVE PERIOD |   |         |
 --------------     ----------A--------    |         |
|              |---|          |           |         |
| KNOWLEDGE    |   |          V           |         |
| ACQUISITION  |   ----------V---------    |         |
|              |---|                  |   |         |
 --------------     | USER SATELLITE  |   |         |
            --->|   AND CONSTRAINT   |<-->| CONTROL |
                   | KNOWLEDGE        |   |         |
                    ------------------     ---------
```
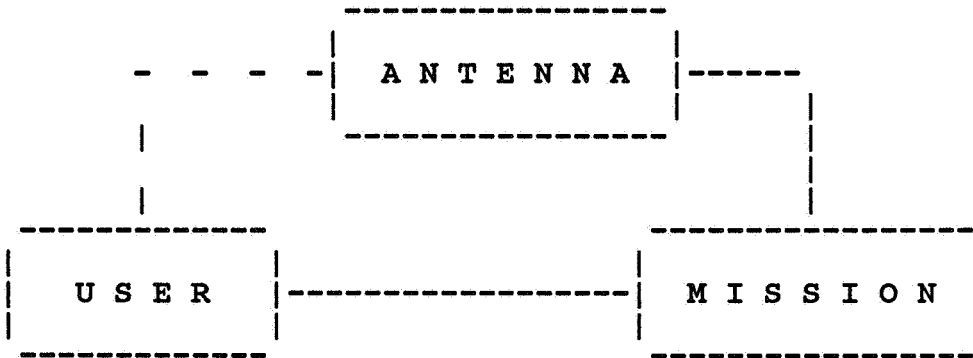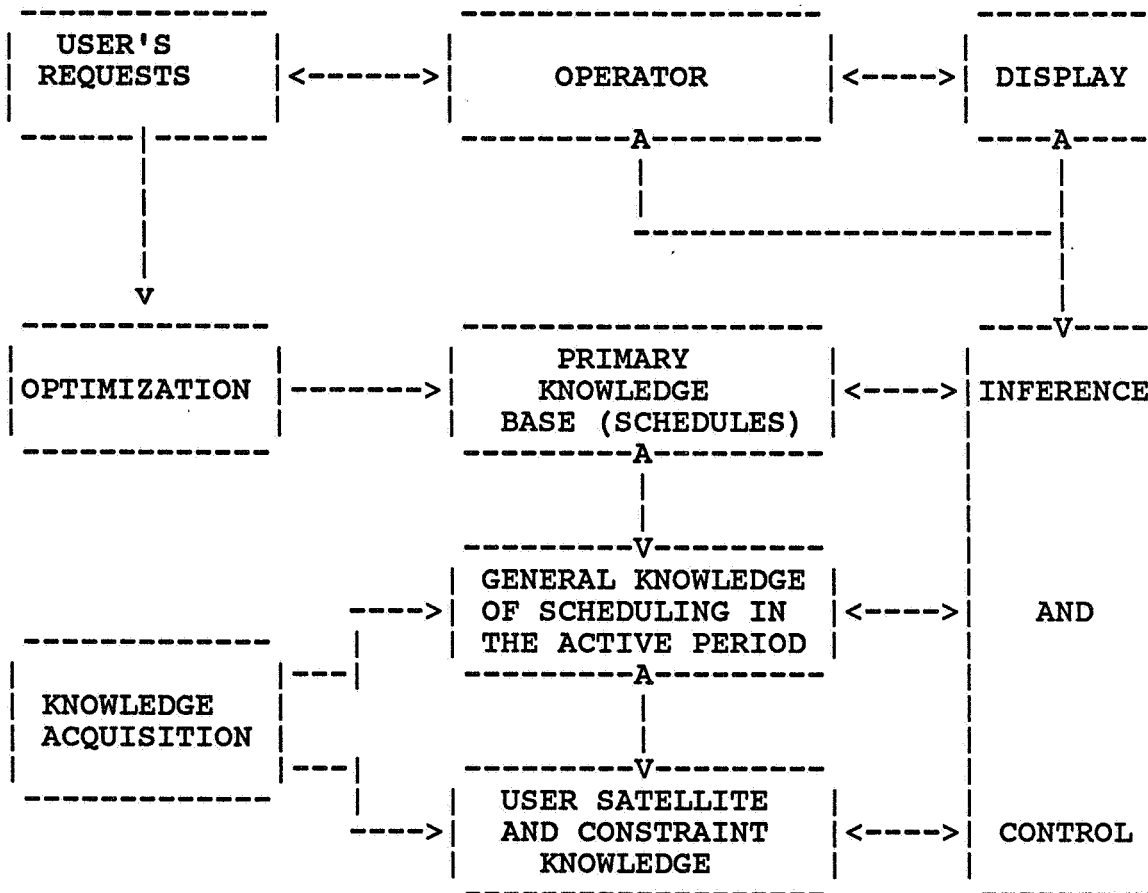
figure 3. NOA Design Showing Layered Data Bases

473

Experiment Scheduling for Spacelab Missions

John Jaap and Elizabeth Davis
Marshall Space Flight Center

INTRODUCTION

The Experiment Scheduling Program (ESP) is the heart of a group of programs developed at the Marshall Space Flight Center to schedule the experiment activities of Spacelab and other Shuttle missions. Other programs in the group either prepare input data for ESP or produce derivative information based on the schedule produced by ESP. The task of experiment scheduling can be simply stated as positioning the experiment activities in a mission so that they collect their desired data without interfering with other activities or violating mission constraints.

Beginning in 1974, the Mission Integration Branch (EL22) at Marshall developed several programs for automated experiment scheduling of Spacelab-type missions on a Univac-1108 and a PDP-11/70. These programs had several major deficiencies: failure to handle enough instances of each type of constraint or sufficient types of constraints, lack of integration into the mission planning system, poor design, lack of extensibility, and failure to meet the emerging need for real-time mission support.

Beginning in 1979, we wrote the requirements for, designed and implemented a versatile program which uses expert-system techniques to generate experiment timelines. ESP became operational in March of 1981. It is well integrated into a group of mission planning programs running on a Digital Equipment Corporation VAX computer. It has supported the experiment scheduling activities for nine Spacelab missions, three of which have been flown, and several partial payloads. The program is being used not only by MSFC, but also by the Flight Project Engineering Office at JSC to schedule the SLS-1 and SLS-2 Spacelab missions. In addition, ESP is being used to schedule 90-day straw-man mission segments for Space Station investigations. There are approximately twenty-five people who are proficient in using the program.

In this paper, we shall describe the program's capabilities as seen by the user, the experiment and mission constraints the program handles, and how the expert system in the program handles these constraints. We have referred to experiments and experiment timelines, assuming that the reader has an intuitive understanding of these concepts. This discussion of ESP can be better understood if we first define several terms.

Experiment - In general, a collection of equipment and procedures which, when executed, add to the body of scientific or technological information. In ESP, the term experiment refers to the "model" or "models" corresponding to the general definition.

Functional Objective - A large section of an experiment's procedures which accomplishes a definite purpose, such as verifying equipment, collecting baseline data, etc.

Model (Experiment Model) - The database representation of an experiment or part of an experiment. A model is a collection of constraint and execution definitions. Some of the definitions apply to the whole model and some apply to the "steps" of the model.

Step - The smallest, clearly delineated part of an experiment model. Steps are usually executed in order, but not necessarily contiguously. Resource and crew requirements of a model are shown at the step level.

Performance - An execution of an experiment model. A model may be performed multiple times to collect additional data.

Experiment Timeline - A time history of experiment and related activities occurring during a mission. These activities are represented by the start and stop times of model steps.

ESP FEATURES AS SEEN BY THE USER

ESP provides an array of options for retrieving, generating, modifying, verifying, and documenting experiment timelines. There is no hierarchy among these options. The program internally maintains up to three schedules in addition to the current schedule. The user may save the current schedule at any point and may choose any saved schedule or the current schedule as the starting point for any option. One possible scenario which shows the use of multiple options for building a schedule follows. Start with a new schedule and retrieve the crew duty cycle from a file, edit the retrieved crew duty cycle with the manual scheduler, retrieve the timeline for several experiments from another file, edit this data, and use the automatic scheduler to schedule other activities. Interspersed with building up the schedule, the user can check the schedule for constraint violations and use the output options to examine the schedule. The user's view of the major features is described below.

When using the external retrieval option, the user specifies the start and stop times of the retrieval and the experiment models to retrieve. The program does not allow double

476

booking of a crew member or overlapping performances of the same model, but no other validation checks are made.

When using the automatic scheduler, the user specifies the models to be scheduled and, optionally, the order in which to process the models. This order does not affect the schedule except that resources are assigned on a first-come, first-serve basis. The user also specifies weighting parameters reflecting the priority of the models, the scheduling start time, the grading criteria for the schedule, and the number of scheduling passes to make.

The manual scheduler, also called the schedule editor, is used to enter activities whose requirements are not well modeled or whose placement is predetermined, as is the crew duty cycle. When using the manual scheduler, the user enters the start and stop times and crew usage for each activity. Resource usage is taken from the model. This scheduler presents a screen full of data to be edited using form editing techniques and commands. When the user issues the command to commit the page to the schedule, the entries are checked for constraint violations. If the violations will not destroy the integrity of the schedule, the user may ignore the warnings and update the schedule. A chart showing where each required resource of an activity is available and the intersection of these availabilities is provided to assist the user. The manual scheduler also provides easy access to the automatic scheduler for quick scheduling of a model.

Violations (overuse of resources) can be introduced into a schedule in four ways: merging of two of more timelines by doing external retrievals, changing the mission availabilities, changing the models, and accepting violations while editing the schedule. ESP can search all or part of a schedule for these violations.

ESP has a complete set of output options which supports both building schedules and preparing charts and tables for documents. The output options are form driven; i.e., the user fills out a form on the terminal screen requesting the desired output, issues the execute command, and the output is produced.

The Experiment Scheduling Program is a highly interactive, user-friendly program designed to run primarily on Digital VT240 and Tektronix 4014 terminals. It checks all input for reasonableness and has in-line help text available as needed throughout the program. The program allows a user to interrupt any activity without corrupting internal or external data. A journal of all user interactions is maintained to assist in tracking the development of a schedule.

EXPERIMENT AND MISSION CONSTRAINTS (PROGRAM DOMAIN)

The domain of the Experiment Scheduling Program includes the experiment and mission constraints which are defined in this section. Each experiment model has several different types of requirements and constraints which determine the time at which the model may be scheduled: time constraints indigenous to the experiment model, time constraints relative to other models, time constraints relative to other timelines, and sharing of resources with other models and activities. ESP has a fixed rule base which handles these requirements. The requirements that the automatic scheduler can handle are discussed below.

Performance Time Windows - All performances of a model are scheduled within specified time windows. Each time window has a desired maximum number of performances to schedule. Windows may overlap and can be in any order, but scheduling is attempted in the windows in the order specified.

Performance Duration - Each performance is scheduled so that its duration does not exceed the specified maximum.

Delays between Performances - Each performance is scheduled within a specified time period relative to adjacent performances of that model.

Step Duration - Each step is scheduled so that its duration lies within a specified range.

Step Delays - Each step is scheduled within a specified time period relative to the previous step of that model.

Startup and Shutdown Steps - A model may have startup steps which are executed on the earliest performance and shutdown steps which are executed on the latest performance. However, there must be a core of steps which is executed on all performances.

Sequencing - A model may be required to be scheduled within a specified time period relative to the performances of another model.

Concurrency - One step of a model may be required to be scheduled at the same time as one step of another model.

Time constraints relative to other timelines introduce the concept of targets and attitudes. A target is usually an object to observe, while an attitude is usually a vehicle orientation. Each target and attitude has a set of acquisition and loss times. Some examples of targets are when a site on the earth may be viewed from the vehicle, when the

vehicle is in sunlight, when the magnetic flux is above a specified threshold, and when TDRS communications are available. Some examples of attitudes are when the payload bay is pointed toward an object, when the vehicle is rotating at a certain rate, and when the vehicle is maneuvering from one attitude to another. For some mission analyses, the attitude timeline is not a constraint but is a consequence of the schedule produced by the program. Attitudes and targets are read from the same file and have the same format. ESP treats this data according to the rules requested by the model, not according to the physical definition; i.e., regardless of what the data is, if it is presented as a target, it is treated as a target; and if it is presented as an attitude, it is treated as an attitude.

Targets - A step may be scheduled during a time of requested target availability. Up to three targets to be intersected may be specified. All targets must be available simultaneously during the entire step.

Attitudes - A step may be scheduled during a time of requested attitude availability or nonavailability. Up to five attitudes to require or avoid may be specified. If the attitudes are to be avoided, all the attitudes must be unavailable during the entire step. If the attitudes are required, one attitude must be available for the entire duration of the step.

The shared resources handled by ESP are divided into two broad types: nondepletable and depletable. A nondepletable resource, such as a TV camera, is one whose availability is restored when the consuming activity terminates. Nondepletable resources are maintained in either integral or floating-point amounts. Those maintained in integral amounts are called equipment. A depletable resource, such as a reel of film, is one whose availability is not restored when the consuming activity terminates. Depletable resource consumption can be specified as a fixed amount, as a constant rate over time, or as a function of time and a specified nondepletable resource. Thus power can be integrated to compute energy. ESP schedules the models in such a way that the resource usage never exceeds what is available.

Shared Resources - A step may require depletable, nondepletable, and equipment-type resources. Resource usage usually terminates upon completion of a step. If it is desirable for resource usage to continue until the start of the next step, *resource carry-through* is requested. All nondepletables and equipment are maintained during the delay, while usage of depletables is maintained only for those resources whose consumption is based on time.

479

Balanced Crew Usage with Lock-in - A step may specify lists of crew members and the number to select from each list. This gives the program some freedom in assigning the crew. If a choice of crew members exists and no other rules take precedence, ESP attempts to balance the crew usage among the crew members as it schedules each performance. Whenever it is required that the same subset of crew members performs several steps of a performance, crew lock-in is specified.

Crew Monitoring - A crew member may be scheduled to monitor a long-running step for short periods of time.

Another program in the experiment scheduling group of programs is used to build and edit the models. The models are stored on a data file and read by ESP. Figure 1 shows a typical model which has most of the requirements discussed above. Most missions require more than 100 models; Spacelab-1 had 346 models. A significant effort is expended by the mission planner in building the models for a mission. A typical schedule contains about 4000 activities. This is less than 10% of the capability of ESP. Table 1 gives a summary of capacities for the program.

THE EXPERT SYSTEM (AUTOMATIC SCHEDULER)

In this section we shall take a bottom-up approach to explaining the expert system embodied in the automatic scheduler in ESP. The automatic scheduler consists of five components: the bookkeeper, which maintains both the experiment timeline and the resource availability timeline; the checker, which determines when constraints are met on a one-by-one basis; the loader, which is the set of rules to determine when to schedule each activity; the explainer, which traces the activities of the loader and the checker; and the selector, which selects the next experiment to schedule. The loader depends on the checker to supply windows of availability, and the checker depends on the bookkeeper to maintain the necessary time histories. While the primary function of the bookkeeper and the checker is to support the loader, they also support external retrieval, manual scheduling, and searching a schedule for constraint violations. The bookkeeper also supports output. Figure 2 shows the major components of ESP and their interfaces.

Bookkeeper
To explain how the bookkeeper works, we shall first show one way a human would do the task and then tell how that same approach was implemented in ESP. Assume that a mission planner wants to schedule some activities which require power. Also assume the mission duration is 7 days and the

available power is 10.0 units. The planner would initialize
a table like the one shown here with the initial time and
value and the final time and a value of zero. The table is
read as follows: beginning at time
0/00:00:00, the power available is
10.0 units; at 7/00:00:00, the power
available changes to 0.0 units.

TIME	POWER
0/00:00:00	10.00
7/00:00:00	0.00

Suppose that an activity is scheduled
from 1/13:45:00 to 3/16:00:00 using
2.5 units of power. Also suppose that
another activity is scheduled from
4/03:28:45 to 4/09:00:15 using 3.75
units of power. Four new times are
inserted into the table showing the
reduction and restoration of power by
each activity.

TIME	POWER
0/00:00:00	10.00
1/13:45:00	7.50
3/16:00:00	10.00
4/03:28:45	6.25
4/09:00:15	10.00
7/00:00:00	0.00

The process gets more intricate when
activities overlap each other. Assume
that another activity is scheduled
from 3/16:00:00 to 5/18:35:30 using
1.25 units of power. Since the start
time already exists in the table, it
is only necessary to insert one new
time. However, the power availability
for all the time points between the
start and stop times inclusive must be updated.

TIME	POWER
0/00:00:00	10.00
1/13:45:00	7.50
3/16:00:00	8.75
4/03:28:45	5.00
4/09:00:15	8.75
5/18:35:30	10.00
7/00:00:00	0.00

This example is lacking only in scale! Doing the task manually
would require keeping up with as many as a hundred resources
and thousands of events. In addition to keeping up with
resource and crew usage, it is also necessary to keep up
with the experiment timeline so that sequencing, concur-
rency, and performance delay requirements can be checked.
Notice that the length of the table is a function of the
number of resource level changes, not the duration of the
mission segment being scheduled.

The bookkeeper in ESP emulates the manual process described
above by using specially designed file formats and processing
techniques to rapidly access and update the data. Since ESP
is implemented on a 32-bit machine and time is maintained in
integral seconds, the mission segment length is limited to
the largest integer that can be stored in a 32-bit word;
i.e., 2,147,483,654 seconds or 68+ years. The data structure
used to store the tables is limited to 100,000 events, thus
guaranteeing 50,000 activities. Updating all the intervening
time points between the start and stop times of an activity
is time consuming but permits the checker to operate much
more efficiently.

481

Checker

To explain how the checker works, we shall again discuss how
a human, now developing some expertise, would use the data
maintained by the bookkeeper to perform the checking function.
A typical request might be, "Where does the available power
first exceed 8.00 units?" A mission planner would quickly
scan the table above and determine that the window opens at
0/00:00:00 and closes at 1/13:45:00. If asked to find the
next window, the mission planner would remember how far down
the table checking had proceeded, resume there and discover
that the next window runs from 3/16:00:00 to 4/03:28:45,
followed by another window from 4/09:00:15 to 7/00:00:00.

In ESP the request presented to the checker might be, "Give
me the first window where all the resources required for my
activity are available." The request would also contain
earliest and latest times of interest. This limiting of the
search range allows the checker to respond much faster. The
checker can respond to questions about availabilities of
targets, attitudes, equipment, nondepletable resources, or
crew. The checker also responds to queries about the win-
dows where sequencing or concurrency requirements are met.

Loader

We shall now describe how an expert mission planner would
load a single step into the schedule. After deciding the
window in which the step might be scheduled, the expert
would ask the checker for the first window meeting one of
the step requirements. If this window were acceptable, the
mission planner would get the window for another requirement.
Checking would continue in this manner until all requirements
were met. When the intersection of the windows was as long
as the step, the step could be scheduled.

In ESP the loading, or scheduling, of steps is accomplished
in a manner similar to the human process described above.
The requirements are checked in a particular order; i.e, the
requirement windows have a defined hierarchy. Each window
is the search range for determining the window below it in
the hierarchy. Whenever a window is found to be unacceptable
(shorter than the minimum step duration), the next window in
the same search range (next higher window) is requested.
This is an example of depth-first searching. Whenever no
acceptable window is found at a particular level, the window
above it becomes unacceptable and the next window at that
level is checked. Checking continues in this manner until a
set of nested windows for all requirements has been generated
or there are no more windows at the highest level and the
step cannot be loaded.

Once a lowest-level acceptable window is found, the step is
loaded within this window according to several rules. The

highest priority rule is to start the step as early as possible (front load). Another rule is to maximize the step duration within the window up to the limit specified on the model. If after assigning the start and stop times of the step there remains a choice of crew members to assign, then they are assigned based on the crew balancing equation. This flexibility is accomplished by placing the crew window at the bottom of the window hierarchy.

But the task is not just to schedule steps, but to schedule performances of models! This task requires finding a place in the schedule which meets not only the requirements of all steps individually, but also the delay constraints between steps and performances, the resource carry-through requirements, the sequencing and concurrency requirements, and the performance windows specified with the model.

Confronted with this larger task, the expert mission planner would begin by defining a window in which the performance must be scheduled. This window is determined by considering the window from the selector, the performance window from the model, performance delays, and sequencing. The expert would load the steps in this window on a trial basis, moving them around until all steps were validly loaded. Only then would the bookkeeping function be conducted and the performance actually scheduled. The detailed performance loading process embodied in ESP is given below and is also depicted in Figure 3.

The program computes the window for the first step to be loaded, beginning at the start of the performance window and leaving enough room for the remaining steps. The window for each subsequent step is similar except that it starts after the end of the previous step by an amount equal to the minimum step delay. After loading a step, the program may reload other steps based on several backtracking rules: if the delay relative to adjacent steps of the model is violated, reload the already loaded step; if the maximum performance duration is violated, reload the first step; if resources are unavailable for carry-through, reload the previous step; etc. When reloading steps, ESP adjusts the step window so that the same backtracking rule is not violated again.

The above discussion does not take into consideration all of the requirements that exist in the domain specified for ESP. Other rules are included to check for depletable resources. If depletable resource usage is a function of time, the step durations may have to be chosen at less than the otherwise available maximum. The scheduling and de-scheduling of startup and shutdown steps must be performed and crew monitoring must be scheduled. Additional back-tracking may be required to select a different crew group

when crew lock-in is in effect and a step other than the first step of the lock-in sequence fails.

After all steps of a performance are loaded, the performance is scheduled and all bookkeeping functions are conducted. In addition to asking the bookkeeper to update its data, the loader also updates the crew balancing parameters, the grading equation parameters and other data.

ESP does one more thing that a good expert would do. After scheduling a performance, the program saves a snapshot of its computed data. When asked to schedule another performance of that model, the program resumes scheduling based on the snapshot. This heuristic feature significantly enhances the program's performance. However, it is possible for a snapshot to be invalidated by scheduling another model. After scheduling each performance, ESP checks all snapshots and clears those that are invalid.

We have only told half of the story! The manual process to handle the concurrency requirement would overwhelm even the most expert of mission planners. When scheduling models with concurrence where neither model can be scheduled without the other, ESP processes them simultaneously. The complexity of the loading task is at least doubled. Since both models are active at the same time and may require the same resources, the checker must account for the already loaded steps of the other experiment when computing availability windows. Determining when and how to backtrack is more complicated. The computation of step windows is also more difficult.

ESP has a last-chance ploy that is invoked after all other attempts to schedule a performance have failed. If any steps have variable durations, these durations are forced to the minimum value and the loading process is repeated. Remember that the original desire was to maximize step durations.

Explainer
The loader provides only summary data to the user as it works, reporting that a performance is scheduled, that it is scheduled with the last-chance logic, or that it could not be scheduled. Of course, a user may examine a schedule with the output options to see where the performances, and the steps of the performances, were placed. Normally the user only wants to see the schedule, but occasionally the user may want to know why a performance was placed at a particular place or why it failed. To obtain this information, the user activates the explainer before invoking the loader. The loader activities and checker responses are then stored and may be either printed or displayed on the terminal. When displayed on the terminal, the tabular data may be scrolled forward or backward in its screen window while an

animated bar-chart representation of the data is shown in another screen window.

Selector

The loader places the models in the schedule based on model requirements and current availabilities. Since ESP cannot deschedule or reschedule a performance while generating a schedule, the order of attempting to schedule the models, referred to as the selection order, has a significant effect on the schedule. Selecting the next performance to schedule is the function of the selector. The user divides the models to be scheduled into groups, assigns a selection method to each group, and then specifies which groups to schedule. The selector processes the groups and passes the models to the loader one performance at a time. ESP provides two methods of selecting the models for scheduling.

The first selection method is the fixed-order method. The user specifies the order of selecting the models, possibly on a performance-by-performance basis. When processing a fixed-order group, ESP makes automatic adjustments to account for sequencing and concurrency.

The other selection method is the random-order method. The user specifies a seed for a pseudo-random number generator, the group members, and relative weighting factors for the members. As it processes a random-order group, ESP takes into account sequencing and concurrency. At the user's request, ESP automatically generates multiple schedules and saves the "best" one. This Monte-Carlo technique allows the program to perform exhaustive searches without further user intervention.

ESP also provides a pseudo or manual selection method. The manual scheduler provides a gateway to the automatic scheduler which bypasses the selector. In this mode the user selects the model and specifies the topmost window in the performance level hierarchy.

SUMMARY

When we set out in November of 1979 to develop ESP, we were good programmers but knew little about scheduling. The requirements, of which we had seen only a sketch, were very challenging. We did not set out to write an expert system, at least not by that name. We firmly believed that if we could write the rules for scheduling on paper in English, we could write the program. We also challenged ourselves to develop a robust program which met all the requirements. After working on the task three months, we published a detailed requirements document. During the next two months, Mrs. Davis formulated the scheduling approach described in

this paper. It required eight more months to fill in the details of her plan. During the design effort, Mr. Jerry Weiler, the scheduling expert at MSFC, provided valuable insight to the scheduling process. In February, 1980, we wrote the first line of code. Another 24 months were required to implement the program. Over the past four years we have made improvements to the user interfaces on a part-time basis. Only in the past two years have we come to realize that we should apply terms like "expert system," "depth-first search," "backtracking," "forward chaining," and "explanation facility" to ESP.

ESP is implemented in FORTRAN 77 on a VAX computer. We used strict structured programming techniques to write 95,000 lines of well commented code without GOTO statements. The program contains 13,000 IF statements and 850 DO-WHILE statements. Emphasis was placed on speed and file I/O was minimized. The program, running on a VAX-11/785 with 15 other users, can retrieve a 4000-step schedule, performing the necessary bookkeeping, in about five minutes. Normally, a mission planner dynamically builds up a schedule by a combination of editing and building the experiment models and continuously adding to or modifying a schedule. Occasionally the user may revert to a clear schedule which has only the preplaced activities in it. Over a period of three weeks, one or two mission planners working together can develop a detailed schedule for a 7-day mission. This time does not include obtaining the data required to build the models.

In the future we expect to make several improvements to the program. The current loading scheme places the activities toward the front of the mission. Experimenters usually want their activities done as early as possible for three reasons: if the experiment fails, there is more time to repair it; they have a chance to analyze data during the mission, modify their procedures and collect more or better data; and if the mission is shortened for reasons of safety, they will have collected more data. However, there is a need to load the schedule in other ways, and we are currently implementing backloading rules. As was noted earlier, the order of selecting the models for scheduling has major effects on the schedule. We are investigating other selection methods which automatically choose the next model to schedule so as to produce a better schedule. Improved selection methods are a prime candidate for the application of Artificial Intelligence.

```
...................................  N2F6A   ...................................
* First perf. steps   1 thru 3  Maximum perf. duration:   1:00:00      PERFORMANCE WINDOWS   *
* Middle perf. steps  2 thru 3                                      Start      End    #Perf  *
* Last perf. steps    2 thru 4  Perf. Delay:  0:00:00,202:00:00   1- 1/00:00:00, 3/09:00:00,  5  *
* Sequenced with N2F2EB1 , Delay:  0/00:00:00, 6/00:00:00         2- 3/09:00:00, 6/00:00:00,  2  *
* Step 3 has TWO WAY concurrency with N1FO3A at step 5                                        *
...................................................................................
Step 1 ----------------------------------------------------------------------------
                                 Perform initial on-orbit checkout
Duration:   0:15:00,  0:15:00                            Nondepletables      Depletables
          Crew               Targets        Equipment  POWER    =   0.645 ENERGY   =   0.164
   Use 1                     SUN            DDU/KB  = 1  ECAS    =   4.816
   PS-1                      TDRS           ECAS-PRG = 1  DATA   =   0.512
   PS-2                                     ECAS-STL = 1
Step 2 ----------------------------------------------------------------------------
Delay:      0:00:00,  0:10:00              Start solar data collection
Duration:   0:03:00,  0:03:00                           Nondepletables      Depletables
          Crew               Targets  Attitudes  Equipment  POWER  =  0.645 ENERGY  =  0.033
   Use 1                     SUN      SOLAR      A/V REC  = 1  ECAS  =  4.816
   PS-1                                          ECAS-PRG = 1  DATA  =  0.512
   PS-2                                          ECAS-STL = 1
                                                H2O-INH  = 1
                                                DDU/KB   = 1

Step 3 ----------------------------------------------------------------------------
Delay:      0:00:00,  0:10:00              Collect solar spectrograph data (auto mode)
Duration:   0:50:00,  1:10:00                           Nondepletables      Depletables
     Crew Monitoring         Targets  Attitudes  Equipment  POWER  =  0.645 ENERGY  =  0.545
   PS-1  Duration   0:10:00  SUN      SOLAR      A/V REC  = 1  ECAS  =  4.816
   PS-2  Cycle      1:00:00                      ECAS-PRG = 1  DATA  =  0.512
         Tolerance -0:05:00                      ECAS-STL = 1
                   +0:10:00                      H2O-INH  = 1
Step 4 ----------------------------------------------------------------------------
Delay:      0:00:00,  0:10:00              Observe sunset and go to standby mode
Duration:   0:03:00,  0:03:00                           Nondepletables      Depletables
                                       Attitudes  Equipment  POWER  =  0.645 ENERGY  =  0.033
                                       SOLAR      A/V REC  = 1  ECAS  =  4.816
                                                  ECAS-PRG = 1  DATA  =  0.512
                                                  ECAS-STL = 1
                                                  H2O-INH  = 1
```
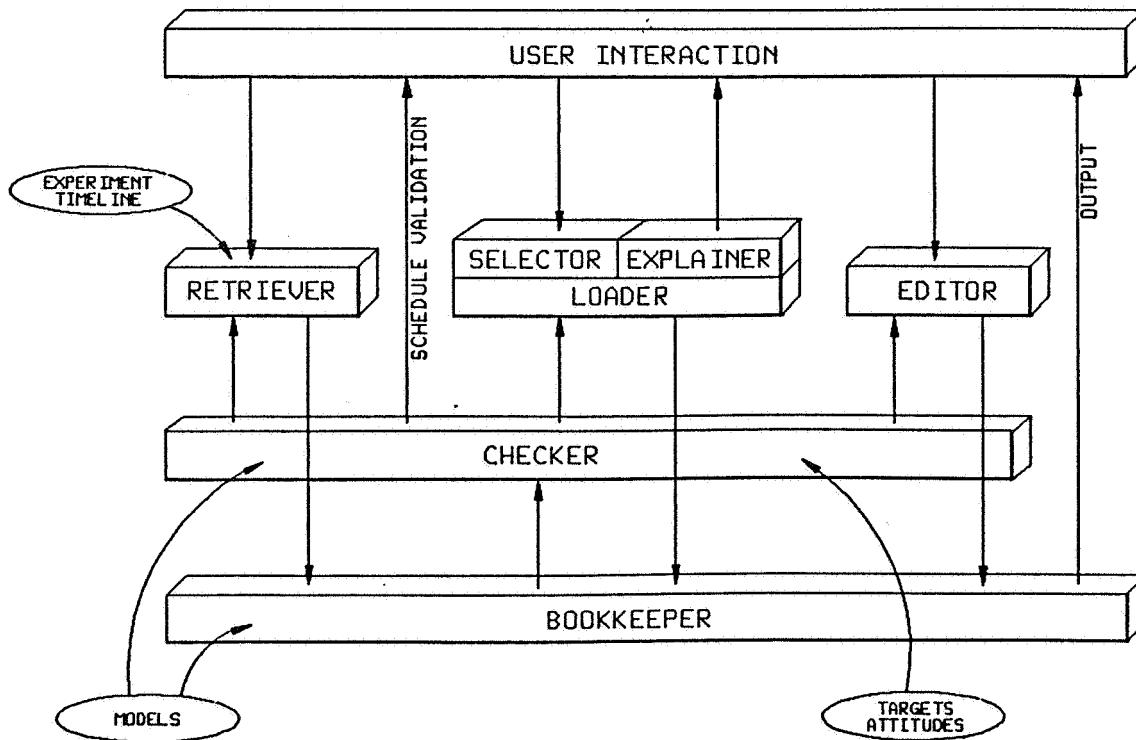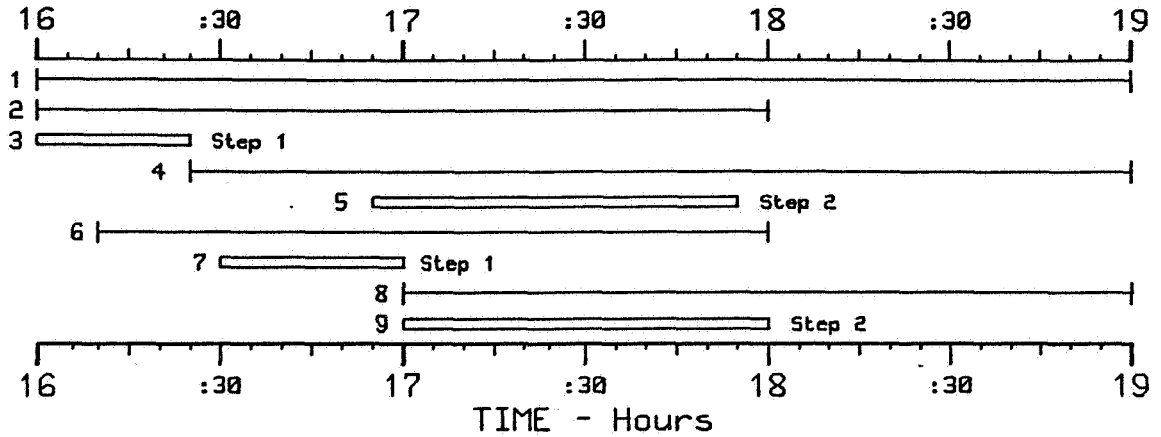
Figure 1: Model of an Experiment Functional Objective

Figure 2: Experiment Scheduling Program Components

NOTES:
* Resource avail-
  ability windows
  are not shown.
* Drawing numbers
  correspond to
  explanations.

```
MODEL:
      Performance window:  0/16:00 to 0/19:00
Step 1:       Operate air sample collector
  Duration:   0/00:20 to 0/00:30    POWER:  1.762

Step 2:       Analyze air sample
  Delay:      0/00:00 to 0/00:15
  Duration:   0/01:00 to 0/01:00    POWER:  0.889
```

EXPLANATION:
1. Bottom window at the performance level.
2. Step window for step 1.
3. Step 1 scheduled as early as possible and for as long as possible in the resource availability window.
4. Step window for step 2.
5. Step 2 scheduled as early as possible in the resource availability window. Note that the step delay is violated.
6. Recomputed step window for step 1 accounting for the actual starting time of step 2 and maximum delay before step 2.
7. Step 1 scheduled as early as possible and for as long as possible in the resource availability window. Note that it overlaps step 2.
8. Recomputed step window for step 2.
9. Step 2 scheduled as early as possible and for as long as possible in the resource availability window. Since there are no violations, the performance is scheduled.

Figure 3:   Loading Steps with Variable Delay and Duration

MISSION DATA

Mission segment length based on output format	= 1000 days
Experiment models	= 500
Crewmen	= 8
Equipment types	= 99
Other nondepletable resources	= 25
Depletable resources	= 25
Targets and attitudes	= 200
Acq Losses per target/attitude	= 500
Steps in schedule	= 50000

EACH MODEL		EACH STEP	
Steps	= 50	Crewmen used	= 8
Performances	= 500	Crew preference lists	= 3
Performance windows	= 10	Equipment types	= 15
Concurrency	= 1	Other nondepletable resources	= 10
Sequencing	= 1	Depletable resources	= 10
		Targets to intersect	= 3
		Attitudes (select or avoid)	= 5

Table 1:   Experiment Scheduling Program Capacities

Intelligent Interface Design and Evaluation[1]

Frank L. Greitzer, Ph.D.

Martin Marietta Denver Aerospace
Space Station Program
P. O. Box 179
Denver, CO 80201

Abstract. Intelligent interface concepts and systematic approaches to assessing their functionality are discussed. Four general features of intelligent interfaces are described: interaction efficiency, subtask automation, context sensitivity, and use of an appropriate design metaphor. Three evaluation methods are discussed: Functional Analysis, Part-Task Evaluation, and Operational Testing. Design and evaluation concepts are illustrated with examples from a prototype expert system interface for environmental control and life support systems for manned space platforms.

Introduction

Increased information processing requirements and the maturation of high-technology hardware and software have led to increased automation in nearly all aspects of society. This trend has caused a shift in the human's role from largely perceptual/motor tasks (e.g., analyzing signals, adjusting controls) to more cognitive ones (e.g., monitoring, evaluating, and managing automated or semi-automated system processes). Unfortunately, the full potential of new technology is not often realized due to poor interfaces with the human operator-turned-evaluator. There is increasing recognition of the need to enhance user interfaces with intelligent support functions, but there is a lack of clarity on what constitutes an intelligent interface, and very little guidance on how to evaluate its effectiveness. This paper addresses three questions about intelligent interfaces: (1) How do we know when we need one? (2) How do we know when we have one? and (3) How do we know how good it is? It is hoped that, by elaborating these issues, some progress can be made toward characterizing intelligent interfaces and systematic approaches to assessing their functionality.

Intelligent Interface Design

The Need for Intelligent Interfaces

At the outset, let us consider where intelligent interface concepts should be applied. Certainly, intelligent interface concepts are not required in predominantly manual operations or completely automated systems. The most appropriate applications are those human-computer systems in which information processing and decision making responsibilities are shared. A prime source of system complexity in these systems is an implicit design philosophy that focuses on the form and literal content of isolated user-computer *transactions* rather than their function in the larger system (Greitzer, Hershman, & Kaiwi, 1985). While proper interface design to support such transactions is essential, the design concept should also reflect more global aspects of user-computer interactions. As the goal-directed responsibilities of the software increases, the relationship between the human and the computer approaches that of two cognitive systems (Fitter & Sime, 1980), and the need for an intelligent intermediary (i.e., interface) becomes more critical (Greitzer et al., 1985). This places a new emphasis on knowledge of the user's goals and intentions, development of new tools and procedures, and incorporation of contextual information. Such joint human-machine cognitive systems provide an integration of partial and overlapping expertise that has the potential to produce better performance than either human or machine alone (Woods, 1986).

Intelligent Interface Features

Given an accurate accounting of the user's expertise and the task requirements, informed decisions may be made about user-computer task allocation and the tailoring of intelligent interface concepts to the problem. This section discusses four major features that contribute to the design of intelligent interfaces.

---

1. *Interaction Efficiency.* This is a collection of related interface concepts that address issues of direct manipulation (Shneiderman, 1983) and user satisfaction (e.g., Rissland, 1984). Shneiderman (1983) advocates graphic representations of objects (icons) that may be manipulated directly by a pointing device. This provides the user easy access to tools and information, facilitates the user's acquisition of an accurate mental model of system operations, and contributes to user satisfaction with the system. Some interaction-efficiency features that foster direct manipulation and user satisfaction are: immediate feedback for user actions; high resolution graphic representatons of processes, objects, tools, and information sources to relieve the user's information processing load; natural and easily recognized icons to relieve the user's memory burden for otherwise obscure command codes; forgiving treatment of user errors (e.g., easy to undo actions); consistent user control of the interaction (i.e., the interface presents the system as a tool to be used at the user's discretion); information presentation formats that are compatible with the user's conceptualization of the problem; ability of experienced users to define "macros" that customize composite functions to save time and reduce memory load; display formats and interaction protocols that facilitate the user's understanding of system functioning; multiple windows that exhibit only that information required to perform the task; and a safe environment that encourages discovery about system operations through hands-on experience.

While these direct manipulation tools and user-friendly features are powerful devices, they do not themselves relieve the user's cognitive burden in process management and formulating goals in complex problem-solving tasks. Here, more cognitive support is required to automate selected tasks, provide more flexible interface dynamics, and perform high-level task management functions.

2. *Subtask Automation.* The purpose of automation is to support and augment human operators and improve system performance, productivity, safety, and economy. Automation may include one or more of the following functions (Rouse and Morris, 1985): synthesis (generation of alternatives); analysis (evaluation of alternatives); decision (selection among alternatives); control (implementation of alternatives); and monitoring (observation of results).

Automation is most appropriate for tasks that people cannot or would rather not do. Candidate tasks for automation include those that are menial, routine, time consuming, or that tend to overload the human operator (e.g., that require large amounts of memorization). Automation of monitoring and control functions is most typical: Examples in office software environments are setting terminal characteristics and backing up files (Rissland, 1984). In space applications, voluminous health/status data must be monitored and checked against established thresholds; equipment inventories and maintenance requirements can be updated automatically as problems are discovered. Automated decision making is less typical because of the reluctance to relinquish total control to the hardware/software system. Nevertheless, life support systems aboard manned space platforms will include automated decisions to shut down faulty components in redundant systems. Automated synthesis and analysis functions are largely in the domain of expert systems technology; space applications include fault diagnosis and recovery for life support systems.

3. *Sensitivity to Context.* Contextual knowledge, particularly task goals and interaction history, determine the flow of the interaction (Croft, 1984). Sensitivity to context elevates the sophistication of the interface, making possible a variety of indispensible services for the user. To capture this contextual knowledge, artificial intelligence programming techniques can be used to model the relationship between high-level goals, required tasks, and associated information requirements. This allows the software to recognize the context of a user's present actions and invoke supporting data analyses or presentations, appropriate algorithms, or expert systems.

Some of the more important context-sensitive functions are: change interaction dynamics in response to user preferences (e.g., amount of feedback, screen formats, command words); adapt interaction dynamics to user workload or experience; monitor progress toward the goal; inform the user of status information; help maintain the user's sense of position; screen the user's commands to prevent mission-critical errors; provide online assistance (especially context-sensitive help messages); recommend actions; anticipate user needs for information presentation or analysis; explain actions or recommendations; report on possible side effects of intended actions; and provide rapid forecasting of "What-if" questions.

4. *Appropriate Design Metaphor.* Whenever users interact with computers they acquire a mental model (Norman, 1983) or picture of how the system works. This mental model is conveyed through a *design metaphor* about the system's functional behavior (Greitzer, Kaiwi, & Hershman, 1986). At worst, the user perceives the system as a "black box" that should be ignored or turned off when its behavior is unsatisfactory. More desirable is a design metaphor that inspires user confidence, satisfaction, and efficiency in completing the task. An understanding of the

user's requirements should motivate the choice of design metaphor that best accomodates the user's abilities, limitations, biases, preferences, and experience. Inappropriate or inadequate conceptualizations lead to poorly designed systems that overestimate, underestimate, or ignore human performance factors that system support functions could accomodate (Greitzer et al., 1986).

The design metaphor should be appropriate to the task and the user's needs and level of expertise. An important consideration here is *control*. A novice or occasional user neither wants nor needs the same amount of control (authority) as an expert user (Rissland, 1984). On the other hand, a novice-oriented interface can lead the user to behave more like a novice (Zachary, 1985). As automated functions increase, the role of the computer system (as embodied in the design metaphor) shifts toward increasing system control. In traditional systems, virtually all goal-directed behavior and responsibility lies with the user. In an "assistant" relationship, the authority resides with the user, who directs the system's actions. An "associate" or "teammate" relationship reflects a collaborative arrangement where some critical tasks are automated in order to produce the fastest, most appropriate response to a situation while minimizing user-system interaction. Here, the user and the system each have authority over selected functions. They can monitor each other's performance so that each can anticipate what the other will do in response to new situations (Riley, 1985). As technology progresses, fully autonomous systems will become possible. However, it is unlikely that humans will be taken out of the loop completely, especially in critical functions such as life support systems.

Manned Space Platform Applications

One of the most important potential applications for intelligent interfaces on manned space platforms is the command and control workstation, which would serve as the astronauts' link to the myriad of systems, subsystems, and information sources vital to the operational integrity of the platform. Sophisticated processing algorithms and automated functions will maintain life support, communications, thermal, power, propulsion, and navigation systems—but there is a requirement to avail the crew rapid and easy access to the health and status of these systems and their components. In addition, the crew will be occupied fully by crowded activity schedules, and they will not have expertise in system/subsystem functionality. Since they will tend to interact with these subsystems only when problems arise, the crew will not acquire accurate, effective mental representations of subsystem operations without intelligent support functions at the user-computer interface.

Some of the functions that such workstations will provide, and thus that would benefit from intelligent interface features, are: develop and display crew activity plans; prompt or remind the crew to perform planned tasks; maintain global activity and state information on hardware, descriptions of current software, and on historical information such as transactions, messages, and alarms; manage global caution and warning messages by synthesizing and filtering messages from subsystems and payloads; provide global fault management support; control the execution of all commands and transactions to insure they are valid; support inventory and maintenance functions; and provide simulations and other tutorial support for onboard training. Appropriate for these support functions is the metaphor of an intelligent assistant, which contains requisite knowledge of onboard systems, personnel capabilities, and required tasks and procedures.

An intelligent assistant would help manage the planning and execution of high-level tasks. It would augment the user's limited memory and attention abilities by tracking the progress of multiple tasks, providing status information, maintaining the "big-picture" and providing checks to ensure that important subtasks are not forgotten. Since crew members may not be familiar with (or even aware of) all of the application programs onboard, the intelligent assistant should maintain descriptions of current databases, applications programs, etc., and provide access to these support functions at appropriate times or in response to user requests. The more intelligent this function is, the less specific the user's request needs to be: The burden of knowing *which* resource has the required information or function lies with the intelligent interface. For example, Jamar (1986) describes an intelligent command language translator (ATOZ) that satisfies high-level queries by recognizing and invoking appropriate data servers and then presenting the answer in a single, consistent format.

At a lower level in the system/subsystem hierarchy, expert systems are being developed for automated control of power and environmental control/life support functions. A major function of intelligent interface design for manned space platforms is to assure a high degree of consistency in the displays and interaction protocols of the various expert systems and other software applications.

Evaluation

Traditional Interfaces

Various guidelines and standards exist for displays, controls, and anthropometry in military and space applications—e.g., Smith and Mosier (1984); MIL-STD-1472C (Department of Defense, 1981); and MSFC-STD-512A (NASA, 1976)—but these address the more "transactional" aspects of user-computer interaction, and they offer little guidance on objective measures and methodological tools.

Recently, some methodological approaches have been tried. Foley, Wallace, and Chan (1984) compared graphic interaction methods and devices based on published experimental findings and personal observations. They used subjective ratings to measure cognitive, perceptual, and motor load, visual and motor acquisition, ease of learning, fatigue, error proneness, and type of feedback. A more quantitative approach was developed and applied by Card, Moran and Newell (1983). Their Keystroke Level Model predicts time required by the user and the system to perform different types of text editing tasks, based on analyses/estimates of times required for elementary actions like pressing a key, pointing with a mouse, or mentally preparing for an action. At a higher level, their Unit-Task Level of analysis uses time estimates for specific functions called unit tasks (e.g., setting character fonts, numbering a figure, positioning text on a page). Roberts and Moran (1983) used a benchmark methodology based on the work of Card et al. (1983) to evaluate nine text editors. A set of 212 critical tasks (e.g., inserting, deleting, or replacing text) was embedded in problems performed by representative (skilled or unskilled) users. Their *functionality* score is operationally defined as the percentage of the 212 tasks that the system can accomodate—this is not a user-performance variable but is judged by expert users based on rating criteria. Their *learning* score is the average amount of time it takes for a novice to learn a task. The *time* score is the average error-free time to perform a task; and the *error* score is the time the user spends dealing with errors, expressed as a percentage of the error-free time score.

Intelligent Interfaces

Advanced concepts of user-computer interaction are best implemented with rapid prototyping methods for exploring display designs and interaction techniques. Thus the interface characteristics tend to be implemented in an incremental fashion with increasing sophistication of user support functions. This evolutionary design process enables evaluation and development to proceed concurrently. An initial evaluation should check that the four major features of intelligent interfaces are represented (as appropriate). More detailed, empirical tests may be planned as the implementation progresses. Three approaches to evaluation are described here. Note that these methods are not mutually exclusive, nor does application of one preclude another.

1. *Functional Analysis.* The functionality variable of Card et al. (1983) and Roberts and Moran (1983) has general applicability for interface evaluation; namely, to ascertain if the conceptual goals of the interface have been realized. The method requires the specification of a detailed, task-specific taxonomy of user and computer actions, and then informed judgments about the extent of their implementation or support in the interface design. Human factors task analyses (Van Cott & Kinkade, 1972) may be used to derive user and computer activities for the taxonomy, allocate responsibilities between humans and machines, and suggest an appropriate design metaphor. The taxonomy should reflect in detail each of the intelligent interface criteria as they apply to the problem domain.

This approach to evaluation seems most valuable for comparing two or more candidate systems. Each system would be evaluated on all of the tasks/features defined in the taxonomy--e.g., using a rating scale or a checklist. To illustrate, consider the ARGES (Atmosphere Revitalization Group Expert System) interface that is being developed by Martin Marietta for space-based environmental control/life support systems (see papers by Mendler, Pachura & Suleiman (1986) and Bailey and Doehr (1986) in this conference). Like other expert systems targeted for space application, this is a special-purpose application that would not require all of the high-level user support functions envisioned for a command workstation. Nevertheless, the concept for interaction is based on the intelligent interface design philosophy described here. The most pertinent interface features for this application are interaction efficiency and context sensitivity.

A rating scale procedure was used to evaluate the implementation of these features in the ARGES interface near the end of its development. A questionnaire containing seventeen statements addressing interaction efficiency and seventeen addressing context sensitivity was administered to five people—four project personnel who were very familiar with the ARGES system and one domain expert from outside the company. They indicated their agreement

with each statement using response categories that were later numerically coded as strongly disagree = 1, disagree = 2 neutral = 3, agree = 4, and strongly agree = 5 (NA = not applicable was also allowed). The statements and their mean ratings are shown in Figure 1. Longer bars indicate greater agreement with the statements (and more effective implementations of the interface features). Two features (#21, online help and #31, recommendations) received NA ratings (they have not yet been implemented) and were not included in the analysis.

Mean ratings of 2.0 or lower were taken to indicate potential problems to be addressed in the next iteration. Deficiencies in three context-sensitive features were indicated: the need for error messages to suggest actions (#20); the ability to change screen formats (#29); and the ability of the interface to adapt to user experience level (#33). The overall average rating for the interaction efficiency features was 3.8; that for the context sensitivity items was 3.2 Twenty-two of the thirty-four features received mean values of 3.5 or more, indicating highly effective implementations. Most of this may be attributed to the extensive use of graphics for schematics, icons, and a mouse-sensitive hierarchical "map" that shows subsystem status and facilitates navigation through the system.

Some refinement of the functional analysis procedure is necessary before a high degree of confidence can be placed in the results. The subjective nature of the ratings is unavoidable, and this is compounded by the abstractness of most features. Brief supporting text can be provided to elucidate the individual items in the questionnaire. Statistical tests can be performed to measure the degree of agreement among the "judges." Finally, it should be noted that this procedure is not a *performance* evaluation because it is not based on *observed* performance of the system. The remaining methods attempt to observe and measure user-system interactions.
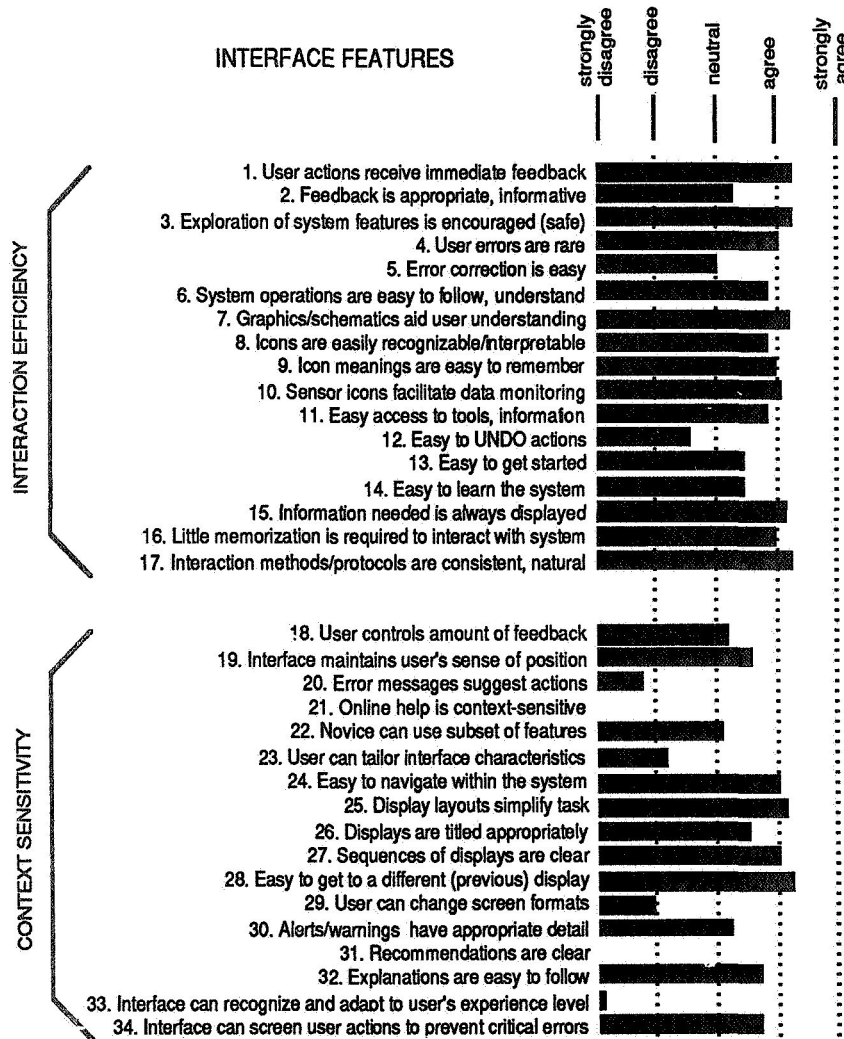

Figure 1. Mean ratings obtained in the functional analysis evaluation of ARGES.

2. *Part-Task Evaluation.* This method conducts operational tests on selected functional requirements, using the following steps: (a) identify the most important intelligent interface features; (b) select tasks or actions that employ these features; (c) specify criteria for successful performance; (d) develop performance measures based on these criteria; (e) embed the features in an operational, part-task simulation and collect performance data on the human-machine system; (f) evaluate performance of the system and diagnose probable causes of inadequate performance.

This method would apply when it is infeasible to conduct complete operational testing of the system, as for example when it is still under development and specific alternative interaction features are being considered; or when the capabilities of the system are so diverse that it is not possible to cover all aspects of performance (this latter case is exemplified by the method used by Roberts and Moran (1983) in evaluating text editors).

Table 1 shows several intelligent interface features selected for analysis in the ARGES interface. Column 1 lists the interface features, column 2 identifies user actions that use these features, and column 3 lists the performance measures for the selected functions. Three interface features are represented: a hierarchical map display that provides status information about platform subsystems and facilitates navigation within ARGES displays; the icons and graphic schematics that facilitate user-computer transactions, display component status information, and support an appropriate mental model of subsystem operations; and an explanation function that displays the reasoning used by the expert system. Performance measures for tasks associated with these features include user errors (e.g., selecting incorrect icons or menu items); number of actions taken to perform the task; and time required to complete an action (latency). While it is preferable to utilize software to record measures of performance, other methods include videotaping and direct observation using a stopwatch. Efficacy of these features may be assessed by comparing performance under alternative implementations or with and without the features. As of this time, no formal part-task evaluations have been conducted for the ARGES interface.

Table 1
Part-Task Evaluation Features

Feature	Tasks	Performance Measures
Hierarchical Map	Call up specified display	errors; number of actions; sequence of displays
	Observe state change	latency
Icons/Schematics	Observe state change	latency
Explanation format	Call up selected explanation	number of actions; errors

The advantage of this method is that it can be applied during rapid prototyping and system development to facilitate the evolutionary design/development process that is typical in implementing expert systems. The disadvantage is that it lacks the "gestalt" of a complete simulation. This is particularly important for assessing intelligent interface features because of the important role that context plays in their design. The observation of relatively disconnected parts of the proposed interface may not adequately reflect the larger problems of providing intelligent support.

3. *Operational Testing.* This method applies the empirical method described in #2, above, to the fully-implemented prototype system. Selection of test features and performance measures occurs as in the part-task evaluation, but additional consideration is given to observing more global user-computer interactions that reflect the ability of the interface to use contextual information, adapt to changing demands, and project to the user an appropriate view of the system. This method requires an operational, fielded system or a simulation in which the user-computer system can be "exercised" with realistic problems selected from a set of preplanned operational scenarios. Scenarios could be defined that focus on specific user-computer interactions, under specified conditions. These conditions would then serve as independent variables in controlled experiments with the interface. Examples of independent variables are

difficulty (complexity) of the problem and "tempo" (information processing demand) of the simulated external environment. Thus the effects of stress and various cognitive variables on system operations may be examined. Examples of performance measures (dependent variables) that might be used are: speed in performing a subtask; time to learn specific functions; long-term retention of commands or interaction methods; error frequency; correct use of tools; error types (e.g., errors that signal inconsistent interface features); shortcuts (especially those used by experts to "get around" awkward interfaces); "path" used in solving problem (e.g., sequence of displays requested); and usage data.

Shneiderman (1986) suggests that explicit acceptance criteria be established early on, e.g., when the requirements document is written. For example, one could specify the number of users to be tested, training time allowed, and required performance levels on selected benchmark tasks. If frequency of errors is recorded, then those with the highest rates should be examined to identify possible changes in software or training materials. Usage data for commands, displays, and tools can be used to modify the interface to simplify access to the most frequently used features. It is recommended that the facility for detailed data collection reside in the system software. This recording of performance data provides documentation of the interaction history, which can also be used by knowledge-based systems comprising context-sensitive functions of the intelligent interface.

If an interactive—as opposed to a more passive "consultant"— expert system is involved, a desirable (but often difficult to obtain) comparison would be between the unaided human, the software alone, and the combined human-machine team. The most conclusive evidence that a collaborative human-machine relationship has been achieved is a demonstration that the human-machine team performs better than either member alone. Verification and validation of the expert system's contribution to the problem is a critical part of the system evaluation, for which we again find little practical guidance. This type of evaluation is also being pursued, but is beyond the scope of the present discussion.

Finally, subjective measures of user acceptance can be obtained to supplement the observed performance measures. In addition to a general satisfaction rating, users can be asked to comment on individual displays or interaction protocols. Other questions should request (expert) user input about the realism of the simulation (to validate the experimental procedure) and about the quality of the contribution of the expert system or other decision aids associated with the interface.

Conclusion

The purpose of intelligent interface design is to incorporate knowledge about the user and the problem domain in a graphic, conceptual "window" through which the user views the domain. The human is in charge; the system serves as a staff assistant/consultant that supports the user by providing informative advice, reminders, and automated support. Evaluating the implementation of this joint cognitive system should be an ongoing process. I have attempted to motivate and define some approaches to evaluation, but more rigorous methodologies and quantitative measures are needed to assess the extent to which the interface design goals have been satisfied.

References

Bailey, P. A., and Doehr, B. B. "Knowledge acquisition and rapid prototyping of an expert system: Dealing with 'real world' problems." Paper delivered to the Conference on AI for Space Applications. Huntsville, Alabama, November 13-14, 1986.

Card, S. K., Moran, T. P., and Newell, A. The psychology of human-computer interaction. Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

Croft, W. B. "The role of context and adaptation in user interfaces." International Journal of Man-Machine Studies, 21, 283-292, October 1984.

Department of Defense. Military Standard, Human Engineering Design Criteria for Military Systems, Equipment and Facilities, MIL-STD-1472C, May 1981.

Fitter, M. J., and Sime, M. E. "Creating responsive computers: Responsibility and shared decision-making," in H. T. Smith and T. R. G. Green (Eds.), Human Interaction with Computers. London: Academic Press, 1980, 39-66.

Foley, J. D., Wallace, V. L., and Chan, P. "The human factors of computer graphics interaction techniques." IEEE Computer Graphics and Applications, November 1984, 13-48.

Greitzer, F. L., Hershman, R. L., and Kaiwi, J. "Intelligent interfaces for C^2 operability." Proceedings, IEEE International Conference on Systems, Man and Cybernetics, November 1985, 782-786.

Greitzer, F. L., Kaiwi, J., and Hershman, R. L. "Interfaces for combat decision support." In R. Penn & A. Crawford (Eds.), Independent Research and Independent Exploratory Development FY85, Navy Personnel Research and Development Center, San Diego, Special Report SR 86-1, January 1986, 35-39.

Jamar, P. G. "Impact of automation on Space Station MMI design." Paper delivered to AIAA Space Station in the Twenty-First Century. Reno, Nevada, September 3-5, 1986.

Mendler, A. P., Pachura, D. W., and Suleiman, S. A. O. "ARGES: An expert system for fault diagnosis within space-based ECLS systems." Paper delivered to the Conference on AI for Space Applications. Huntsville, Alabama, November 13-14, 1986.

National Aeronautics and Space Administration. Man/System Requirements for Weightless Environments. George C. Marshall Space Flight Center, Alabama, MSFC-STD-512A, December 1976.

Norman, D. A. "Some observations on mental models." In D. Gentner & A. L. Stevens (Eds.), Mental Models. Hillsdale, NJ: Erlbaum, 1983, 7-14.

Riley, V. "Monitoring the monitor: Some possible effects of embedding human models in highly automated manned systems." Proceedings, IEEE International Conference on Systems, Man and Cybernetics, November 1985, 6-9.

Rissland, E. L. "Ingredients of intelligent user interfaces." International Journal of Man-Machine Studies, 21, October 1984, 377-388.

Roberts, T. L. and Moran, T.P. "The evaluation of text editors: Methodology and empirical results." Communications of the ACM, 26 (4), April 1983, 265-283.

Rouse, W. B., and Morris, N. M. "Understanding and avoiding potential problems in implementing automation." Proceedings, IEEE International Conference on Systems, Man and Cybernetics, November 1985, 787-791.

Shneiderman, B. "Direct manipulation: A step beyond programming languages." IEEE Computer, 16 (8), August 1983, 57-69.

Shneiderman, B. "Correct, complete operations and other principles of interaction." In G. Salvendy (Ed.), Human-Computer Interaction. Amsterdam: Elsevier Science Publishers, 1984.

Shneiderman, B. Designing the user interface. Reading, Massachusetts: Addison-Wesley Publishing Company, 1986.

Smith, S. L., and Mosier, J. N. Design guidelines for the user interface for computer-based information systems. The MITRE Corporation, Bedford, MA 01730, Electronic Systems Division, September 1984.

Van Cott, H. P., and Kinkade, R. G. (Eds.), Human engineering guide to equipment design. Washington, D.C.: American Institutes for Research, 1972.

Woods, D. D. "Cognitive technologies: "The design of joint human-machine cognitive systems." The AI Magazine, 6 (4), 1986, 86-92.

Zachary, W. "Beyond user-friendly: Building decision-aid interfaces for expert end users." Proceedings, IEEE International Conference on Systems, Man and Cybernetics, November 1985, 641-647.

THE EMERGENCE OF MULTI-USER EXPERT SYSTEMS

Michael S. Freeman
National Aeronautics and Space Administration
Marshall Space Flight Center

Expert systems technology has been one of the most written-about topics in computer science for the past five years. A great deal of research has been done on the selection of appropriate application areas (or "domains"), development of tools, classification of types of expert systems, and innumerable other related subjects. A wide variety of expert system architectures has been designed and analyzed. But there is one common characteristic in virtually every case: somewhere in the system description there is a conceptual diagram, with a component labeled "USER". Not "USERS", but "USER". Almost without exception, the assumption is made that the expert system will have a single user, at least at any given time.

There is a fairly straightforward explanation for this odd unanimity. Ludwig Wittgenstein, one of the great philosophers of this century, proposed in his "Tractatus Logico-philosophicus" that our thought is inherently constrained by our language [1]. Similarly, the research being done in expert systems has been to a great extent constrained by the primary vehicle available for pursuing it: the single-user Artificial Intelligence (AI) workstation. Even the purely theoretical research on expert systems has focused on issues derived from actual applications, which are selected for compatibility with "state-of-the-art" tools and methodologies.

This is understandable in that most organizations are still in the mode of "exploring" the utilization of expert systems. It is natural to want to show the new technology in the best possible light, and in the shortest possible time. Thus the problem chosen as a demonstration is usually one which fits nicely on a powerful new workstation, supported by the sophisticated new expert system "shells", in order to have the most positive impact on the managers who can authorize further development. The desire of the aspiring "knowledge engineer" to get his hands on that enticing new hardware and software may also be a factor.

Still, as was discovered in the development of an expert system to support the Analytical Integration of Spacelab Payloads, there exists a set of problems for which the single-user workstation is not a viable solution. In some cases, it may be necessary for the system to support numerous experts working on different aspects of a single logical activity simultaneously. In others, the activity may extend over a prolonged period of time during which different experts will work on different phases of the project to produce a single integrated result. Another possibility is that the results produced by the expert system must be continuously available to some group other than that actively exercising the expert system. These are some types of applications which require multi-user expert systems.

However, a key-word search of the National Technical Information Service for "multi-user expert systems", and variations thereon, produced no references over the past five years. Discussions with vendors of expert system building tools, as well as groups involved in expert system development, confirmed that this area of research is essentially a blank at this time.

The following definition of multi-user expert systems is based on experience in NASA domains, which usually involve a large and diverse set of engineering disciplines combined in a project organization.

> A multi-user expert system is one in which several users, having different skill levels, objectives, responsibilities, and types of authority, must jointly develop an integrated product through a single logical activity involving utilization of the expert system in multiple sessions, either sequentially in a non-contiguous manner or simultaneously.

Implicit in this definition is the possibility of simultaneous utilization of the expert system by different users, as well as by the same or different user at different times. Also, a multi-user expert system must provide for appropriate, limited access to any portion of the knowledge base by any user for differing purposes. These characteristics of multi-user expert systems strongly distinguish them from other types of expert systems.

These characteristics can be grouped into five primary areas in which they differ significantly from those of single user expert systems. It is not possible to present the research on which these areas are based in the scope of this paper, so they are simply listed below:

o   The sophistication required in the domain model.

o   The complexity and scope of individual rules.

o   The interactiveness of the rules.

o   The difficulty of enforcing knowledge base consistency.

o   The complexity of the explanation subsystem.

These differences are of degree rather than kind, except with regard to rule interaction and consistency enforcement. In the latter, additional constraints are placed on the generation of rules for multi-user expert systems which are not present in single-user expert systems.

It is the identification of additional applications which require multi-user expert systems, in which these special characteristics exist, that is expected to drive the development of expert system building tools for time-sharing systems. An understanding of multi-user expert systems should logically guide the development of expert system building tools in the time-sharing environment. This has not been the case to date. Expert system development environments such as ART and KEE are being ported to time-sharing environments with the apparent goal of duplicating, as far as possible, their original user interfaces and capabilities.

AI tools, even those implemented on time-sharing systems, are strongly oriented to the single-user class of expert systems. Practically all serious expert system development tools are implemented in the format of isolated, single-user AI workstations. Their products are essentially conceived to be stand-alone applications which read in the knowledge base, possibly interact with a single user, achieve some goal state, possibly update the knowledge base on disk, then terminate. AI tools ported to time-sharing environments have provided only minimal access to the special-purpose applications available in those environments to support multiple users. These include Database Management Systems (DBMS's), Forms Management Systems (FMS's), or even other languages. This is a severe handicap in developing multi-user expert system applications.

The AI tools which are available in the time-sharing environment required by multi-user expert systems have also, until very recently, been strongly research oriented and have not provided a practical delivery vehicle for commercial or operational expert systems. LISP, for example, has long been available on university computers in a time-sharing mode. But it existed in a multitude of dialects, each requiring extensive support environments which were poorly documented and subject to change at any moment since they were themselves treated as ongoing research projects. This meant that expert systems based upon them were not easily transportable, required a disproportionate share of system resources, were not well documented, and were subject to failure due to unplanned (and sometimes unsuspected) changes in the support environment. This could be accepted in constructing a system built on graduate student labor whose required life expectancy was Defense of the Dissertation plus a day, give or take a day. It is certainly not acceptable for a system in which an organization has invested a significant amount of limited resources to achieve a critical objective. AI tools are needed in the time sharing environment which are as reliable, well documented, well supported, and efficient as the software used to handle payroll processing.

Although the AI tools in the time-sharing environment have significant limitations at present, which pose serious obstacles to the development of multi-user expert systems, the traditional tools available there offer strong support to some requirements unique to this new class of expert system. Chief among these is the availability of very powerful Database Management Systems for manipulating and controlling the knowledge base. Standard DBMS functions are especially useful in this context for the following:

o   Access control at the Working Memory Element (WME) level.

o   Grouping of WME's for user-specific retrieval.

o   Simultaneous access to the knowledge base by multiple users.

o   Efficient storage and retrieval of large quantities of data.

o   Historical accounting of knowledge base usage.

499

Another very useful type of tool for the development of multi-user expert systems found on time-sharing computers is that used to develop sets of standard user interfaces (FMS and TDMS in the VAX/VMS environment, for example). Such tools support the rapid construction of shareable software and offer the following general advantages.

o   Adherence to guidelines which produce a user-friendly interface.

o   Predefined interfaces with other tools such as DBMS's.

o   Independence of the user interface from the expert system development process.

o   User interfaces which are more tightly integrated with operating system resources, and are therefore more efficient.

A third type of tool which can be especially effective in the development and operation of multi-user expert systems involves communication among users of the time-sharing system. During the development process, this can include the knowledge engineer and the discipline specialists who are contributing their expertise to the expert system. The discipline specialists can access the prototype, refine their inputs based on system performance, and efficiently communicate the revisions to the knowledge engineer by using the electronic communication tools available on the system (MAIL and PHONE in the VAX/VMS environment). Similarly, the knowledge engineer can easily construct very specific requests for clarification in the same fashion. An important side benefit is the record of such communications kept by the system.

Finally, the time-sharing environment required by multi-user expert systems typically provides access to a number of traditional programming languages. It may be possible to separate out functions which can be more efficiently performed by these languages. The time-sharing system provides tools for integrating such hybrid systems into a single executable module. These tools include support for explicit calling conventions, librarians, project management systems, linkers, and debugging systems. Given the current interest in utilizing traditional languages, especially C, for the development of AI-based projects, and integrating such projects into the mainstream of software development projects, this may turn out to be the greatest advantage of time sharing systems. The tools developed in the future to support construction of multi-user expert systems should provide easy access to traditional languages.

As has been discussed in a previous paper [2], the nature of NASA tends to produce expert system domains which are inherently multi-user. It is in these domains that this new class of expert system has first been encountered. As expert systems move into the main-stream of large scale engineering projects, these encounters will become more frequent. A need for expert system building tools which can accommodate the unique characteristics of these multi-user domains as they emerge is a problem which needs to be explicitly addressed as the current set of tools migrate to time-sharing environments, and as new tools are developed.

REFERENCES

1. Wittgenstein, Ludwig _Tractatus Logico-Philosophicus_ Routledge and Kegan Paul, London, 1961, Page 3.

2. Freeman, M. and Hooper, J. "Factors Affecting the Development of Expert Systems in NASA", Proceedings of _Artificial Intelligence - From Outer Space ... Down to Earth_, Huntsville AL, October 1985.

A Robotic System For Automation of Logistics N88-29407
Functions On the Space Station

J. C. Martin
R. B. Purves

R. N. Hosier
B. A. Krein

Boeing Aerospace Co.
Space Station
P.O. Box 1470
Huntsville, Al. 35807-3701

Westinghouse Manufacturing
Systems & Technology Center
9200 Berger Road
Columbia, Md. 21045

ABSTRACT

Spacecraft inventory management is currently performed by the crew and as systems become more complex, increased crew time will be required to perform routine logistics activities. If future spacecraft are to function effectively as research laboratories and production facilities, the efficient use of crew time as a limited resource for performing mission functions must be imployed. The use of automation and robotics technology, such as automated warehouse and materials handling functions, can free the crew from many logistics tasks and provide more efficient use of crew time. This paper focuses on design criteria for a Space Station Automated Logistics Inventory Management System through the design and demonstration of a mobile two-armed terrestrial robot. The system functionally represents a 0-g automated inventory management system and the problems associated with operating in such an environment. Features of the system include automated storage and retrieval (rack, drawer, and individual item), item recognition, two-armed robotic manipulation, and software control of all inventory item transactions and queries.

INTRODUCTION

Present day spacecraft require the crew members to perform logistics activities involving element storage, retrieval, redistribution, and logging. The crew time required to perform these tasks is significant compared to the time available to perform mission objective activities. Additionally, the problems associated with 0-g storage further complicates the attempt to achieve a controlled logistics system. Past experience with the Skylab missions show that items can and will be misplaced forcing long searches to be initiated which occasionally are completely unsuccessful. This wastes crew time and can jeopardize mission success. The Space Station presently being proposed by NASA will function as both a research laboratory and a production facility creating a substantial increase in the size and scope of the on-board logistics system. Present estimates indicate 100,000 items will be contained in the station inventory and resupply missions must be accommodated every 90 days for the life of the station. An inventory of this

magnitude is beyond the capability of the crew to manage efficiently and still meet mission objectives. Automation of the logistics system, or portions thereof, must be incorporated for the successful operation of the station both at IOC and during growth phases. A concept for on-board logistics automation is shown in Fig. 1. The system investigated in this paper is a 1-g simulation of this concept and demonstrates the feasibility and potential effectiveness of automated logistics handling on Space Station.

DESCRIPTION

The automated logistics system is being developed by Boeing Aerospace Company's Space Station Robotics Group with lead subcontractor support supplied by Westinghouse's Manufacturing Systems and Technology Division and secondary support from Transitions Research Corporation. The system design was driven by the requirement to show effective automated transfer and logging of elements within the module. The system, shown in Fig 2, consists of two six degree-of-freedom robots mounted on a rail based transporter within a half scale mockup of a radially configured Space Station logistics module. The objective is to achieve a relatively high fidelity system which could represent the concept of robots operating under 0-g conditions within a logistics module. Commercially available products and components were used where possible to reduce system cost and development time.

System operation consists of responding to requests for item storage, retrieval, or movement and manipulating the corresponding logistic elements to accommodate the request using vision system recognition and barcodes to locate items and verify their identity. Implementation of this system required the close coupled integration of robotic, vision, barcode, transporter, force sensors, and high level control subsystems (Fig. 3). Communication between these subsystems was the key element in the development process.

DESIGN

RACKS/DRAWERS/ITEMS

As mentioned above, the objective of the system is to represent 0-g operation within a module. In order to present the appearance of 0-g, the half scale foamcore racks and drawers are mounted horizontally to allow nonconstrained parts to fall, thus requiring all elements which are manipulated to be securely constrained at all times. Rack and drawer handles were originally designed to utilize front twist latches but cost and complexity forced use of non-operational handles/latches with magnetic latches in the rear. The design still represents the concept of mechanisms which are conducive to both robotic and human operation, allowing the crew to perform the necessary functions without special tools in the event of a system failure. Several generic storage boxes (styrofoam) and simple tools such as open end and crescent wrenches are used to represent typical on-orbit inventory items.

504

TRANSPORTER

A rail based transporter system is used to position the robots within the mock-up module. The transporter system consists of three main parts (1) a two robot carriage, (2) the transporter drive and positioning system, and (3) the drive controller with an interface to the system coordinator. The purpose of the transporter is to grossly position the robots in the area of their intended target and allow the robots, via vision and force detection, to determine the correct movements to accomplish the task. Such a system has the benefit of being able to operate in environments which are not rigidly defined or highly structured. Since the station configuration will be dynamic, the capability to determine the position of such things as a handles, latches, and parts within a drawer is considered a basic requirement.

ROBOTS

Two commercially available six degree-of-freedom (6 DOF), articulated joint, Puma 560 robots were selected for use in the system. The selection was based on the wide spread use of the Puma command language, VAL II, in production and developmental arenas and their 6 DOF articulated joint capability. The Puma's are mounted in the transporter carriage and physically perform the logistics element transfers. The carriage and rack/drawer size forced the robots to be mounted such that a significant portion of their work envelopes are overlapping, creating a significant coordination problem. As shown in Fig. 3, the Puma controllers are interfaced to the system coordinator via RS-232. Robot-to-robot messages are used to synchronize arm movements, thus providing pseudo coordinated motion. Due to the interface limitations of the controllers, machine code communications routines were written to supplement the VAL system.

VISION SYSTEM

An Automatix AV-5 vision system was chosen for use in the demonstration. The selection was based primarily on its powerful communications capability (six RS-232 control ports). The AV-5 contains two 68000 microprocessors, one dedicated to vision applications and the other to communications. The vision system is used to identify items within a drawer which are to be removed and to locate open areas within a drawer where items can be placed items without contacting the other contents. Prior to grasping an object, the vision system captures a frame from the wrist mounted camera on one of the robots, applies a weighted image selection criteria (area, parameter length, number of holes, etc.) to the captured areas, compares the image geometric parameters to previously trained templates, takes matched templates and determines object centroid and orientation, and determines correction coordinates for the robot to properly grasp the item. Additionally, the AV-5 serves as the system coordinator, integrating communications and control commands for all subsystem elements as shown in Fig. 3.

END EFFECTORS

The end effectors used on each Puma are different due to their

functional requirements. Ideally, both robots should be capable of performing all tasks, but for the purpose of the demonstration, it was felt that specialization was satisfactory. Puma #2 performs the dual role task of rack manipulation and item retrieval/storage while Puma #1 performs the similar tasks of rack and drawer manipulation. A parallel pneumatic gripper with integral barcode reader wand is mounted on Puma #1. The pneumatic gripper has Hall Effect sensors located in the fingers to detect head-on contacts. This gripper can read barcodes on a drawer or rack handle and then grip the handle using force sensing to determine the proper approach. Typically, pneumatic mechanisms are not used in space but it is felt that the concept of parallel operation can be adequately demonstrated with this lower-cost gripper.

The task of grasping individual objects and drawer/rack handles are quite different so the end effector on Puma #2 provides a servo-driven parallel gripper with automated finger change out capability. One set of fingers matches the drawer/rack handles while the others are smaller general purpose fingers for grasping individual inventory items. The servo gripper provides encoded position, velocity, and force parameters to its controller and gripper operation can be constrained by any of these three parameters. Additionally, the fingers of the servo gripper are spring loaded and coupled with a pressure sensor to determine head-on forces similar to the pneumatic gripper.

BARCODE READERS

Two barcode readers are incorporated into the system to verify element identities. Code 39 (3 of 9) barcode format was selected for use in the logistics system. Primary reasons for adoption of the Code 39 standard was the fact it is a DOD standard, full ASCII representation of the logistic elements can be used, and the easy access to commercially available hardware. A barcode wand is integrated into the fingers of the pneumatic gripper. This reader serves to check the rack and drawer codes prior to a transfer. A laserscan reader is used to verify the code on individual items and is capable of interpreting barcodes up to 18 inches away. The laserscan reader is used in conjunction with robot #2 and is mounted to a shelf attached to the transporter carriage. This seems to be a useful approach since item shapes will not be structured and conventional wanding may not be practicable. Both readers are interfaced directly into the AV-5 system coordinator (AV-5) via RS-232.

SYSTEM CONTROL

System control is accomplished through an Hewlett Packard 9836CS controller. The HP software, written in PASCAL, provides the operator with menu driven command screens for system level command and query. The HP controller represents the station Inventory Management System, where an integrated database providing complete information on all trackable inventory elements and corresponding element parameters should reside. Typical database information should include element ID, description, general classification, mass, volume, quantity, reorder limit, storage location, and process state. Through such a database, an astronaut can determine the status of all

inventory elements. Using operator selected menus, the system controller constructs and transmits the proper command strings to the system coordinator and provides the direct access and manipulation commands available to the operator. In addition to the crew interface, the system controller also serves as the link to integrate the logistics system to a higher level control such as the station Data Management System (DMS). This link serves to integrate the logistics system with other station functions and provides a means by which logistics transactions can be controlled without direct crew interaction. Data communications, to both the system coordinator and the high level control, are accomplished via RS-232 buses for this demonstration.

HIGH LEVEL CONTROL

The high level control for the Inventory Management System will most likely be performed by the station Data Management System. A VAX 11/750 represents the station DMS in this demonstration and serves to integrate the other station subsystems located in the BAC Space Station Labs. Using subsystem inputs, an on-orbit station scheduler can direct the Logistics System to perform activities at scheduled times without crew interaction. High level control, such as the station DMS, should have complete access to the logistics database to supplement station scheduling and to initiate resupply requests to ground control.

OPERATION

As a whole, system operation is quite successful, demonstrating automated identification, storage, and retrieval of inventory items and drawers. Inventory redistribution is also accomplished through a combination of existing software commands, allowing items to be moved from drawer to drawer and complete drawer units to be exchanged within the rack structure. This simulates the capability to automate module mass redistribution on-orbit. System control is successfully implemented from the HP computer.

Several problems have been encountered during the development stage. Most of the problems stem from the Puma robots. The rack/drawer removal and store sequences require the robots to perform a straight line move followed by a shallow arc. The VAL II kinematic solutions to these moves are awkward, causing the robots to use all six axes when only three are required. The movement of these extra joints is not due to hardware limitations and results in the two arms crashing together. Some creative programming was required to work around the VAL kinematics. Another significant problem was created by the lack of an easy way to interface VAL to external controllers (i.e. the AV-5). A machine code communications program was developed to establish the needed links. Several small problems were corrected such as the modification of rack and drawers slide rails to incorporate more compliance during storage sequences and reduce overall weight.

CONCLUSIONS

The system operation to-date is encouraging. It appears that automated inventory management using robotic manipulators can be achieved on Space Station with minimal or no advances in present day robotic technology, though machine vision still needs development. Further development is required on this system before specific design guidelines for station planning can be determined, but several general guidelines for station scarring have surfaced during system design and debug activities. Station designs should include (1) subsystems designed for both human and automated (robot if applicable) operation, (2) incorporation of compliance in mating parts (i.e. drawers, racks, etc.), (3) barcode and lettered labeling of all station elements, (4) visual orientation cues attached to station elements and structure, (5) lighting systems which provide bright, evenly distributed illumination and utilize flat finish, high contrast surfaces in areas where machine vision is to be used, (6) adoption and use of standard storage containers which can be easily distinguished and manipulated by automated techniques, (7) an IVA track/transporter system to assist the crew in large payload manipulation and robotic system mobility, (8) inclusion of integrated force/torque and tactile sensors to aid compensation of robot and environment inaccuracies, and (9) the use of proximity sensors coupled with force/torque, flexible arm construction, and slow robot motions, to provide a safe working environment for station crew.

If Space Station is to benefit from the productivity gains A&R technology can provide, appropriate hooks and scars must be incorporated at IOC or during early assembly because modification on-orbit will no doubt be extremely costly if not impossible to accomplish.
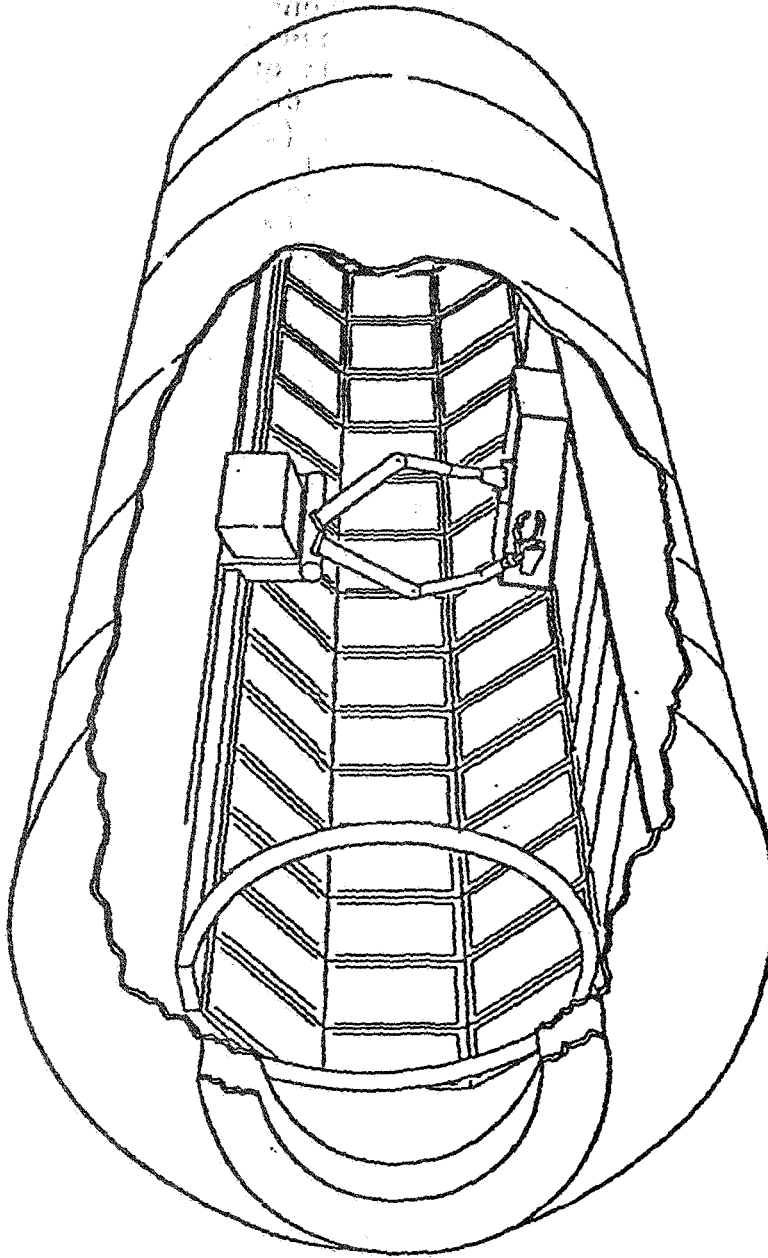
SPACE
STATION

Figure 1: Conceptual System

ROBOTIC LOGISTICS SYSTEM
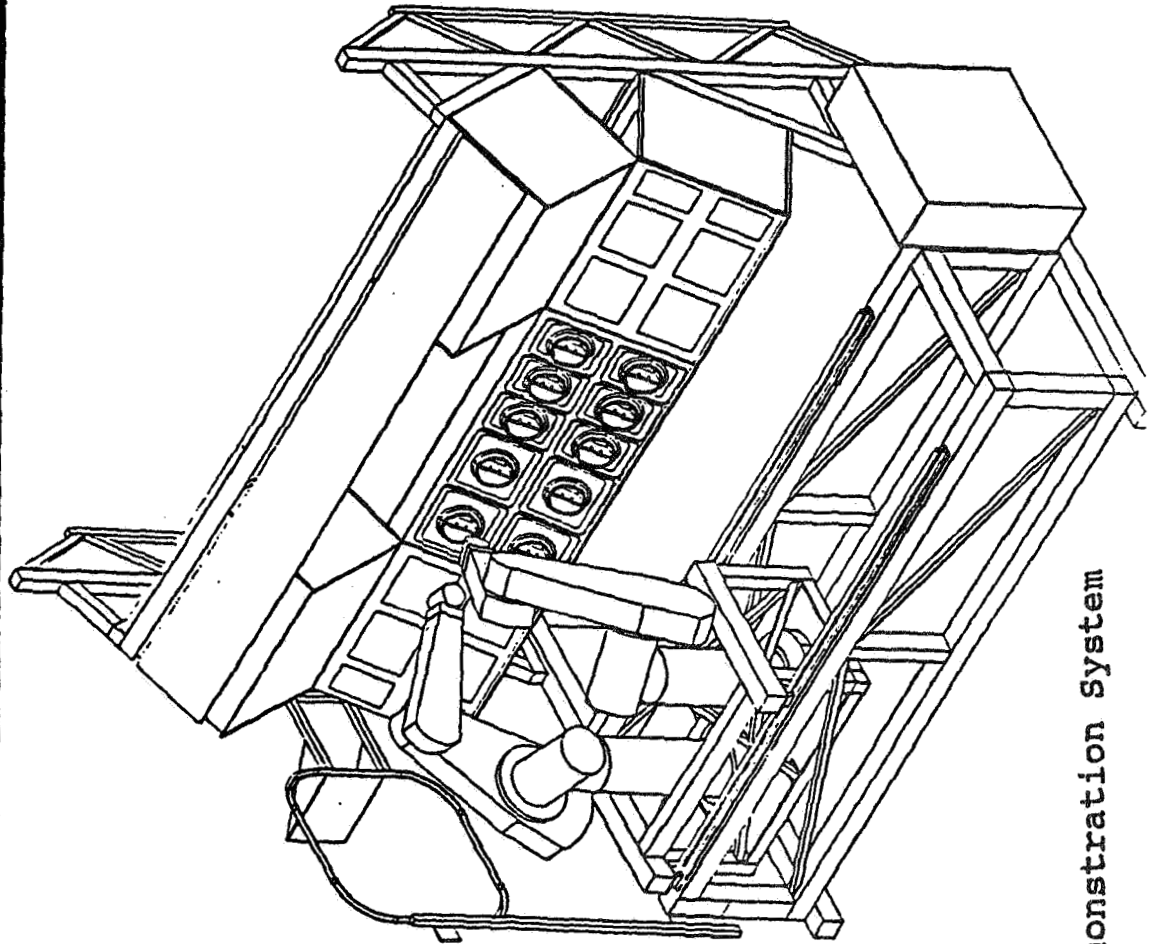
Figure 2: Demonstration System

510

ROBOTIC LOGISTICS SYSTEM
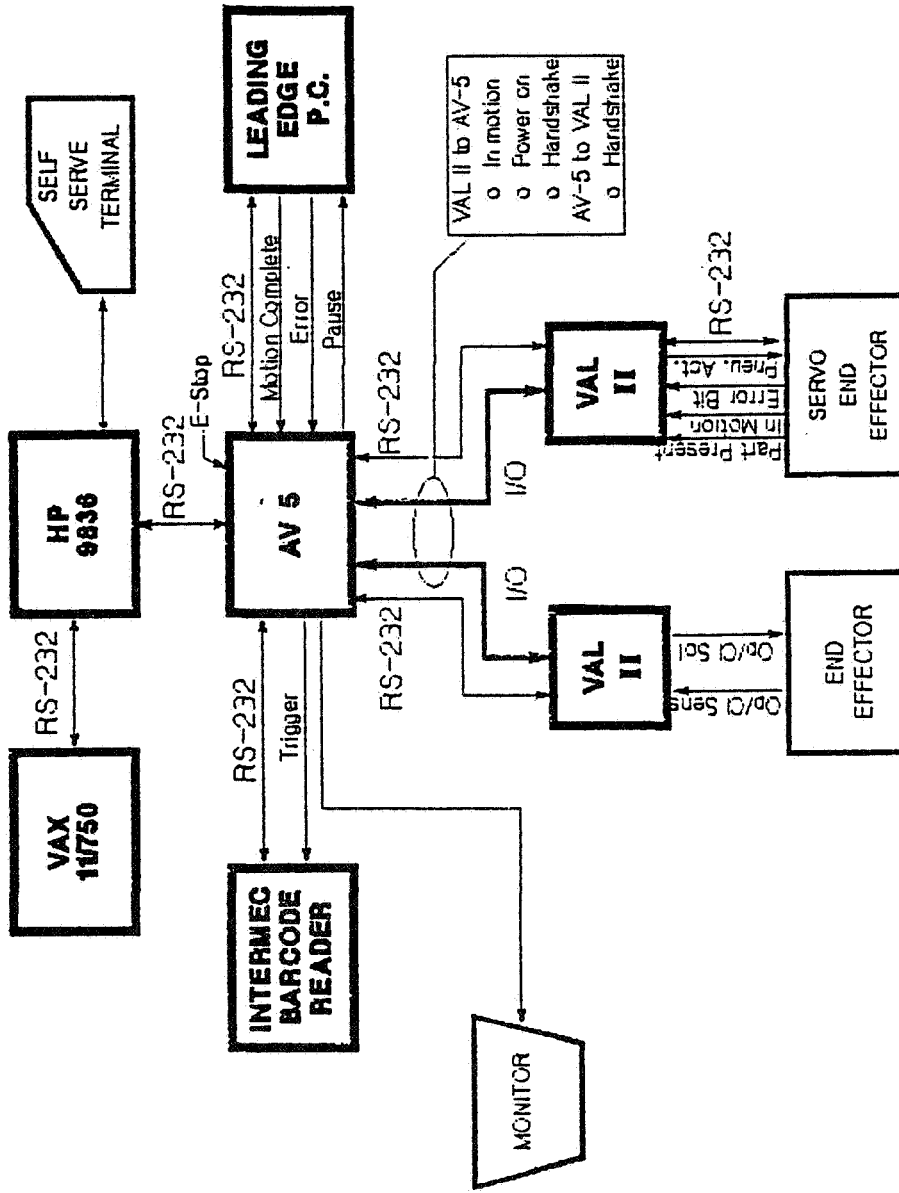
Figure 3: Demonstration System Architecture

PERSONNEL OCCUPIED WOVEN ENVELOPE ROBOT

Francis Wessling, Ph.D.
William Teoh, Ph.D.
M. Carl Ziemke, P.E.

1.0 Introduction

The Personnel Occupied Woven Envelope Robot (POWER) project is a joint effort of the Johnson Research Center at the University of Alabama in Huntsville (UAH) and Wyle Laboratories, also of Huntsville, Alabama. This work is being performed under the Innovative Research Program for the National Aeronautics and Space Administration (NASA) Office of Space Science and Applications in Washington, D.C.[1] This is a two-year effort, of which a little less than one year has been completed.

The purpose of POWER is to provide an alternative to extra-vehicular activities (EVA) of space-suited astronauts and/or use of long slender manipulator arms such as are used in the shuttle RMS (remote manipulator system).[2] Both of these existing methods for performing work around the U.S. space station have their limitations. POWER provides the capability for a shirt-sleeved astronaut to perform such work by entering a control pod through air locks at both ends of an inflated flexible bellows (access tunnel). The exoskeleton of the tunnel is a series of six degrees of freedom (Six-DOF) articulated links compressible to 1/6 of their fully extended length. The operator can maneuver the control pod to almost any location within about 50 m of the base attachment to the space station (Figure 1). POWER can be envisioned as a series of hollow Six-DOF manipulator segments or arms wherein each arm grasps the shoulder of the next arm. Inside the hollow arms is a bellows-type access tunnel. The control pod is the "fist" of the series of linked hollow arms. The "fingers" of the fist are conventional manipulator arms under direct visual control of the nearby operator in the pod. This configuration allows many applications, which include:

1. Changing out and servicing payloads on the Power Tower payload platform.
2. Maintaining subsystems such as propulsion and attitude control.
3. Providing satellite and free flyer service.
4. Performing immediate fly around inspections.
5. Supporting shuttle cargo bay operations.
6. Performing remote control operations for hazardous duty.
7. Capturing satellites during final approach.

Wyle has been studying the need for space related services in the commercial sector.[3] POWER appears well suited to many of these applications.

2.0 Desirabilty of Space Station Robotics

POWER is being developed as part of a general NASA interest in the use of automation and robotics as applied to the U.S. space station and projects beyond it.[4] This interest follows consideration of the results to date of the first quarter century of the U.S. manned space program. Much has been accomplished but new challenges are still ahead. Just as "space age" technology has been spun off to numerous earth applications, we must now utilize applicable terrestrial high technology in our upcoming space endeavors. This implies the

513

use of teleoperation and robotics. Many writers use the words teleoperator and robot interchangeably. The word robot actually implies pre-planned or autonomous action. However, a teleoperated robot or telerobot is also possible. Such a device could employ a man-in-the-loop who is assisted by certain specialized computer programs in directing a mobility platform and/or operating manipulator arms. POWER is considered to be such a telerobot.

Recent events have reminded all of us of the potential hazards of space flight and orbital operations. Robots, teleoperators and telerobots can limit these hazards both by reducing crew sizes and by reducing hazardous exposure of the crews that are in orbit. Recently, the first steps of assembly of large space structures has been initiated in the EASE/ACCESS shuttle mission 61-B.[5] This mission required two astronauts for EVA plus a third person part of the time operating the RMS arm. It may be possible for one shirt-sleeved astronaut inside the POWER control pod to perform similar assembly tasks now requiring up to three astronauts, two of whom must wear space suits and use manned maneuvering units.

3.0 Related Earlier Developments

3.1 The Serpentuator

A study of past related developments in expandable/collapsable space structures revealed several related developments. One of the first devices considered was the NASA/MSFC "Sepentuator" patented in 1970 by H.F. Wuenscher.[6] As shown in the patent disclosure, the Sepentuator was envisioned to serve some of the same purposes as POWER but on the orbital workshop (later renamed "Skylab"). The Serpentuator was a snake-like multi-jointed device with 18 three-foot-long electromechanically actuated links. Multiplanar movements were possible because each link moved $\pm$ 20° about a hinge axis at 90° to that of the adjacent link. Also, the base section contained a rotatable joint. Apparently, the Serpentuator was intended to support, move and retrieve objects in space because little was said about the use of end effectors or tooling on the free end. The Serpentuator was never used in space and would have required considerable additional work in the area of control algorithms and mechanical strength in order to be operated with precision. Also, the lack of position feedback and the difficulties involved with distant viewing of the movement of the tip appear to be serious operational problems. Review of this device did not reveal any features useful for the POWER design.

3.2 The Astromast

The Astromast is a trademark of the Astro Research Corporation, a subsidiary of Spar Aerospace Products, Ltd.[7] When extended, it is a triangular cross-section space frame similar to a tall, commercial radio or television station antenna mast. However, because of its unique linear lattice construction, it can be twisted like a helical spring about its long axis and stored with great compactness. The Astromast comes in two types. One has continuous longerons that are elastically coiled for storage. Thus, a controlled release of their stored spring energy permits them to self-deploy. The other type of Astromast has hinged, articulated longerons and is not self-deployable. These structures were interesting but not directly applicable to the POWER concept.

3.3 The Flattened Tube (STEM)

A flattened tube that can be stored as a tape and, when released, springs into the shape of a slit cylinder of indefinite length was another early candidate as a design element. These tubes called STEMs for Storable Tubular Extendible Member have been known for 20 years.[7,8] Their compactness is remarkable. A mast made from such a slit tube could be stored on a reel with a diameter 1/20 of the extended tube height. There is a more rigid version called the BI-STEM in which two slit tubes nest inside each other to provide an extended tube that approaches a non-slit tube in stiffness and resistance to buckling. This concept appeared to have merit for POWER.

3.4 The Collapsible Tube Mast

The European Space Agency (ESA) is developing two different mast concepts.[9] The collapsible tube mast (CTM) is made from a tube formed from a pair of thin springy shells. They can be manufactured from metal or composites and are formed into a biconvex shape. The two halves are then bonded at the edges by welding or with adhesive. The formed tube can subsequently be flattened and rolled up into a small volume around a drum. It springs back to its original shape upon unrolling and then behaves like a closed beam. This idea appears to be an extrapolation of the Mars Viking surface sampler boom.[10] The Viking Lander descended safely to the surface of Mars in 1976 and obtained soil samples by means of 136 inch (3.45 m) long fully retractable boom. The boom consists of two flanged half-cylinders of stainless steel sheet welded together at the flanges. The resulting hollow tube carries a flat-pack cable system inside it and is stored flat on drums similar to the CTM concept.

3.5 The Extendable Retractable Mast (ERM)

A second ESA development for a mast is the Extendable Retractable Mast (ERM). The ERM consists of a set of telescoping tubes made from carbon fiber reinforced plastic. The thin walled tube segments are manufactured using wound carbon fibers. Longitudinal stringers support bending forces, and circumferential hoops give added support to the ends of the tubes. The deployment and retraction mechanism consists of a central threaded spindle and a threaded nut on each tubular section's base. The mechanism is designed so that only one nut contacts the threaded portion of the spindle at any time. A latching mechanism secures two tubes together when extended and a separate mechanism allows the nut of the next tubular element to be engaged by the spindle. This concept could serve as a linear actuator for POWER, if necessary.

4.0 POWER Mechanical Design

Given the background of literature search work briefly described in section 3.0, the basic mechanical design of POWER proceeded in at least two basic parallel paths. One general direction was to design a system of multiple linked segments based on a structure with four slit-tube (STEM type) longerons. The other principal approach was to use mechanical actuators as the movable structural members. Before these general approaches were selected, several other alternatives were considered, including, scissors jacks, reel and cable systems, hydraulic or pneumatic jacks, and a telescoping central jack in support of a pivot arm. In all, 14 different systems were considered and several were tested

in model form. In order to help select the best candidate designs from among 12 segment-type candidates, matrices of advantages and disadvantages were constructed (Tables I, II and III). With these matrices, supplementary performance calculations and review of models of candidates 2, 3, 6, 11 and 12, the choice was narrowed to two basic approaches.

4.1 Tape Measure (STEM) Design

Both final segment design contenders were expected to meet the criteria of:

o Six DOF movement of flanges
o Compression ratio approximately 6:1
o Maximum tilt angle about longitudinal axis of at least 30°
o Segment position determinable from actuator feedback

The STEM approach (simulated by curved tape measure blades) consisted of two plates or flanges (each one common to the next segment) connected by four tapes positioned at 90° increments around a base circle (Figure 2). The tapes are kept in tension by the inflated central bellows. The curved tapes in tension provide stiffness against unwanted rotation not available from cables. The tapes were rolled flat on take-up reels. This system was math-modelled and evaluated with respect to several criteria including fundamental frequency and maximum axial, shear and bending loads as well as total segment weight and load carrying ability. Based on this analysis, the concept was abandoned.

4.2 Selected Segment Design

Figure 3 is a drawing of the selected POWER segment design. The central bellows-type access tunnel is not shown. The linear actuator chosen is a 24 Volt d.c. device capable of 12 in. (46 cm) extension with a maximum force of 1000 lb. (4448 N). Load calculations by Wyle revealed that this force was more than twice as much as was needed to manuever a four-segment model on earth. It was not considered practical to demonstrate POWER under normal gravity with more than about four segments total length because of the unrealistic gravity loading as compared to space conditions. However, the full scale, complete POWER system will be able to withstand launch stresses. The linear actuators are arranged in pairs with three sets of common mounting points equidistant on each segment flange. Thus, these flanges form equilateral triangles with actuator twin mounts at each vertex. The whole arrangement with hinge-pin actuator attach points approximates a six-DOF table. The opposing flanges can nutate with respect to each other but not rotate. The actuators lock during power off and provide potentiometer type position control. Current draw at maximum load is 4.8 Amperes each.

5.0 Control System Design

The POWER Control System design strategy was selected after several different approaches were considered. One approach was to move the control pod to a target so as to minimize compression work on the pressurized bellows. This compression work will normally exceed the friction and inertia work, both of which can be reduced to small values by good bearings and slow operating rates. However, allowing the bellows to expand upon extension without maintaining one atmosphere of air within it, reduces the compression work upon contraction.

Thus, minimizing the work of compression as a control design consideration was ignored. POWER lacks an accurate external position sensor in the control pod so the unit will be manuevered in an open loop fashion under continuous human control. The control mode will be positional rather than rate because of the large masses involved. All strategies discussed herein assume motion commencing from the contracted (parked) position of the column of segments.

Three control strategies have been considered to date: the tip-biased, the base-biased and the equal biased methods. In the tip-biased strategy, more preference is given to move those segments that are closest to the free end of POWER. The main advantage of this scheme is that less mass must be move at any given time, thereby minimizing the power consumption, and thus, the dynamics of the body are easier to handle. The main disadvantage of this scheme is that the control equations turn out to be very complicated, and other constraints are needed to permit unique solutions. The base-biased strategy is similar to the tip-biased one, except that preference is given to those segments that are closest to the base (the fixed end) of POWER. This strategy suffers from the disadvantage that any movement involves the entire column, and is expensive from the viewpoint of energy consumption.

The last strategy is the equal-biased scheme. In this scheme, equal emphasis is given to all segments. Further, whenever possible, each segment will move by the same amount. This scheme does not look very attractive at a first glance until one examines the equations of motion. The solution of the equations of motion shows a high degree of symmetry such that they can be generalized to any number of segments. Also, the same logic can be used (in a somewhat unusual way) to constrain portions of the column to assume an arbitrary shaped curve. This consideration is very significant since it is directly related to strategies for obstacle avoidance. A scheme has been developed in which obstacle avoidance can be implemented as part of the control equations, based on the assumption that these obstacles, if present, are fixed (not moving) relative to the base of POWER. The control strategy is now being implemented and tested at UAH through simulation studies with graphics output.

6.0 Conclusions

Progress to date on POWER has resulted in selections of a mechanical design that meets all of the original criteria for the project. The most complex parts of the system (actuators and controls) are shelf-available items. This does not necessarily mean that full-scale versions of these parts would be operable in space but this does not appear to be a serious design problem. Plans for the future include building and testing a four-segment powered model as part of the demonstration of the control algorithms, which are being developed. Beyond this, there is the selection of materials for the tunnel, and conceptual designs of the control pod and mission planning of applications of POWER.

517

REFERENCES

1. F. Wessling and G. Bakken, "Personnel Occupied Woven Envelope Robot (POWER)", The University of Alabama in Huntsville and Wyle Laboratories, June 28, 1985.

2. H. S. Taylor, "A Large-Scale Manipulator for Space Shuttle Payload Handling - The Shuttle Remote Manipulator System", Amer. Nuc. Soc. 26th Conference on Remote Systems Technology, 1976.

3. D. L. Christensen and J. W. Monroe, "Scenario for Comercialization of Space-Related Services", Wyle Laboratories Research Report No. WR84-07.

4. "Advancing Automation and Space Technology for the Space Station and for the U.S. Economy", NASA Tech. Memo 87566, April 1, 1985.

5. "EASE/ACCESS Postmission Management Report", Spacecraft Payload Project Office, NASA/MSFC, Huntsville, AL 1986.

6. H. F. Wuensher, "Sepentuator - U.S. Patent No. 3,520,496", Issued July 14, 1970.

7. R. Kumar and S. Ahmed, "Some Boom Choices for Design of Deployable Solar Arrays", Solar Energy, Vol. 16, pp. 159-163, 1974.

8. G. Wolf and A. Wittmann, "The Flight of the FRUSA", AIAA Paper No. 72-510 presented at the AIAA 9th Electric Propulsion Conference, April 17-19, 1972.

9. M. A. Aguirre Martinez, "ESA Sponsored Developments in the Field of Deployable Masts", European Space Agency Journal, Vol. 9, No. 3, pp. 313-320, 1985.

10. D. S. Crouch, "Mars Viking Surface Sampler Subsystem", 25th Conference on Remote Systems Technology, Amer. Nuc. Soc., 1977.

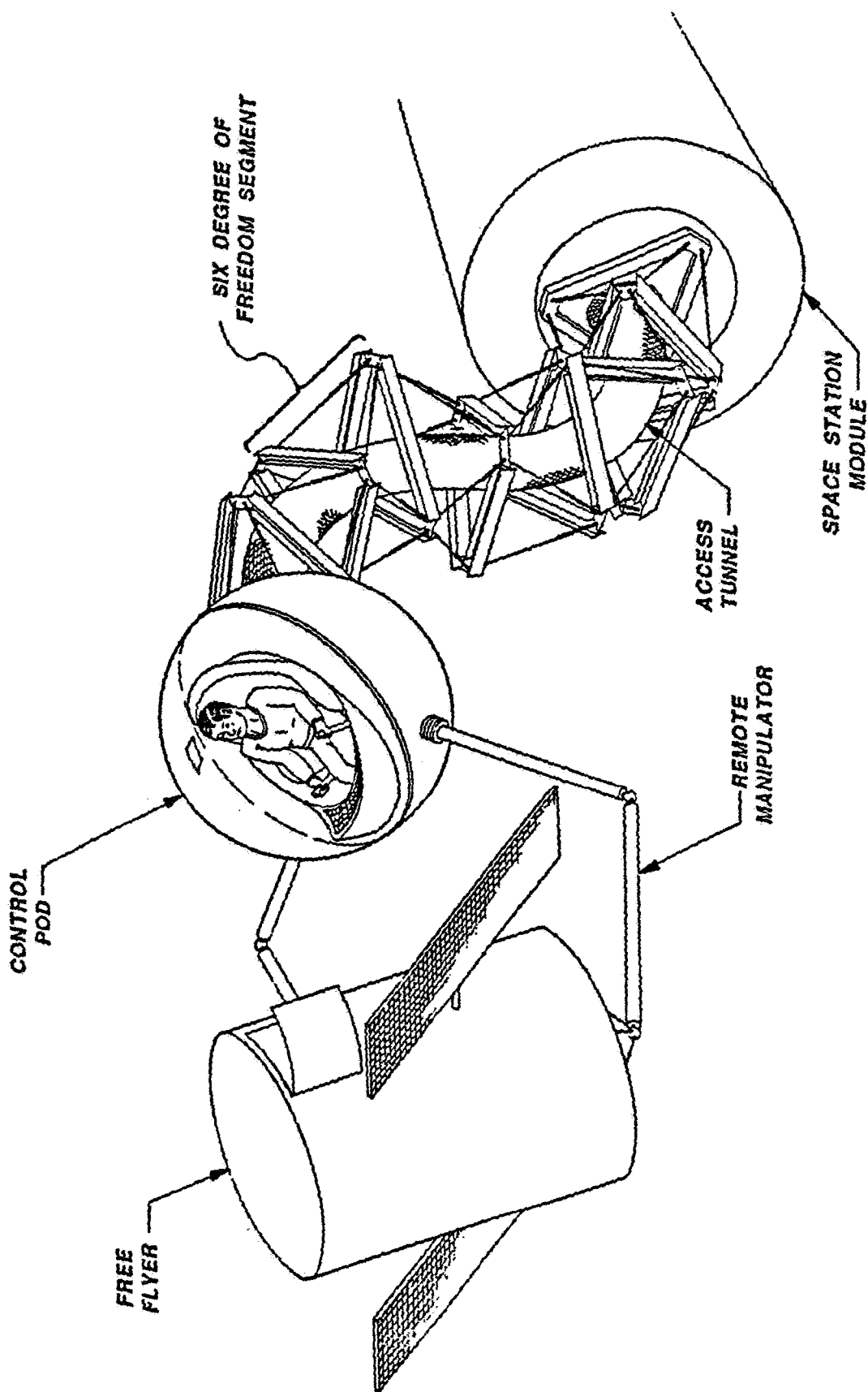<div align="center">TABLE I POSSIBLE MECHANISMS FOR POWER</div>

1. Cables originating at the base of the boom, threading through other segments.
2. Cables originating at the base of each segment.

At each segment:

3. Linear actuators.
4. Scissors jack.
5. Hydraulic or pneumatic jacks.
6. Telescoping central jack and support of pivot arm.
7. Scissors jack and hinge segments.
8. Worm screws and motors at each segment.
9. "Camper Jacks".
10. Scissors jack, hinge segments and ball joints.
11. "Tape measure" curved steel tapes (precurved struts) instead of cable.
12. Six degree of freedom table with six legs.

<div align="center">TABLE II ADVANTAGES OF TABLE I MECHANISMS</div>

ADVANTAGES	DESIGN MECHANISMS											
	1	2	3	4	5	6	7	8	9	10	11	12
Modular Construction		X	X	X	X	X	X	X	X	X	X	X
Positional Freedom		X	X	X	X	X	X	X	X	X	X	X
Incremental Positioning	X	X	X	X	X	X	X	X	X	X	X	X
Definite Positioning		X	X	X	X	X	X	X		X	X	X
Existing Technology		X	X	X	X	X	X	X	X	X	X	X
Structural Rigidity		X	X	X	X		X	X	X	X	X	X
Rotation Measurable		X	X	X		X	X	X	X	X	X	X
Extension Measurable		X	X	X	X	X	X	X	X	X	X	X
Position Independent of tunnel pressure		X	X	X	X	X	X	X		X	X	X
Control Solvable	X	X	X	X	X	X	X	X	X	X	X	X
Number of Activators per segment	3	3	3	2	3	2	6	3	3	12	4	6
Degrees of Freedom per segment	3	3	3	1	3	2	3	3	3	6	3	6

<div align="center">TABLE III DISADVANTAGES OF TABLE I MECHANISMS</div>

DISADVANTAGES	DESIGN MECHANISMS											
	1	2	3	4	5	6	7	8	9	10	11	12
Cable Stretch	X	X										
Position-Force Dependent	X	X										
Limited Maneuverability	X											
Seals Problems					X							
Blocks Tunnel Center						X						
Heavy Construction				X		X				X		
Friction Susceptible	X	X				X			X			
Bending on Members	X		X		X	X		X	X		X	
Tunnel Puncture Danger	X	X	X					X				
Position Indefinite	X	X							X			

SIX DEGREE OF FREEDOM SEGMENT

CONTROL POD

FREE FLYER

ACCESS TUNNEL

SPACE STATION MODULE

REMOTE MANIPULATOR

P.O.W.E.R.

(PERSONNEL OCCUPIED WOVEN ENVELOPE ROBOT)

Figure 1

WYLE LABORATORIES
SCIENTIFIC SERVICES & SYSTEMS GROUP
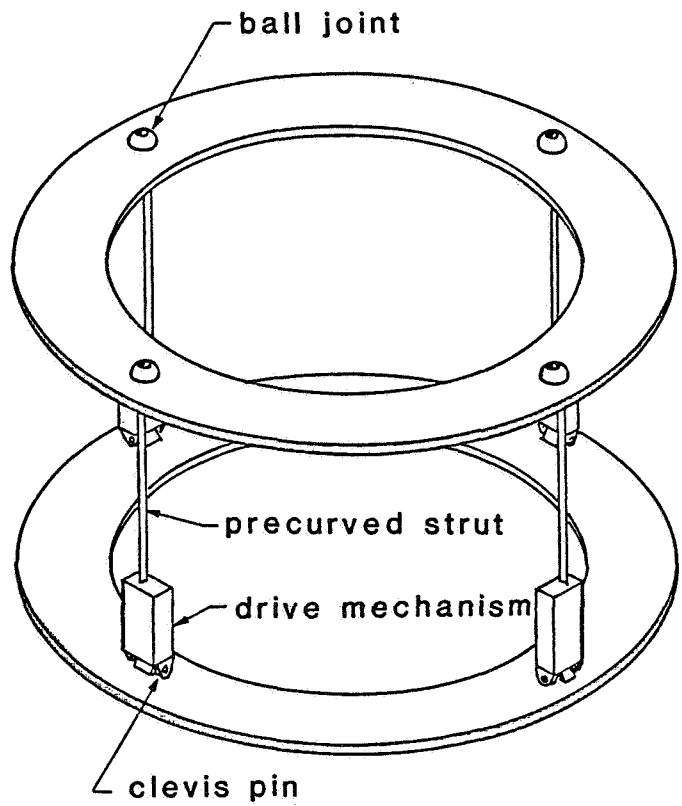
The University Of Alabama In Huntsville
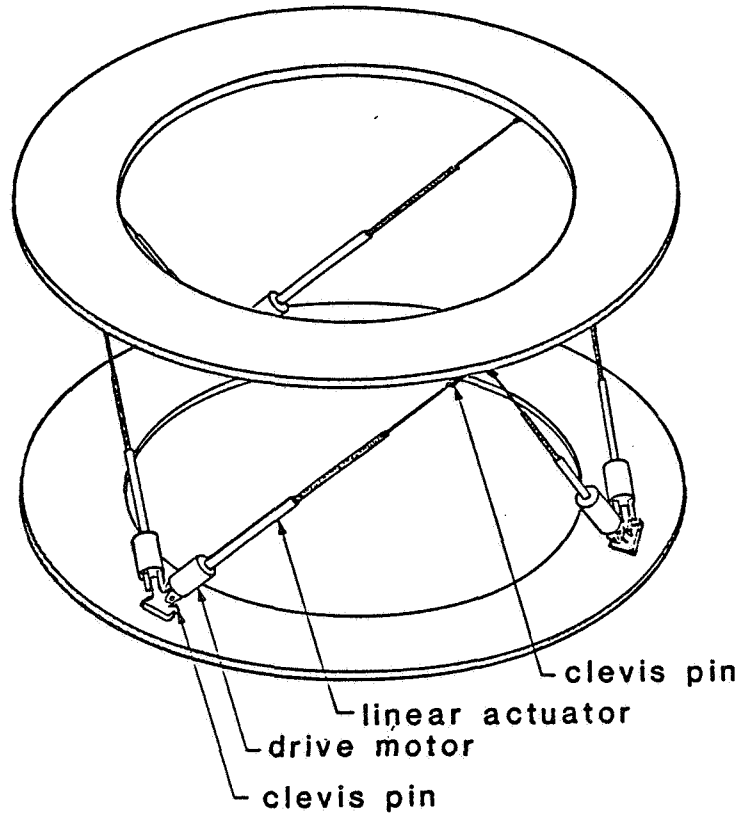
520

Figure 2 "STEM" Mechanism

Figure 3 Six Actuator Mechanism

521

REMOTE SERVICING OF SPACE SYSTEMS

S. L. Collins
R. B. Purves

Boeing Aerospace Company
Space Station Program
P.O. Box 1470
Huntsville, AL 35807-3701

ABSTRACT

Space systems are difficult to maintain on orbit. The difficulty arises from the limited ability and availability of the astronaut work force in the hazardous space environment. Remote robotic manipulation can free the astronaut from the hazardous working environment while also increasing the work force. However, remote robotic servicing is not without its own set of problems and limitations, such as communication time delay and unstructured worksites. This paper describes tests and test equipment designed to increase our understanding of the remote servicing problems and to allow development of potential solutions. A half scale satellite mockup has been developed for evaluating and improving upon the design of replaceable subsystems, such as batteries and electronics boxes. A servicer system, that includes a six degree-of-freedom PUMA 560 robot and interchangeable end effectors (tools), has been developed to aid in driving out servicer design requirements. The results include the time delay impact on servicing timelines and requirements for the servicer system.

INTRODUCTION

The Space Station represents the advent of a new era in space, routine on-orbit servicing and maintenance of space systems. The permanently manned Space Station will consist of many systems that must be repaired, maintained, or replaced on-orbit. The Station will also function as a base for satellites and other vehicles that will require servicing or maintenance.

The repair of the Solar Max satellite demonstrated and verified the concept of EVA (extra vehicular activity) service and maintenance. However, the use of EVA is greatly limited due to many factors such as a small working crew and the hazardous working environment of space. The use of remotely controlled robotic systems for many servicing tasks can free the crew from the hazards and increase the available work force. Remote robotic servicing is, however, not without its own set of problems and limitations, such as communication time delay and
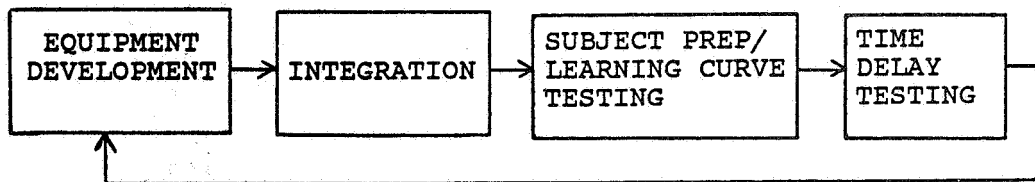
unstructured worksites.

Satellite servicing tasks typically require manual control over automated control due to the many unique tasks [TRW, 1984]. These unique tasks represent the majority of the tasks required during the early phases of the Space Station. It does not appear to be cost effective to structure these unique tasks for automated servicing. Unstructured worksites drive the robotic servicer design away from automated control towards manual control.

Conversely, the communication time delay from the ground control station to the robotic servicer through TDRSS(tracking and data relay satellite system) drives the robotic servicer design towards more automated control. The maximum round trip communication time delay was estimated to be slightly greater than 1 second [Wetherington, 1974]; however, more recent studies have indicated that the delay may be as long as 8 seconds.

A study, summarized in this paper, has been conducted to increase our understanding of the remote servicing problems and to allow development of potential solutions. Typical servicing tasks with time delays from 0 to 8 seconds were included to determine design requirements/improvements in the replaceable subsystem and robotic systems (manipulator, end effectors, and the man/machine interface) and to determine the impact of time delay on servicing timelines.

APPROACH

The study has been broken into four major phases.

| EQUIPMENT DEVELOPMENT | → | INTEGRATION | → | SUBJECT PREP/ LEARNING CURVE TESTING | → | TIME DELAY TESTING |

The development phase included the design and fabrication of all required hardware and most of the software. Some integration software was also developed in this phase. The integration phase focused mainly on integrating the manipulator system with the man/machine interface. Testing began with subject preparation. Subject preparation doubled as learning curve tests. The final phase of the iterative process was time delay testing. After time delay testing, the equipment will be improved and testing will continue.

EQUIPMENT DEVELOPMENT

Satellite Mockup

The satellite mockup (Fig. 1) represents an unstructured worksite. The mockup contains four half scale, space qualified type, replaceable subsystems:

1) Flight Guidance Equipment
2) Rate Sensing Unit
3) Battery
4) Multimission Modular Spacecraft Module (MMS)

These replaceable subsystems (Fig. 1) are attached to the mockup by different fasteners and in different orientations. Obstacles can be located to increase the difficulty of changeout of the subsystem. The mockup allows for approximately 3 feet travel of the subsystems in vertical position and 360 degrees in rotation.

Robotic Servicing System

The robotic servicing system consists of two major systems, a manipulator system and a man/machine interface system. The manipulator system is represented by a Unimation Puma 560, a set of end effectors, and three video cameras. The Puma 560 is an electrically powered, 6 degree-of-freedom robot shown in Figure 2. The Puma 560 was selected because it meets the reach envelope and degree-of-freedom requirements. The end effectors are tools that attach to the end of the robots last flange. They have been designed to accomplish several selected servicing tasks. The end effectors that have been developed consist of a MMS tool, a gripper, and a hexdriver, shown in Figures 3 -5. The gripper has been designed with individually compliant fingers. Each end effector is electrically powered and mates to a common interface, Figure 6. The common interface allows for manual end effector changeout without the need for additional tools, such as a wrench or screwdriver. An interface that allowed for automatic end effector changeout has not yet been developed due to the desire to minimize the weight extending from the Puma 560's joint six flange.

The man/machine interface system is a workstation as shown by Figure 7. The workstation consists of three video monitors, a robot control CRT and keyboard, a video switcher, and controllers for the robotic system. The workstation can be easily reconfigured to accommodate different types of operators, hand controllers, and/or tasks. Software has been developed to allow for time delayed joystick control of the manipulator, voice control of the cameras, and data acquisition.

INTEGRATION

The workstation and manipulator systems are quite complex systems alone; thus, the integration of these two systems further complicate matters. The workstation and manipulator systems are physically separated. The systems lie in adjacent rooms (Fig. 8). The controlling devices are connected to the manipulator systems.

Manipulator System	Controller
Robot	Joysticks
End Effector	Switches
Cameras	Voice Activation

525

The joysticks are hooked into the manipulators computer/controller, which displays information onto the workstations terminal. The monitors are tied into a video switcher that allows for input selection, from any video camera or workstation video tape recorder. The video cameras are controlled from the workstation with a voice activation system. The workstation must accommodate these control and feedback devices while also accommodating the human operator in their use.

SUBJECT PREPARATION/LEARNING CURVE TESTING

Subjects

Six Boeing employees were chosen as subjects to participate. They have no prior experience with hands on manipulator control and they have normal or corrected visual acuity as determined by a Class II Flight Visual exam. Two subjects are female and five of the six are engineers. In addition to the six subjects, two SKYLAB astronauts also participated.

Apparatus

The robotic servicing system was used, along with the workstation for the subject preparation/learning curve testing. Visual feedback through the video monitors was the only feedback provided to the subjects. One video camera was attached to the end effector and the other two cameras are positioned one to the left and one to the right of the robot. Two hand controllers, 2-three degree-of-freedom joysticks and a teach pendant were used for hand controller comparison. The gripper end effector was used. Blocks, pegs, and balls were the objects used in four pick-up and place tasks.

Procedure

The subject was seated at the workstation, isolated in the control room. The subjects performed one of four tasks using one of the two hand controllers. For example, a subject used the teach pendant to pick up a block from a top a shelf and place it on a target on a lower shelf. Each attempt was considered a run. The subjects were not allowed to manipulate the cameras. In all, the subjects completed approximately 400 runs and approximately 80 hours of testing.

Results

A typical subject learning curve is shown in Figure 9. The two lower curves represent subtasks and their total represents the top curve. An abort represents a dropped or knocked over object or other subject caused anomaly. The subjects unanimously selected the joysticks as the hand controller they would choose to attempt a difficult task. The data also backed up their choice. The data showed that more learning occurred while the subjects used the joysticks. Faster performance times were also generated using the joysticks. The data also indicated that the subjects were prepared to perform more

difficult tasks.

Many subjective finding were made. The subjects tended to rely on a particular view, one of the three provided. They often failed to see important information displayed in the other views. Also, each subject consistently performed within his personality. For example, some individuals are perfectionist, or risky. These qualities in their personalities were consistently apparent in their approaches to the problems created by the tasks.

TIME DELAY TESTING

Subjects

The same six subjects that participated in the preparation/learning curve tests were used in the time delay tests. The subjects had a minimum of thirteen hours of manipulator control experience.

Apparatus

The robotic servicing system was used, along with the workstation for the time delay testing. Visual feedback through the video monitors was the only feedback provided to the subjects. One video camera was attached to the end effector and the other two cameras were positioned one to the left and one to the right of the robot. One hand controller, 2-three degree-of-freedom joysticks, were used for controlling the manipulator system. The tasks included the changeout of the replaceable subsystems on the satellite mockup. The end effector was dependant on the task.

Procedure

The subject was seated at the workstation, isolated in the control room. The subjects performed the changeout tasks using the joystick hand controllers. For example, a subject opens a J-hook fastener with the hexdriver end effector and opens a hinged door with the gripper end effector. Each task attempt was considered a run. The subjects were allowed to control the cameras, pan/tilt/zoom. Four time delays from 0 to 8 seconds were used,(0, 1, 4, and 8 seconds).

Results

Time delay significantly increases the time required to perform a task. However, the communication time delay only slightly impacts the time in which the manipulator is actually moving, Fig. 10. This is in close agreement with another manipulator study [Hill, 1975]. Much time is spent controlling cameras and waiting for feedback from a command. The data also indicated that if a task could be performed without time delay that it could be performed with time delay.

Force feedback was not investigated in this study; however, it is felt that force feedback is a required capability that the manipulator system should possess. The changeout of the MMS

module should not be attempted without force feedback due to hidden drive screw activation and heavy reliance on proper end effector orientation. The force and moment information should be presented to the operator in numeric or graphical form rather than forces through the hand controllers.

A problem that had occurred in the learning curve test was not apparent in the time delay tests. Once the subjects were in control of the cameras, they stopped relying on a single view, thus they made better use of the available information.

The subjects used two control strategies, a move-and-wait strategy and a rate strategy. The move-and-wait strategy involves a single manipulator command input followed by a wait until the manipulator finishes its move and the action is perceived by the operator. The move-and-wait strategy is mostly used for fine positioning. The rate strategy involves initiating a command input and holding that input. The operator estimates the movement that will occur after the command is terminated. He terminates the command at that point where he estimates movement will continue to the desired location. The rate strategy is used for gross positioning.

The results showed that some tasks required two manipulators to successfully complete the task. Replaceable subsystems can be designed to minimize two arm manipulator tasks, however, two manipulators would still be beneficial should an anomaly occur.

Overall, it was noted that with proper coordination between subsystem designers and robotic servicer designers, the majority of the remote robotic servicing problems will be alleviated.

SUMMARY

The communication time delay and unstructured worksites offer up potential stumbling blocks for remote robotic servicing of space systems. A study has been conducted to increase our understanding of the remote robotic servicing problems. The study has involved the development of a satellite mockup that consists of replaceable subsystems and is representative of an unstructured worksite. The changeout of these subsystems by a manipulator system, which consists of a manipulator arm, interchangeable end effectors, and a workstation, has been conducted with communication time delays between 0 and 8 seconds. The results include 1) time delay causes a significant increase in the time required to perform a servicing task, 2) time delay has little impact on the actual move time of the manipulator, 3) dual armed manipulator systems are required for changeout of many current subsystem, 4) force feedback is required to perform some servicing tasks, 5) two control strategies were employed by the operators to deal with time delay (move-and-wait and rate), 6) proper coordination between subsystem designers and robotic servicer designers will alleviate the majority of the remote servicing problems, and 7) subjects(operators) consistently perform within their personalities.

REFERENCES

Hill, J. W. (1976). Comparison of Seven Performance
Measures in a Time-Delayed Manipulation Task. IEEE
Transaction on Systems, Man, and Cybernetics, April 4.

TRW (1984). Space Station Automation Study - Satellite
Servicing. NAS8-35081, December.

Wetherington, R. D. and J. R. Walsh (1974). Typical
Teleoperator Time Delay Profiles. NAS8-30919, December.
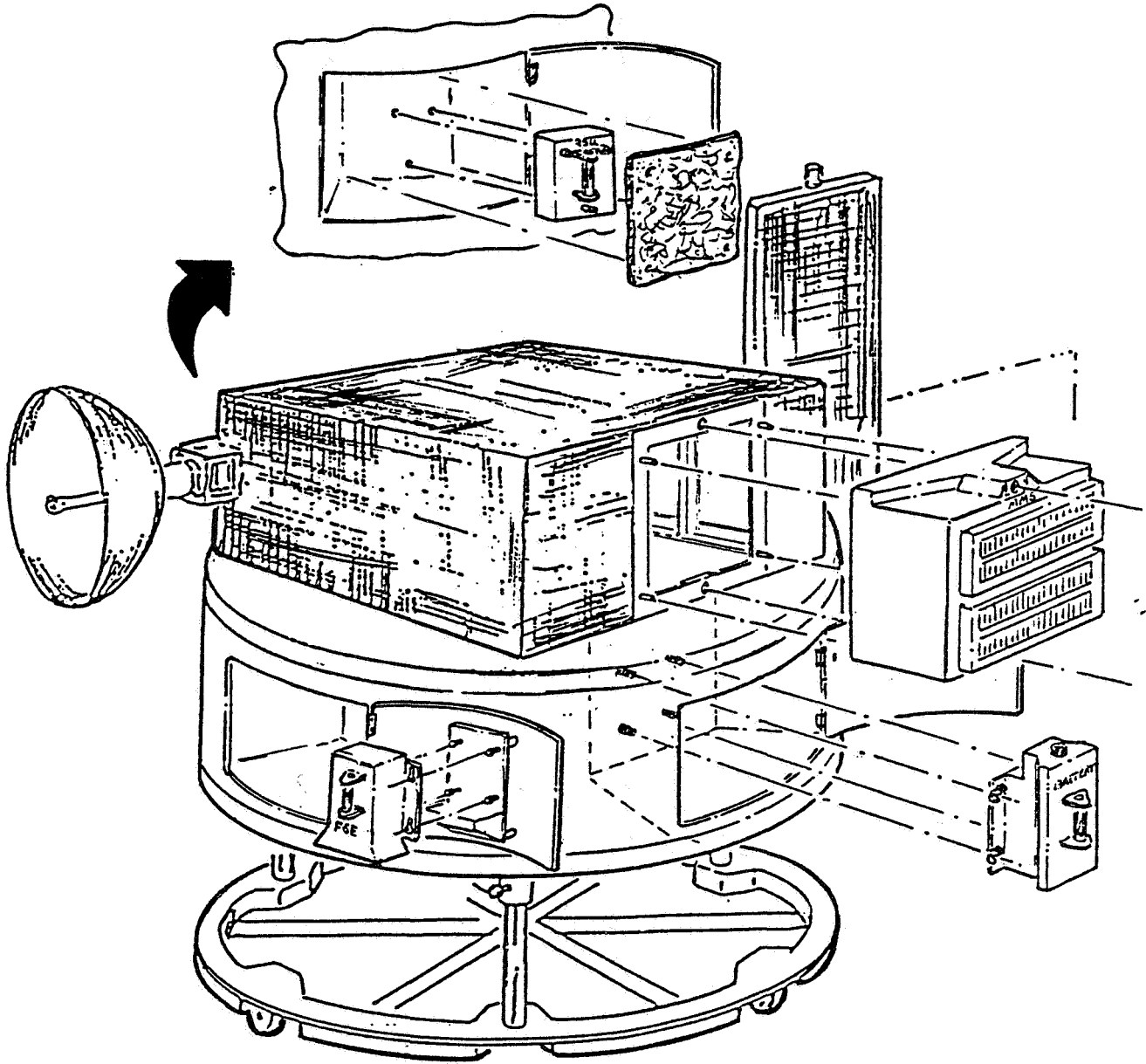
View As Seen From Other Side

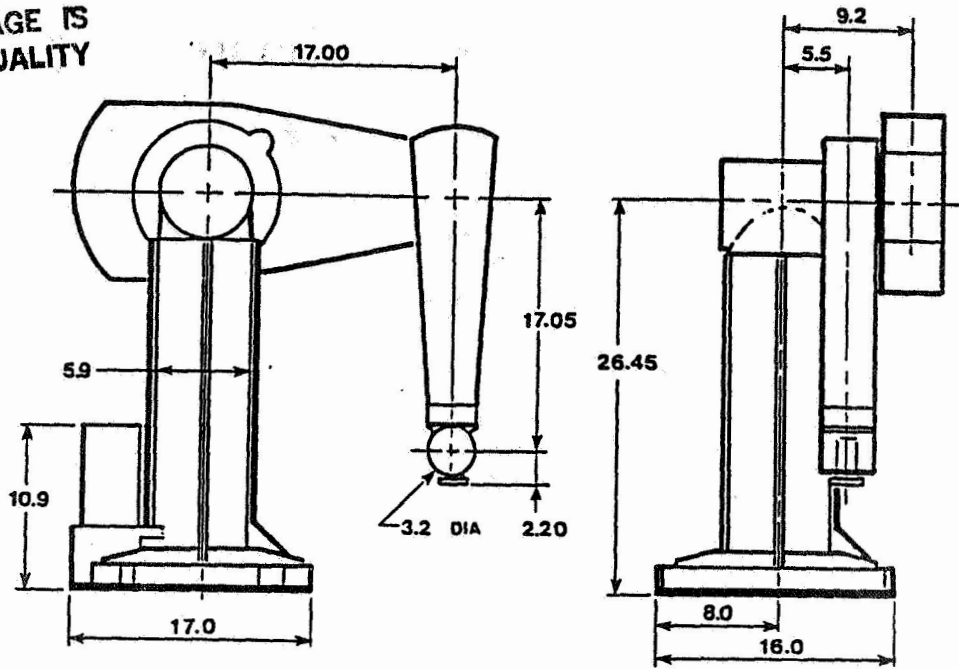
FIGURE 1   Satellite Mockup and Replaceable Subsystems

FIGURE 2   Puma 560 Robot
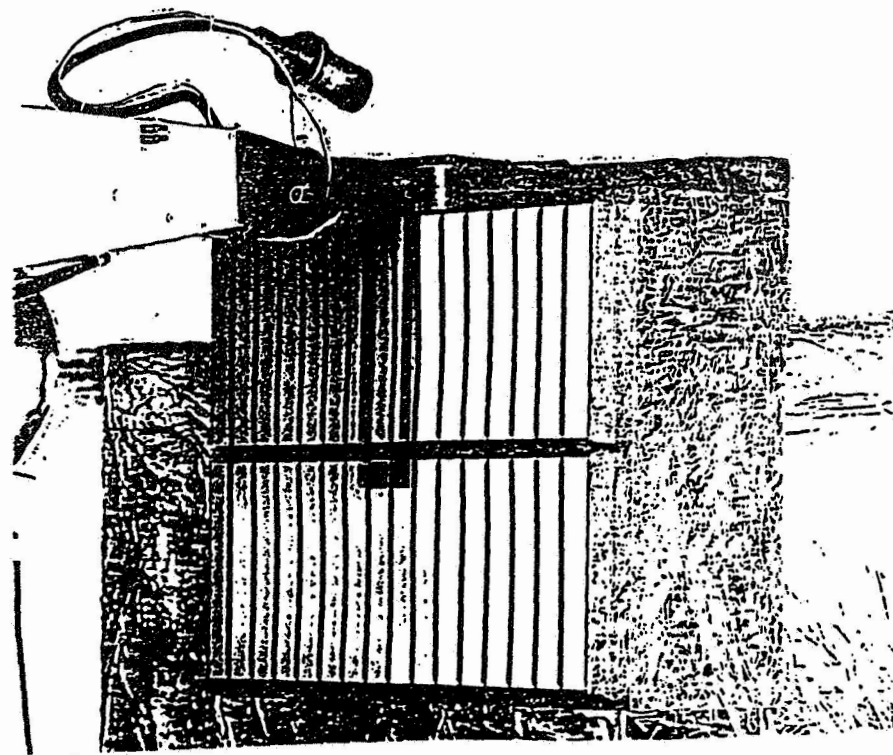
FIGURE 3   MMS End Effector

531
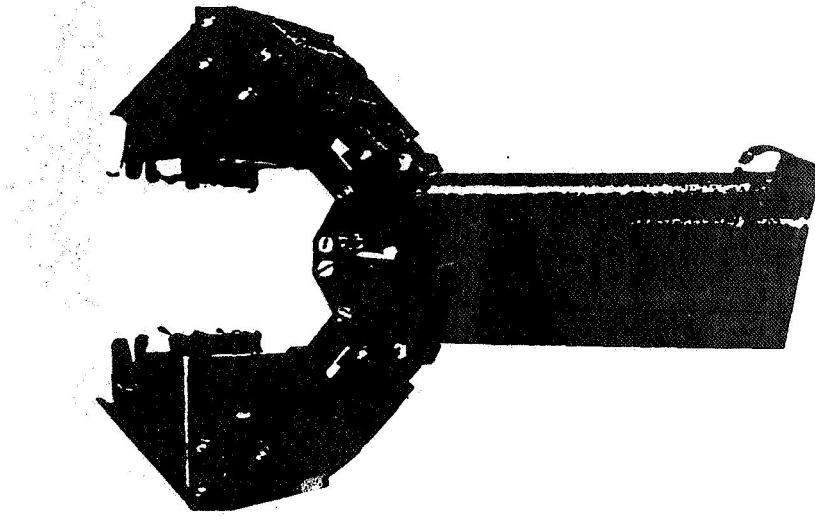
FIGURE 4   Gripper End Effector
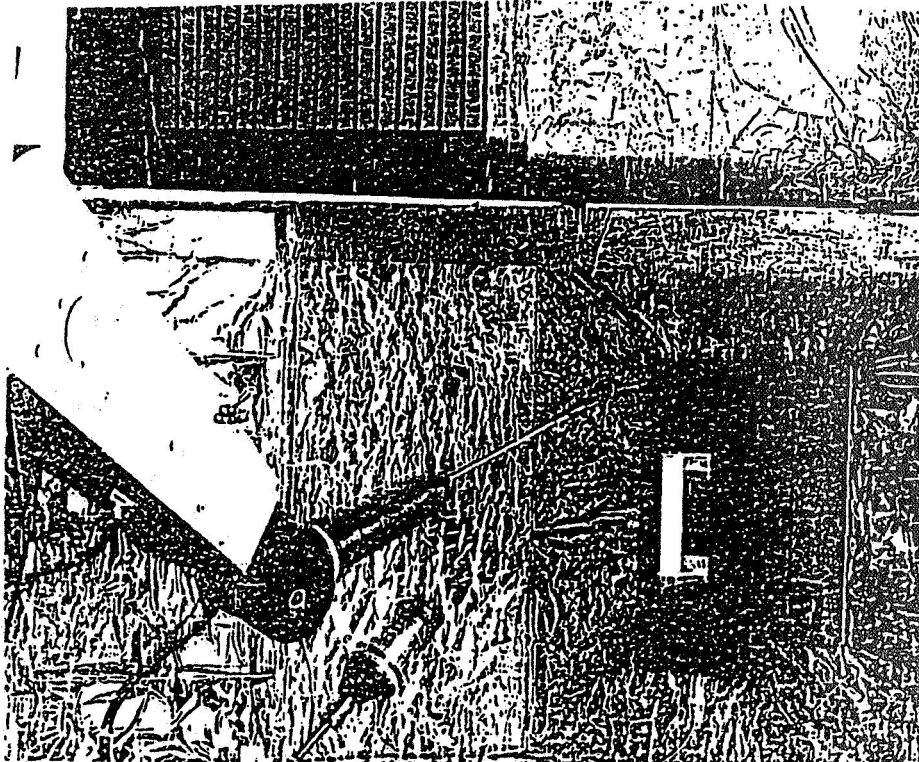
FIGURE 5   Hexdriver End Effector

532
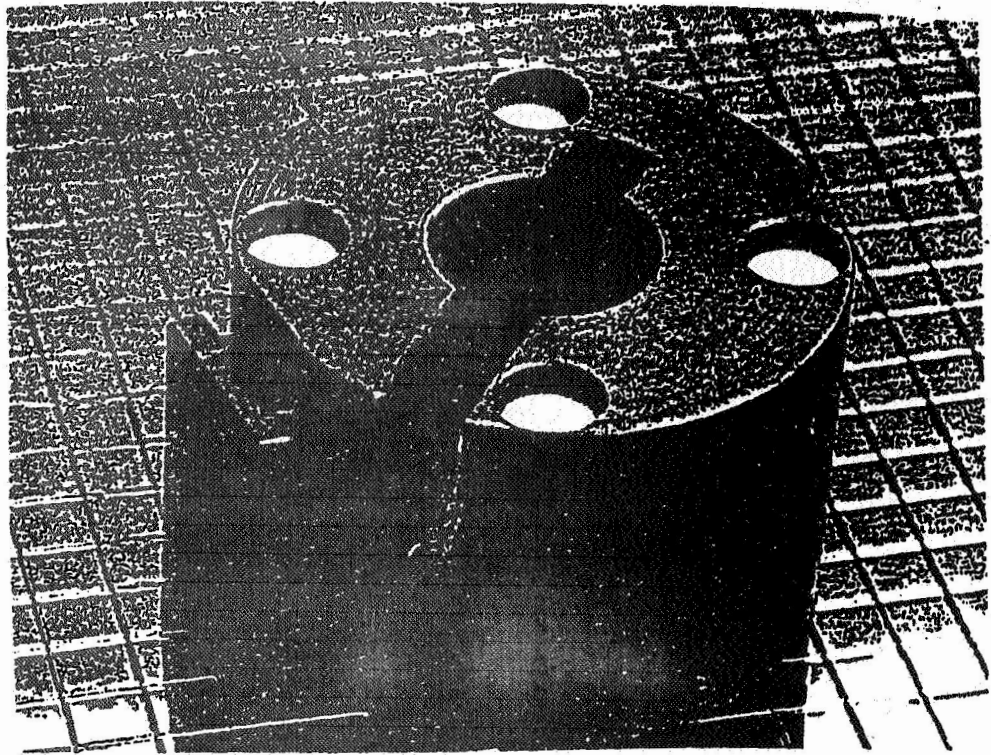
FIGURE 6 End Effector Common Interface

FIGURE 7 Workstation

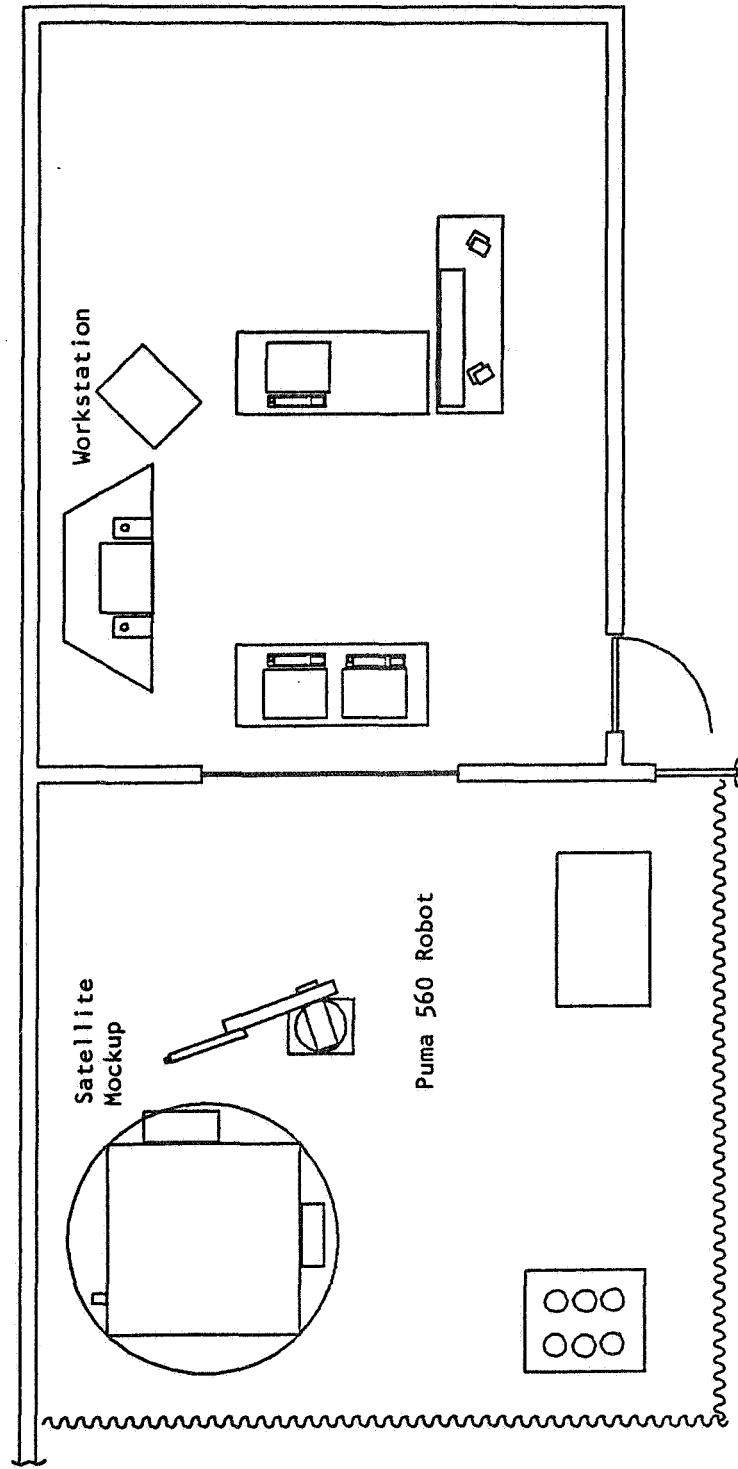REMOTE SATELLITE SERVICING

SPACE
STATION

BOEING

Workstation

Satellite
Mockup

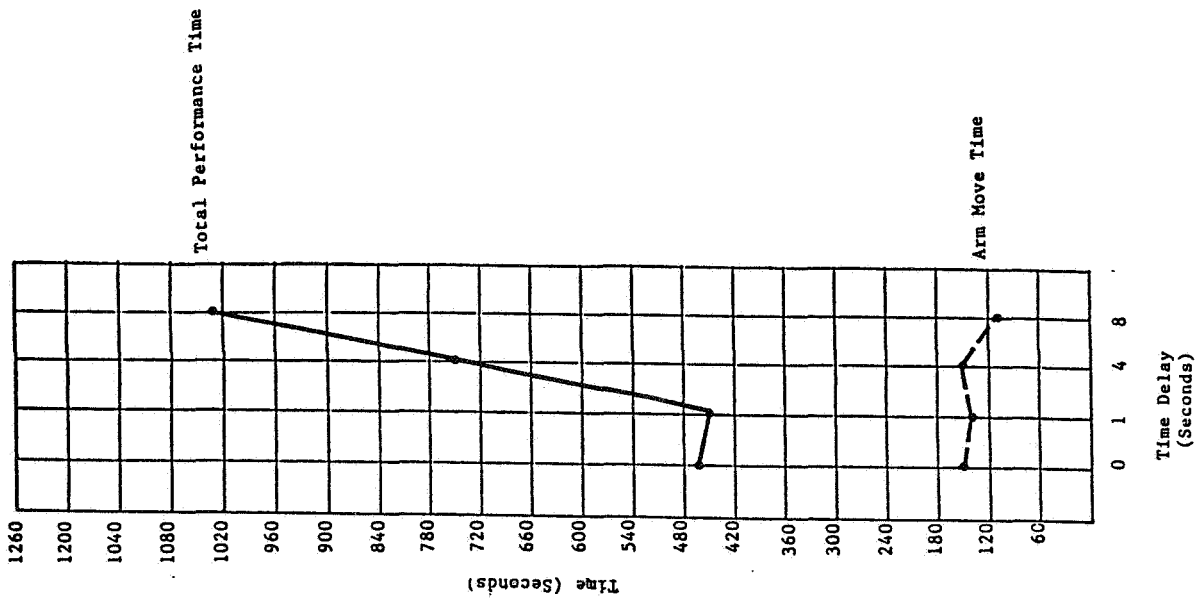Puma 560 Robot

FIGURE 8 Robotics Laboratory Layout

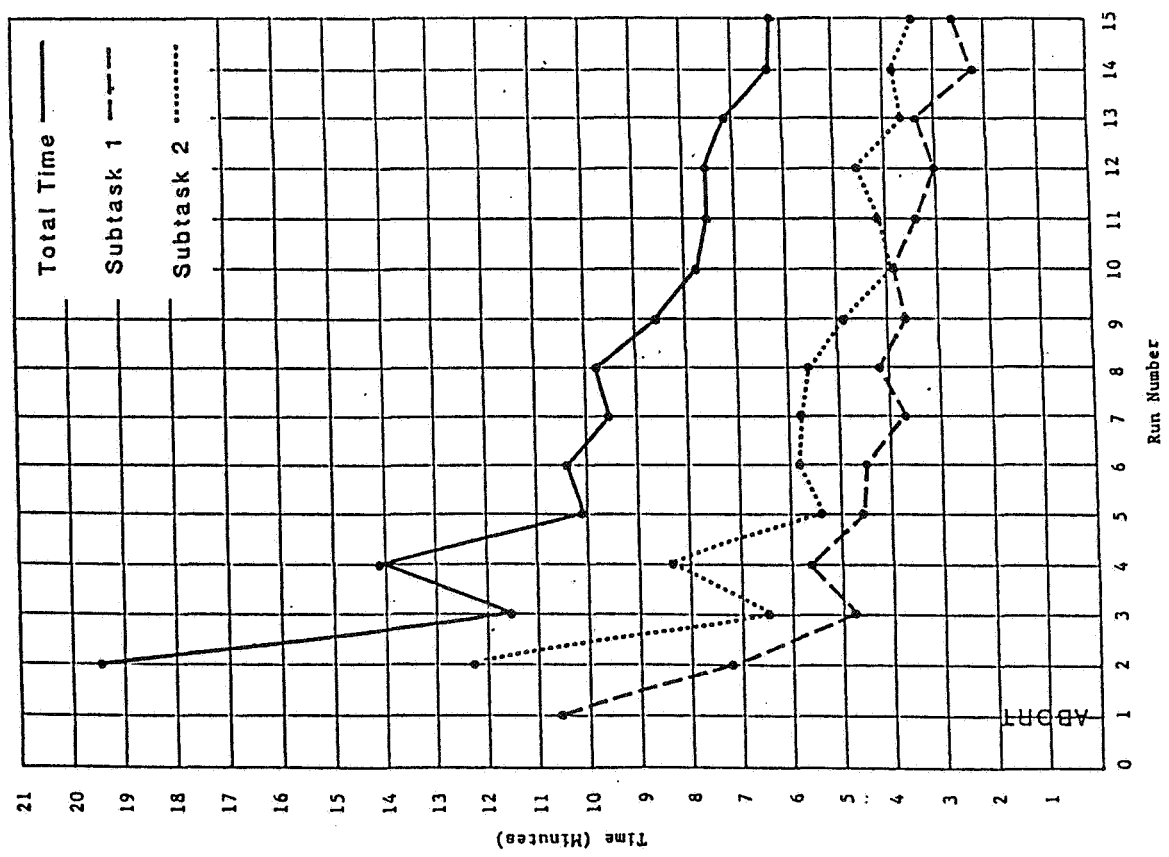FIGURE 10 Averaged Time Delay
Curve (with learning)

FIGURE 9 Typical Subject Learning Curve

535

A Teleoperated Robotic Manipulator System
for Materials Processing Experiment Servicing

Steven Suchting
R. Byron Purves
Boeing Aerospace Company
Space Station Program
Huntsville, Alabama 35806

Jeffrey L. Grover
Roy Scruggs
Georgia Institute of Technology
Georgia Tech Research Institute
Atlanta, Georgia 30332

1.0 Overview

In 1984 Congress authorized NASA to begin the Space Station Program, and requested that ten per cent of program funds be spent in implementing automation and robotics (A&R) on the Space Station. In response to that request, Boeing established several Independent Research and Development (IR&D) projects to explore possible uses for A&R on the Space Station. One of those projects, an automated materials processing experiment, is discussed in this paper. The project uses a teleoperated robot to demonstrate telescience applied to a Chemical Vapor Transport materials processing experiment.

1.1 Congress and ATAC

It was the intent of Congress that use of advanced automation and robotics technologies would not only lead to a more effective and efficient Space Station, but would lead to more productive terrestrial applications by enhancing the scientific and technical base of the United States. To assure Congress that A&R would be considered for use on the Space Station, the Advanced Technology Advisory Committee (ATAC) was established. ATAC prepared and sent to Congress a report called "Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy." The report consisted of requirements and considerations for the use of A&R in future space systems, representative examples of A&R in space related applications, and a discussion of the current technology base with recommendations for future research and development. The report then emphasized the transfer of A&R technologies to terrestrial applications and the importance of A&R to Space Commercialization. In addition to the report, ATAC was chartered to assess the progress of NASA and its contractors in their efforts to incorporate A&R in their Space Station designs.

1.2 Boeing's Role

The Boeing Company, as a contractor to NASA's Marshal Space Flight Center in the Space Station Phase B Study, was required to submit a document describing potential A&R candidates and a plan for their implementation on the Space Station. In addition, we began several Independent Research and Development (IR&D) projects to gain knowledge and experience in implementing automation and robotics technologies on the Space Station. These projects include (1) applications of Expert Systems and Artificial Intelligence to various Space Station subsystems, (2) teleoperated satellite servicing studies, (3) an automated logistics module management system, and (4) a robotic system to service a materials processing experiment.

1.3 A Materials Processing Experiment

This paper discusses the latter project, a system to automate the processing of experimental materials samples. There are several methods that will be used to process materials on the Space Station Laboratory, and Boeing has a special interest in a process called Chemical Vapor Transport (CVT). CVT is a process in which a material contained in an ampoule is heated in a furnace until it is vaporized. The ampoule is then positioned in a slightly cooler portion of the furnace where the material solidifies into crystals. In space, convection currents in the vapor are eliminated which allows the crystal to grow larger and more uniformly. We recently signed a Joint Endeavor Agreement (JEA) with NASA to fly our CVT equipment on the Space Shuttle to process materials that will eventually be used to make electronic devices. In return for the flights, NASA will have the opportunity to use a portion of each flight to process their own materials in our furnaces. In these experiments control of the furnace temperature and the important step of positioning the ampoule in the furnace will be performed by astronauts who will be able to view the crystal as it grows.

1.4 An Automated Materials Processing System

Even though the early experiments will be performed by astronauts, crew time on orbit is an expensive and scarce resource, and demand for crew time on the Space Station is expected to be very high. Therefore, minimizing the requirements for crew servicing of experiments would allow increased sample throughput due to higher utilization of processing equipment. It follows that automation and ground control of the CVT process will reduce the need to compete with other payloads for crew resources, and will increase the number of opportunities to run processing experiments. This shortens the development time (and the costs) necessary to reach the goal of producing systems to manufacture electronic materials in space. A facility to manufacture materials in space will likely contain several furnaces serviced by robots because of their flexibility, i.e. the ablility to perform several types of tasks with a single programmable mechanism. Thus, our reasons to conduct an IR&D project to create an Automated CVT Crystal Growth System using a robot were (1) to reduce the need to use on-orbit crew time in conducting experiments, and (2) to study the design considerations needed to allow servicing by robots of a materials processing system.

2.0 A Ground-based Telerobotic CVT Experiment

In early 1986 Georgia Tech Research Institute (GTRI) performed a requirements definition and preliminary design study for an Automated CVT Crystal Growth System. This study combined GTRI's interest in advanced robotic systems with Boeing's interests in applying A&R technologies to perform potential Space Station functions. GTRI was then awarded a contract to perform sensor to robot integration and to develop software needed for robot operation. This work is discussed in a companion paper "Concepts for Robot Motion Primitives Required for Space Station Teleoperations". The project began in March and will end with a final report in November.

2.1 Project Objectives

This project was begun with several primary and secondary objectives. One of the most important objectives was to gain knowledge of the problems and potential of a telescience/telerobotic capability. Telescience is the ability to monitor and control an experiment from a remote location. Similarly, telerobotics is the ability to monitor and control a robot from a remote location. A related objective was to demonstrate to NASA and ATAC that Boeing has an understanding of the utility and capabilities of telescience and telerobotics as they might be used in the Space Station Laboratory. Other related objectives include: (a) determine experiment and laboratory design factors necessary to enable useful operations with a robot; (b) determine monitoring, control and communications requirements for experiment operations from a telescience workstation; (c) perform actual "hands-off" crystal growth experiments from a remote location. Another of the primary objectives of the project was to develop a base of expertise in the application of robotics to space applications. Secondary objectives related to this one include: (a) obtain a set of hardware and software tools with which we may perform further research; (b) learn the capabilities and limitations of a typical six-degree-of-freedom arm, e.g. reach, repeatability, kinematics, etc.; (c) work with a "high-level" robotic programming language; (d) discover methods to interface sensors to develop intelligent robot motions.

2.2 CVT Process Tasks

In order to automate the CVT crystal growth process, the tasks involved in running an experiment on the Space Station were analyzed. A scenario was developed which assumed that the processing equipment had already been installed in a Space Station rack, that the ampoules containing the material to be processed had been prepared on the ground, and that the ampoules had been placed in a temporary storage location in the equipment rack. Consideration was given to demonstrating a transfer from a logistics module storage location to the equipment rack, but payload limitations of the robot prevented this.

To begin the process, a thermal profile appropriate for the material being processed is created with the furnace controller. A request for appropriate resources is made to the Space Station resources scheduler, and an approval with appropriate start time is returned from the scheduler. At the appropriate start time an ampoule is removed from its storage location and placed into the furnace. The furnace controller raises the furnace temperature to vaporize the material in the ampoule. The ampoule is then positioned in the furnace's temperature gradient so that crystal growth begins. The ampoule is repositioned as necessary throughout the process to maintain an optimum growth rate. At the end of crystal growth the ampoule is removed from the furnace and placed in a cooling location so that the furnace may be used to process the next sample. When the processed ampoule has cooled it is replaced in the storage box for later shipment back to Earth.

2.3 Automated System Components

The Automated CVT Crystal Growth System consists of two major subsystems, and each subsystem consists of hardware and software components. The two major subsystems are the telerobotic servicing system, and the CVT processing rack.

2.3.1 Telerobotic Servicing System

The robotic system is built around a Unimation Puma 650 six-degrees-of-freedom robot that is operated with Unimation's VAL II robotic programming language. Two types of sensors are integrated with the robot. A Lord Corporation F/T 15/50 force-torque sensor is mounted to the robot's wrist, and a Lord Corporation LTS-200 combination array and vector sensor is mounted to the end effector to provide a tactile sense. The end effector is a specially designed pair of fingers that are pneumatically actuated through a programmable valve which provides sixteen levels of gripping pressure. The sensors and valve are interfaced to Unimation's controller through parallel and serial ports. Also interfaced to the robot controller is a Hewlett-Packard 9836 Engineering Workstation which provides supervisory high-level software functions. The HP9836 is used because it is a standard component in Boeing's Laboratory Data Management System.

Software for the robotic system runs on both the Unimation robot controller and the HP9836. The robot controller is responsible for communications with the sensors and for low-level sensor-coordinated robot motions. The HP9836 provides a user interface, handles data management functions, and dispatches commands to the robot controller. Data management maintains information on objects, the positions in which those objects may be placed, and the current state of the system being serviced, i.e. what objects are where. The user interface allows access to data information and to high-level motion commands. A significant feature of the system is the ability to create script files of all motion and data commands, and to name and execute the script as a single command. The software for this system is discussed in more detail in the previously mentioned companion paper.

2.3.2 CVT Processing Rack

The CVT processing rack consists of the CVT furnace and its support equipment. The furnace is a breadboard model of the furnace that we will fly on the Space Shuttle, and is transparent to allow direct observation of the crystal growth process. An ampoule positioning mechanism was designed for this project. The positioner accepts an ampoule from the robot, inserts the ampoule into the furnace, and allows micrometer positioning of the ampoule within the furnace. A cooling location is mounted below the positioner and uses the same ampoule holding mechanism as the positioner. A storage box is located below the cooling location and has room for four ampoules. Direct observation of crystal growth is made possible by a mid-range microscope and video camera combination. Furnace temperature control is accomplished using a Hewlett-Packard 3054 Data Acquisition System, an HP 6010 Programmable Power Supply, and an HP9836 Engineering Workstation. This equipment communicates with each other through an IEEE-488 standard Interface Bus. The HP9836 also communicates with the ampoule positioner through a standard RS-232 connection.

Software for the processing rack resides on the HP9836 and consists of a user interface, a PID furnace controller, and a positioner interface. The positioner interface consists of translating user commands and positioner status into ASCII command strings for a programmable motor controller. The furnace controller requests thermocouple data from the acquisition system, applies a PID control algorithm, and updates the power supply states. The user interface allows creation and modification of a thermal profile for the furnace to follow over time. A graphic display is provided of the desired temperature profile, the actual temperature readings from several locations within the furnace, and the power consumption.

2.4 Operational Scenario

Remote operation of a CVT crystal growth experiment is simulated by placing the CVT equipment rack within reach of the Puma robot, and by placing the HP9836 user interface computers in an adjacent control room. The control room is visually and acoustically separated from the robot laboratory, although between them there is a large window with blinds to allow direct observation if necessary. Visual information of the robot and equipment rack is provided by color video cameras in the laboratory. A workstation in the control room contains three video monitors; two are normally used to view the work area and one is used to monitor crystal growth.

Initialization of the system requires several steps. The robot subsystem is initiallized by loading from a data base the current state of the CVT system, i.e. what objects are in what positions in the rack. The robot then attempts to find a reference location on the rack. The reference location is currently a post perpendicular to and extending from the plane of the front surface of the rack. The robot grasps the post and uses the force-torque sensor to fine-tune the location of the post relative to the robot. All other locations in the rack are referenced relative to that post. The CVT system is initialized by moving the positioner to a home location that allows the ampoule to be loaded without danger of it coming into contact with the furnace. A profile of ampoule position versus time is then entered into the CVT control computer. Also entered is a temperature profile for the furnace.

The process of crystal growth begins by commanding the robot to grip a requested ampoule. The robot is then told to insert the ampoule into the furnace positioner and then to release the ampoule. Optionally, these steps could be specified in a script file, and that script file could then be executed. The robot is now free to service other experiments until processing of the crystal is complete. In our current setup, the robot executes a script which conducts several activities on a mockup task board.

The CVT computer commands the ampoule positioner to insert the ampoule into the furnace. The furnace temperature profile is then automatically begun. At the appropriate point in the temperature profile, the ampoule position profile is begun. The experiment operator views crystal growth with the microscope-video system and changes temperature and position parameters as necessary. When crystal growth is complete the ampoule positioner withdraws the ampoule from the furnace to its home position. A command to grip the ampoule is issued, and the robot assumes the ampoule is in the position where the it was last placed. The ampoule is moved to the cooling location, and the next ampoule is loaded into the positioner. After several minutes the ampoule is removed from its cooling location and replaced in the storage box.

541

3.0 Conclusions

This project has demonstrated the feasibility of structuring materials processing experiments on the Space Station laboratory so that they can be executed with the aid of a telerobotic system at little if any additional cost to the experiment. Although not all experiments may be so structured, a significant number of them could be. These experiments would benefit by not needing to compete for a scarce crew time resource, and other experiments would benefit from increased crew availability. A telerobotic and telescience capability would result in increased "science throughput" of the laboratory.

All of the objectives mentioned earlier were achieved with this project. The potential of telescience/telerobotics appears to increase the productivity of the Space Station laboratory and is a necessary step to the manufacturing of materials in space. We have performed valuable "hands-off" ground-based crystal growth experiments using automated equipment that has increased the state of the art of CVT materials processing. In addition, we have obtained a valuable set of tools with which we may further explore sensor-based robotics research and the application of robotics to space-based systems. Experiment and laboratory design factors necessary to allow telescience with a robot will be discussed in more detail in a future paper.

APPENDIX

Contained in this section are papers and abstracts which were
not included in the body of the proceedings.

DMES - Distributed Module Expert System
        S. Purinton, C. Wang, NASA-MSFC

Utilization of Artificial Intelligence Techniques for the
Space Station Power System
        T. C. Evatt, E. W. Gholdston,
        Rockwell International/Rocketdyne Division

A Methodology for Multiple Rule System Integration and
Resolution within a Singular Knowledge Base
        F. N. Kautzmann, MITRE Corporation

Blackboard Architectures and Their Relationship to
Autonomous Space Systems
        A. Thornbrugh, Martin Marietta Denver Aerospace

Design Consideration in Constructing High Performance
Embedded Knowledge-Based Systems (KBS)
        S. Dalton, P. Daley, Martin Marietta Denver Aerospace

Validation of Expert Systems
        R. Stachowitz, J. Combs, Lockheed Missles & Space Co.

A Nonlinear Filtering Process Diagnostic System for the
Space Station
        R. Yoel, Boeing Aerospace Company
        M. Buchner, K. Loparo, A. Cubukco,
        Case Western Reserve University

Mathematical Algorithms for Approximate Reasoning
        J. Murphy, S. Chay, M. Downs, Westinghouse R&D Center

Real-Time Space System Control with Expert Systems
        D. Leinweber, L. Hawkinson, LISP Machine Inc.
        J. Perry, OAO Corporation

A Flexible Search Strategy for Production Systems
        P. Dey, S. Srinivasan, K. R. Sundaraghavan
        University of Alabama at Birmingham

Semantic Based Man-Machine Interface for Real-Time
Communication
        M. Ali, C. S. Ali,
        University of Tennessee Space Institute

C-7

The Concurrent Common Lisp Development Environment
        S. Curtis, Gold Hill Computers

Automated Software Development Workstation
        D. A. Prouty, P. Klahr, Inference Corporation

AI Tools in Computer Based Problem Solving
        A. J. Beane, Digital Equipment Corporation

Networking & AI Systems: Requirements & Benefits
        S. Curtis, Gold Hill Computers

An Expert System for Natural Language Processing
        J. F. Hennessy, Digital Equipment Corporation

Expert System Technology as a Data Processing Tool
        J. F. Hennessy, Digital Equipment Corporation

Next Generation Space Manipulator
        P. Brunson, W. Chun, P. Cogeos, Martin Marietta Denver
        Martin Marietta Denver Aerospace

Automated Practical Reasoning Systems
        M. Lewis, State University of New York, Binghamton

AN EXPERT SYSTEM FOR A DISTRIBUTED REAL-TIME TRAINER

DISTRIBUTED MODULE EXPERT SYSTEM

Steven C Purinton
Caroline K Wang

NASA
Marshall Space Flight Center
Huntsville, Alabama 35812

Abstract

The problem addressed by this expert system concerns the expansion of capability of a Real Time Trainer for the Spacelab flight crew. As requirements for more models or fidelity are placed upon the system, expansion is necessary. The simulator can be expanded using a larger processor or by going to a distributed system and expanded by adding additional processors. The distributed system is preferable because it is more economical and can be expanded in a more incremental manner.

An expert system was developed to evaluate modeling and timing capability within a real-time training simulator. The expert system is based upon a distributed configuration. Components of the modelled system are control tasks, display tasks, keyboard tasks, data buffer tasks, network tasks, emulator tasks, processors, displays, and a network. DMES allows the configuring of processors, tasks, display use, keyboard use, and selection of alternate methods to update the data buffer. Modules can be defined with execution occuring in a specific processor on a network. The maintenance of the realtime data buffer can be accomplished with one of several methods. The effects of execution delays, data buffer maintenance, display location, and network traffic are displayed on the screen and stored in a history file for a more thorough examination offline.

The system consists of a knowledge front-end editor to interactively generate or update the knowledge base, an inference engine, a display module, and a recording module.

DMES

Distributed Module Expert System

By

Steven C. Purinton
Caroline K. Wang

MARSHALL SPACE FLIGHT CENTER

Description

DMES is an expert system concerned with the execution of a realtime training simulation on a distributed system.

History

A facility was developed at Marshall Space Flight Center to provide training for the Spacelab payload crew and mission specialists. The training facility originally consisted of a two processor Host with shared memory and a Spacelab mockup. Communications links between the Host computers and the mockup provided for serial ASCII, display, discrete, and analog data. Modules functionally representing the experiment computer I/O unit, operating system, and application software as well as payload hardware, and environment were developed. A control task was developed to provide for the selection of a training configuration, module scheduling, mode control, and monitoring of the simulator. Modules are components of the simulator and may be a single task or multiple tasks with a single function.

As the Payload Crew Training Complex evolved, a telemetry communication link to the Payload Operation Control Center, a shuttle aft flight deck workstation, scene generation, and two processors were added to the host. Training sessions were conducted using one processor dedicated to communications and two processors hosted the experiment modules, system modules, and control tasks. Figure 1 depicts this system.

The sequence of execution was complex because some of the processing was serial and other parallel. Some modules, such as environment, time, and, input/output are required to execute only once between instances of an experiment or application software module. Multiple instances would (or could) mean the omission of an event in the simulator. This condition occurs when execution is not predictable as with multiple processors or a timesliced system. Simularily, multiple instances of an experiment or application would cause effects as wasting time, a mixture of input-output data, or other undesirable problem.

With three active processors as Host, a training session configuration would evolve using experience, intuition, and testing. The order of execution and location for execution would be the parameters varied. The verification process could consume several days of testing.

The simulation has been transfered to a single VAX and expansion to a distributed system is in progress. The training simulator is composed of serial and parallel modules and must also deal with network delays and the problem of updating a centralized data base. This growth presents the facility with a much more complex problem when it comes to determining a configuration which will execute in real time. In response to this problem an expert system is being developed to allow the analysis of a training session.

Implementation

The expert system loads the knowledge base, which consists of the following items:

1. Number of processors
2. Default tasks
3. Number of experiment modules
4. Processor location for each experiment module
5. Data dependance for each module
6. Execution time for each module
7. Network delays
8. Database read/write
9. Basic cycle time

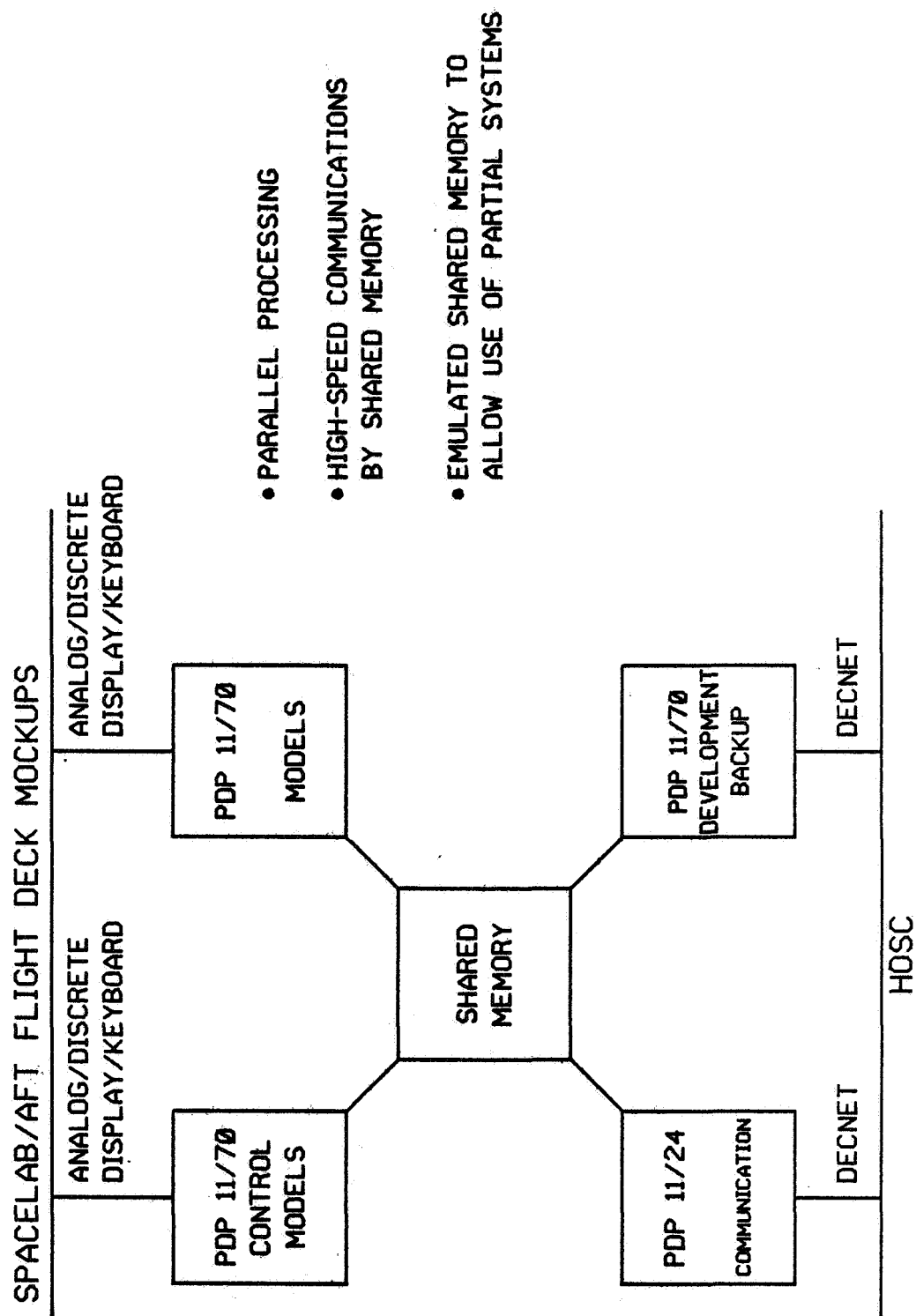A knowledge editor allows input and adjustment of these items.

A control task is started in the first processor, this in turn schedules the execution of other tasks and modules within the first processor and schedules the execution of the control task in additional processors. A cycle is complete when all modules in all processors have finished execution. Execution is defined as being active for the amount of time specified in the knowledge base. Any database accesses from the other processors will extend the active time by the amount of access time. A failures would be a database access out of sequence or too much time used during a cycle.

Enhancements

The operating system is not part of the expert system, but rather the control task for the simulator is modeled. As the expert system is refined it may be necessary to include the operating system and it effect as elements. Different methods to update or distribute the database need investigated. Also distribution changes in functions such as display updates or I/O can be included. Accuracy and resolution can be improved as needed to make the expert system useful. Randomness can be used in some areas to improve problem detection.

PAYLOAD CREW TRAINER

SPACELAB/AFT FLIGHT DECK MOCKUPS

ANALOG/DISCRETE DISPLAY/KEYBOARD

ANALOG/DISCRETE DISPLAY/KEYBOARD

PDP 11/70 CONTROL MODELS

PDP 11/70 MODELS

SHARED MEMORY

PDP 11/24 COMMUNICATION

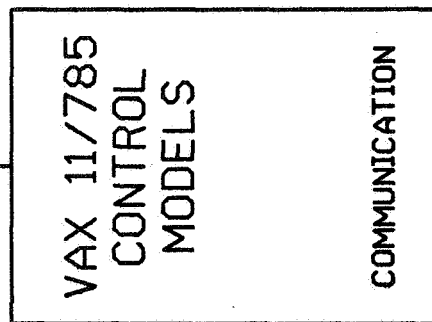PDP 11/70 DEVELOPMENT BACKUP

DECNET

DECNET

HOSC

• PARALLEL PROCESSING

• HIGH-SPEED COMMUNICATIONS BY SHARED MEMORY

• EMULATED SHARED MEMORY TO ALLOW USE OF PARTIAL SYSTEMS

549

REPLACEMENT PAYLOAD CREW TRAINER

SPACELAB/AFT FLIGHT DECK MOCKUPS

IEEE FOR ANALOG/DISCRETE
DISPLAY/KEYBOARD

VAX 11/785
CONTROL
MODELS

COMMUNICATION

DECNET

HOSC

- ALL COMPONENTS IN A SINGLE MACHINE
- MULTIPLE SIMULATORS POSSIBLE IN A SINGLE MACHINE

550

DISTRIBUTED EXPANSION OF VAX-BASED PCTC

SPACE LAB/AFT FLIGHT DECK MOCKUPS

- DISTRIBUTED SYSTEM
- REMOTE CONTROL OF EXECUTION
- REMOTE UPDATING OF DATA

IEEE FOR ANALOG/DISCRETE DISPLAY/KEYBOARD

VAX 11/785 CONTROL MODELS

COMM

MICROVAX REMOTE CONTROL MODEL

MICROVAX REMOTE CONTROL MODEL
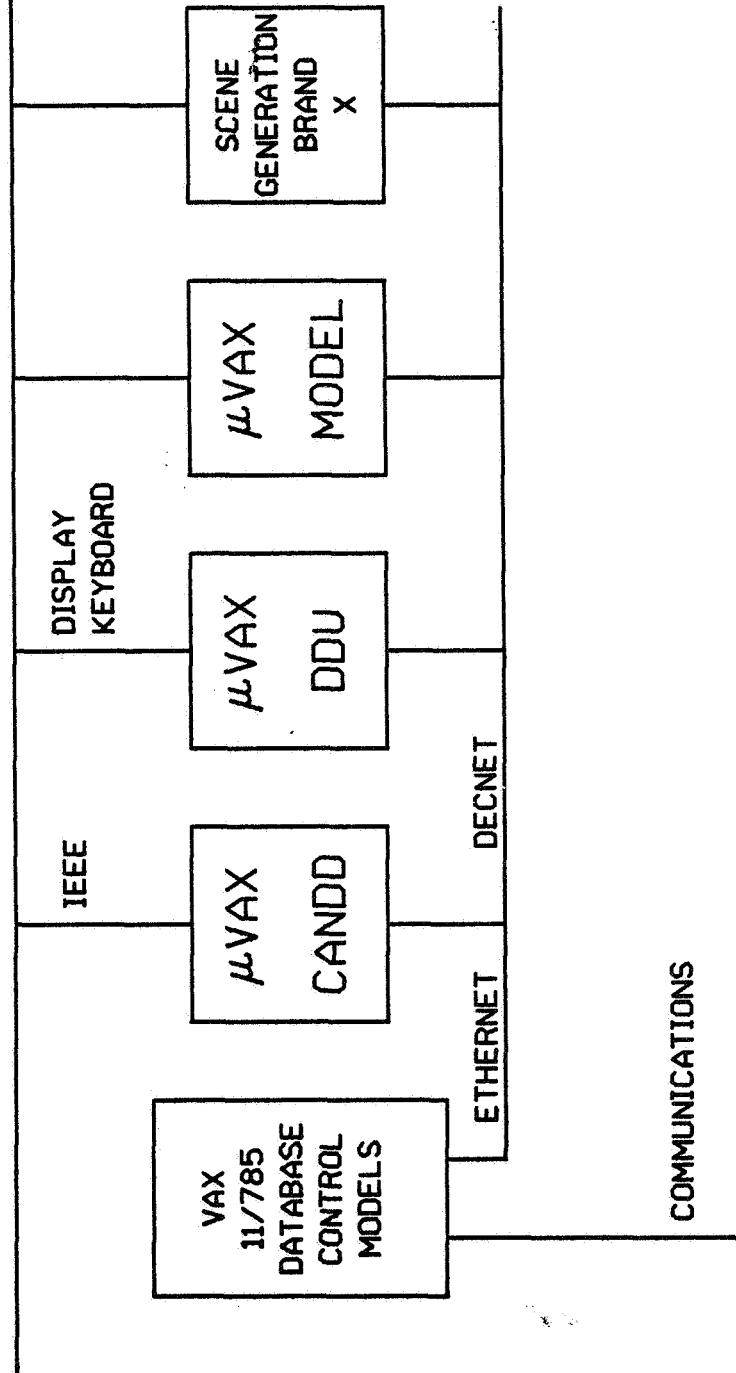
ETHERNET/DECNET

DECNET

HOSC

DECENTRALIZED RESOURCES

SPACELAB/AFT FLIGHT DECK MOCKUP

Features

Distributed Module Expert System (DMES) is a prototype frame based Expert System designed to evaluate timing and loading conditions for a digital simulator (for Payload Crew Training).

DMES was developed on Symbolics 3670.
DMES has the following features:

1.   An user friendly interface utilizing the Symbolics window system capabilities to:

         a. Create information data base
         b. Display information data base
         c. Update information data base

2.   Automatic knowledge base generation.

         Automatically generate the knowledge base from the information data base.
         The knowledge base includes:
         The procedure for each individual processor.
         Information for graphics display.

3.   Real time graphics display

         DMES is utilized to operate one cycle per second.
         DMES·Checks on the multiple processors of the simulator and searches for the current experiment for each processor and puts each experiments into the queue stack and fires the experiments on schedule.
         DMES also graphically-displays the current status of the experiments and the same time stores the detail information into the buffer for future analysis.

4.   Generate DMES history file
         There is an option for writing the history file from the history buffer.

5.   The history file can be graphically displayed for closer analysis.

Benefits

DMES was designed by using common lisp for the expert system portion and zetalisp for a user interface window flavor techniques and graphics.
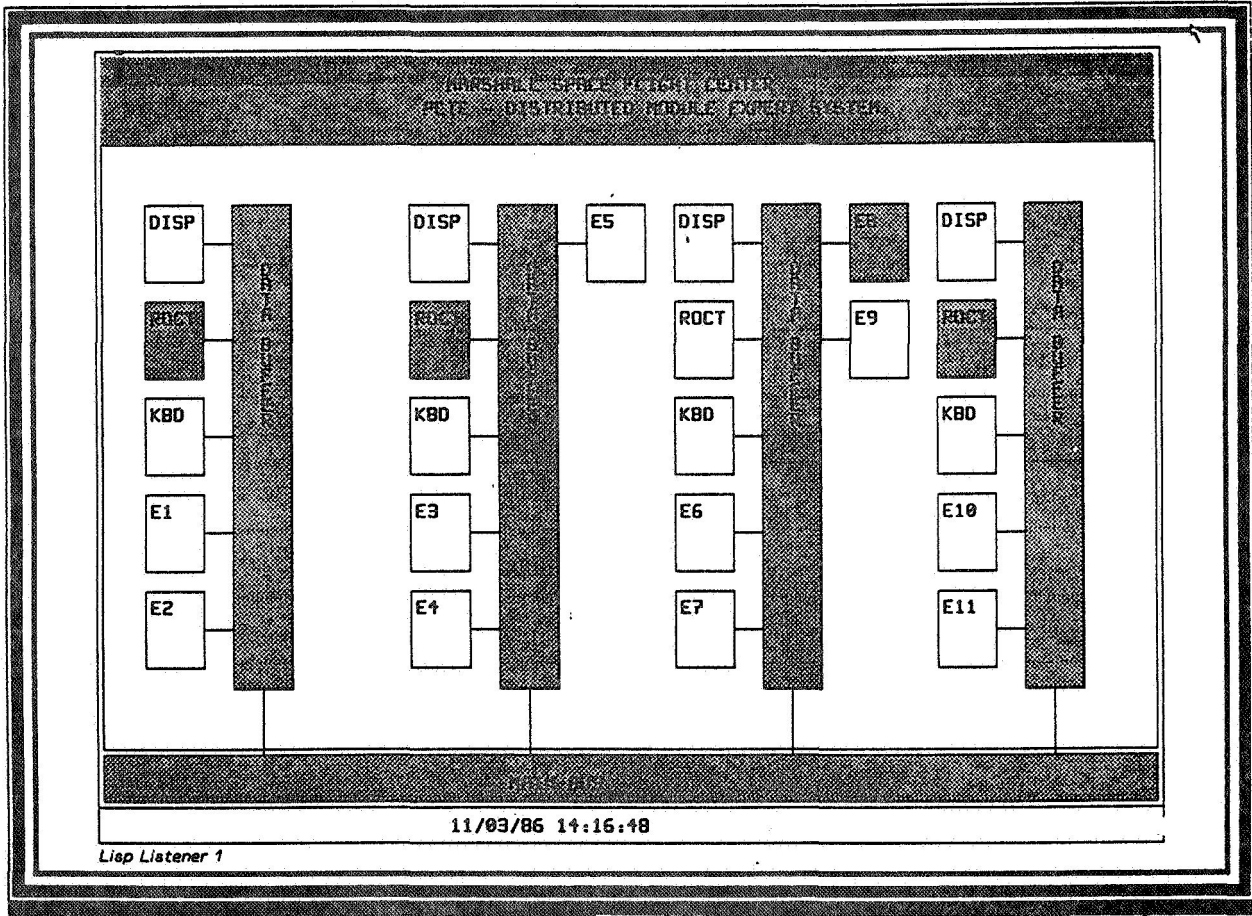It can be transfered to other systems by rewriting the user interface and graphic display.

DMES can be helpful in studying and modeling the complex distributed system for multiple processors. It helps to quickly see the problem and also can modify the model easily.

DMES TOP LEVEL MENU
CREATE DATA BASE
DISPLAY DATA BASE
UPDATE DATA BASE
EXIT

file name: c:>caroline>dmes>demo.dat

Distributed Module Expert System

file name:		c:>caroline>dmes>demo.dat
number of	processors:	4
Processor	name	p1
number of	experiments:	2
ROCT	wait time	4
	set inactive	T
	wait time	10
	set inactive	T
DISPLAY	wait time	2
	read data	T
	wait time	6
	set inactive	T
KBD	wait time	1
	write data	T
	wait time	2
	set inactive	T
Experiment	name	e1
EXPERIMENT	wait time	2
	read data	T
	wait time	4
	write data	T
	set inactive	T
Experiment	name	e2
EXPERIMENT	wait time	4
	read data	T
	wait time	6
	write data	T
	set inactive	T

Press any key for next page

COMMAND LINE:

11/03/86 14:16:48

Lisp Listener 1

UTILIZATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES
FOR THE SPACE STATION POWER SYSTEM

Thomas C. Evatt and Edward W. Gholdston

Rockwell International/Rocketdyne Division
Canoga Park, California

Abstract

Due to the complexity of the Space Station Electrical Power System (EPS) as currently envisioned, artificial intelligence/expert system techniques are being investigated to automate operation, maintenance, and diagnostic functions. A company-funded study has been conducted to investigate this technology as it applies to failure detection, isolation, and reconfiguration (FDIR) and health monitoring of power system components and of the total system. Control system utilization of expert systems for load scheduling and shedding operations has also been researched. A discussion of the utilization of artificial intelligence/expert systems for Initial Operating Capability (IOC) for the Space Station effort is presented along with future plans at Rocketdyne for the utilization of this technology for enhanced Space Station power capability.

Introduction

Automation will need to be an integral part of the Space Station Electrical Power System (EPS). At Initial Operating Capability (IOC) and beyond, the implementation of artificial intelligence technologies is envisioned at increasingly sophisticated levels to minimize the need for crew interaction and to maximize station productivity. The goal at IOC is for the power system to automatically operate, reconfigure itself in case of failure, monitor system health, and provide a diagnostic expert system to assist with maintenance, fault isolation, and component replacement (Ref. 1-9). Current research at Rocketdyne is focused on the implementation of these functions, with projections beyond IOC for the implementation of advanced artificial intelligence (AI) hardware and software as they are developed.

IOC Power System Automation

There are four basic motivating factors for inclusion of advanced automation in the station power system:

- Crew safety (hazard exposure, extravehicular activity (EVA) time, etc.)

- Crew and ground operational efficiency

- System/subsystem reliability (maintenance and replacement costs)

- Mission success (power availability for scientific experiments, etc.)

With these requirements in mind, an initial power system architecture has been formulated with capabilities that can be expanded in a series of incremental steps, each of which would improve the reliability and efficiency of the overall power system. The current EPS design consists of solar dynamic (SD) and photovoltaic (PV) power generation units, energy storage equipment, and power management and distribution systems. Figure 1 shows a schematic of the complete power system illustrating the locations of power sources, bus switching assemblies, and load points. The widely distributed nature of the sources and loads has led to a computer control architecture that reflects a similar distribution around the station. The processors are configured in a hierarchical arrangement that allows many control functions to be retained at lower, local levels, as shown in Fig. 2. At these lower levels, the power system will generally have "classical" control (algorithm control to a given setpoint), with manual flight crew override, and ground override as a last resort. Global coordination and other higher level functions, such as load scheduling, are

PV/ESS ELECTRONICS ENCLOSURE

SD ELECTRONICS ENCLOSURE

● POWER DISTRIBUTION AND CONTROL ASSEMBLY
  SOURCE PMAD
■ POWER MANAGEMENT PROCESSOR
▲ MAIN BUS DISTRIBUTION ASSEMBLY
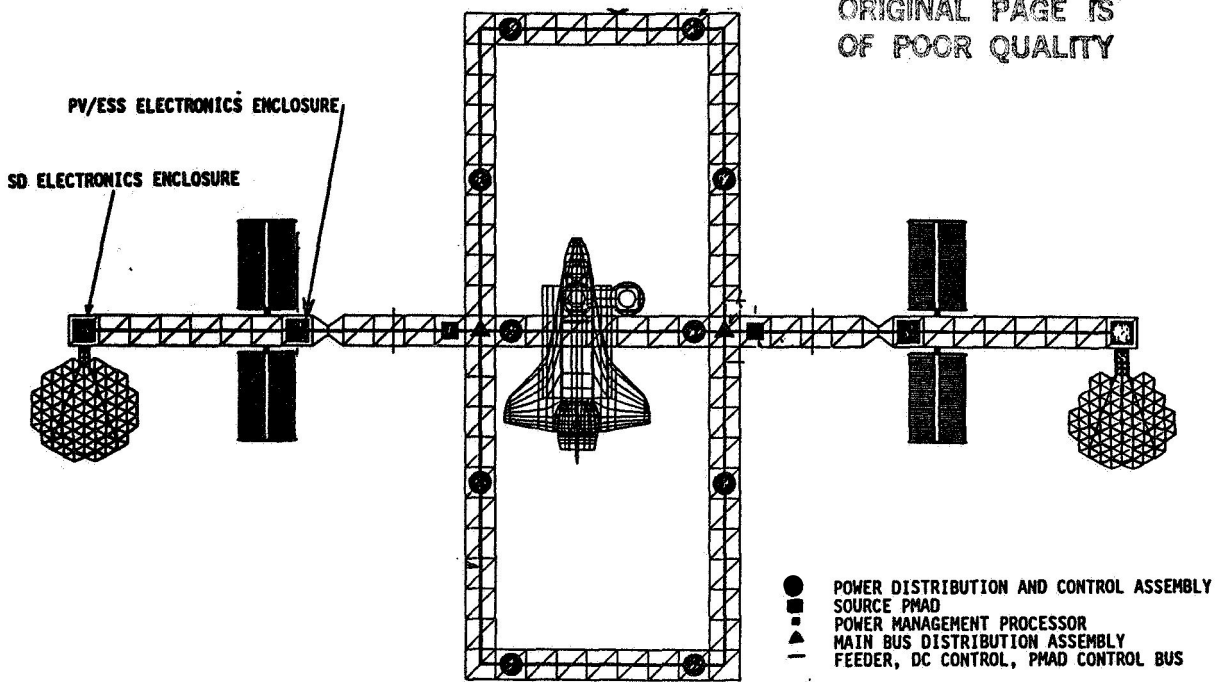— FEEDER, DC CONTROL, PMAD CONTROL BUS

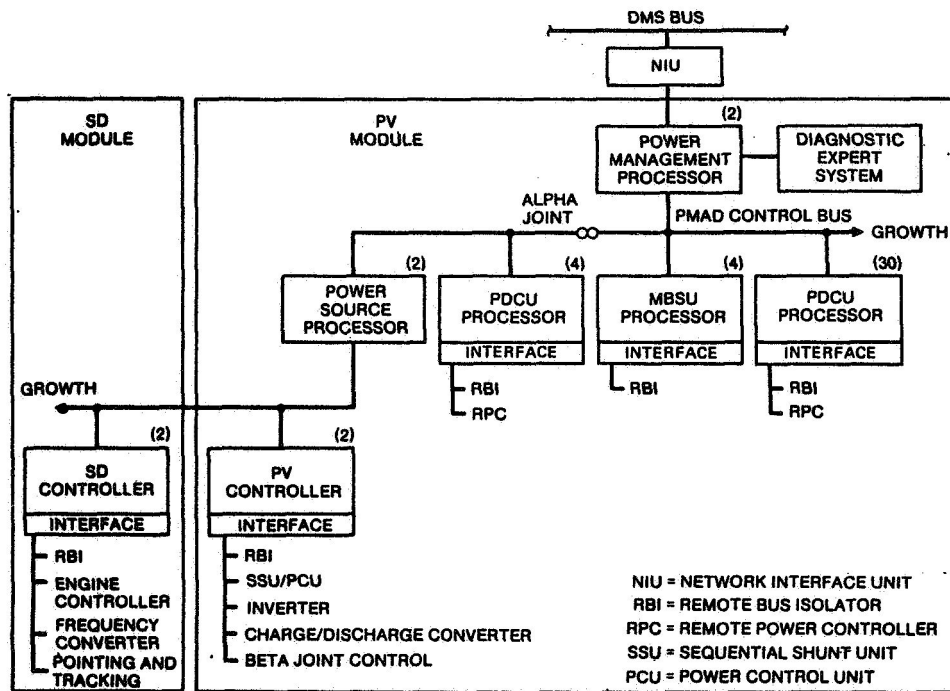Fig. 1. Space Station EPS Components Locations

Fig. 2. Control Functions

centered in the Power Management Controller (PMC), at the top of the hierarchy. It is at this level that interaction with a dedicated expert system is being considered.

The kinds of functions being evaluated for an expert system, operating as an extension of the PMC, include advanced health monitoring and trend analysis, failure mode analysis, equipment fault prediction. load flow analysis, and state estimation. For many of these functions, the mathematic calculations required are extensive. For example, some state estimation techniques require iterative manipulation of large line-

556

flow matrices (Ref. 10). Because of the need for the EPS expert system to utilize such computational data, there will have to be a close coupling of information between the conventional PMC processor and the expert system, with the PMC performing most calculations, leaving the expert system free for heuristic reasoning. Studies are currently being conducted to determine the best implementation approach for different types of expert systems. A load management expert system, for example, will require real-time data and might therefore be a candidate for a conventional processor approach (Ref. 2). A diagnostic expert system, on the other hand, requires a great deal of detailed information on the system configuration, models of performance, failure modes effects analysis (FMEA), etc., in a knowledge base that would run inefficiently on a conventional processor and might require a specialized symbolic or parallel processor.

Electrical Power System Evolution

The basic architecture for future EPS configurations will include enhancements to the IOC hardware and software architecture and functionality. The addition of a specialized expert system processor, such as a compact LISP processor, is a logical first step. Early enhancements are expected to involve the development and testing of a more advanced diagnostic expert system that will be stationed on the ground and receive its data from the Space Station via telemetry. To verify the performance of a prototype expert system, two sources of system data will be utilized: one will be the actual operating data from the orbiting station, and the other will be the results of running off-nominal tests in the Rocketdyne and the NASA Lewis Research Center (LeRC) test facilities. After operational experience is obtained from day-to-day interaction with the power system (failure rates, histograms of the types of failures, etc.), the ground-based expert system could become validated and ready for implementation on the station. The iterative development process will utilize the knowledge of power system experts during the initial stages of the program well before the Space Station is operational. The NASA LeRC Space Power Facility and the Rocketdyne Space Power Electronics Laboratory will be utilized to simulate failure modes, and to exercise diagnostic strategies for early knowledge-based technology assessment.

System Architecture

The IOC architecture represents a conventional distributed and hierarchical architecture approach that greatly simplifies the difficulties typically involved in adding functional capability by utilizing specialized processors. However, it is important in the EPS IOC architecture design to allow for the direct incorporation of specialized processor architectures for the power system. Issues of how to utilize LISP (Ref. 11-13) processors are now under consideration for military applications. The Texas Instruments Compact LISP Machine developed under DARPA funding represents an important step in utilizing advanced technologies for military aircraft and autonomous vehicle applications. This processor has been designed to be fully MIL-SPEC qualified and will undergo testing within the next few years. The bus architecture has been designed to accommodate the specialized bus requirements as well as providing for the addition of 1750A processor architectures running in parallel with a compact LISP processor. This technology represents a possible growth option for the Space Station EPS, and the utilization of such a backbone architecture is a potentially logical direction. Thus, at IOC, the PMC could be configured with a conventional processor architecture with the capability of adding a LISP machine or another appropriately configured conventional processor after initial ground testing of the system is completed. Alternatively, the IOC system management controller could be removed and replaced with a program management controller actually containing the LISP machine and software.

Diagnostic and Trend Analysis

Diagnostics of key power system components represent a significant requirement for the EPS (Ref. 6, 14, and 15). The safety, risk, and associated cost issues of power system component unavailability is prompting the development of a knowledge-based diagnostic system that detects deterioration of power system components in the early stages. Since typically, a human operator must diagnose a situation by interpreting what the actual condition of the equipment is from a number of measured variables, the diagnostic knowledge of the operator will be the critical link in making decisions for power system maintenance and operation. Steady-state system stability will also be an issue when considering

removing any unit from the interconnected grid (Ref. 13 and 16). Projected resource needs, as well as the availability of backup units, further complicates the picture. The utility industry has tended to counter these concerns by increasing the number of variables monitored and improving the quality of the display of these variables. This, however, is in contrast to the real need of an operator, especially under decision-making stress, to know the actual condition of the equipment (what is actually malfunctioning, not just which variables are abnormal). An on-line diagnostic system would provide the Space Station crew with a significantly higher level of information to allow them to make better quality decisions, especially under rapidly changing conditions.

Failure Detection, Isolation, and Reconfiguration for Growth

The Space Station power system will grow in size and evolve making it difficult to characterize a static maintenance problem. This makes crew training in the maintenance of particular systems difficult and could result in a major dependence on ground expertise. It also makes it difficult to specifically configure generalized automated test equipment if any shared resources are used. Requirements for automatic fault detection/fault isolation have been established for meeting Space Station availability requirements (Ref. 14 and 17). A typical failure detection and isolation configuration is shown

in Fig. 3. A base of historical performance data will be required for trend monitoring and incipient failure detection of subsystems (with similar requirements for health monitoring). Specific "signature" assessment modules must be defined for the distinct failure mechanisms of different component types—mechanical, electronic, fluid, etc. Figure 4 shows a diagram of this kind of diagnosis and performance monitoring.

The large mix of fault isolation procedures results in the requirement for a procedures storage system of some sort, and "user friendly" access to these procedures. The dynamics of the EPS could result in a configuration control problem in trying to keep the procedure storage system current. This would be solved in part by the use of knowledge engineering techniques to uncomplicate and speed the search for the required procedures.

Failure detection, testing, and maintenance have been categorized as knowledge-intensive and experienced-based tasks (Ref. 1). Although test procedures and maintenance manuals normally will contain recommended detection, localization, testing, and maintenance and repair actions, their use for the EPS does not ensure completion of troubleshooting and repair tasks in a timely fashion. Skilled power system maintenance staff, apart from using test procedures and maintenance manuals, use heuristics and an understanding of how the system works to solve component problems. For example, when a symptom such as
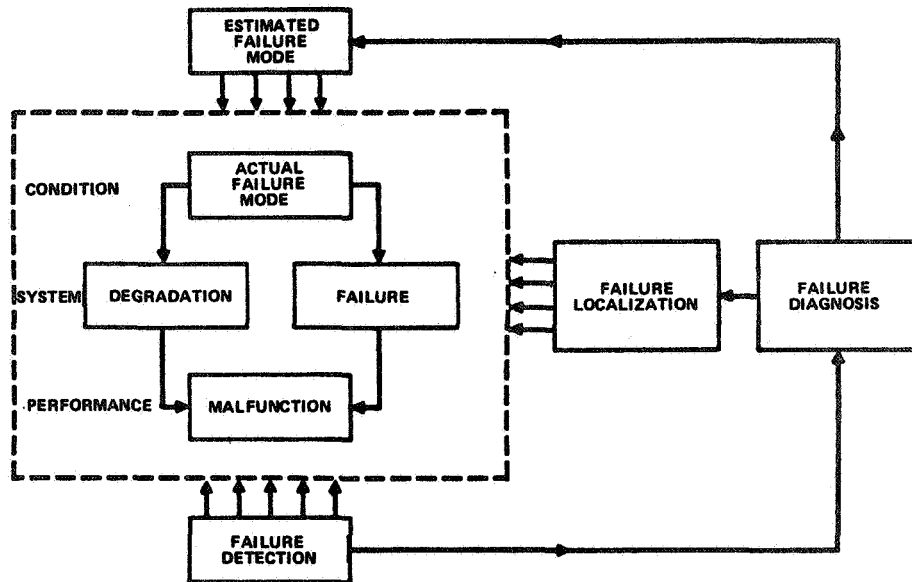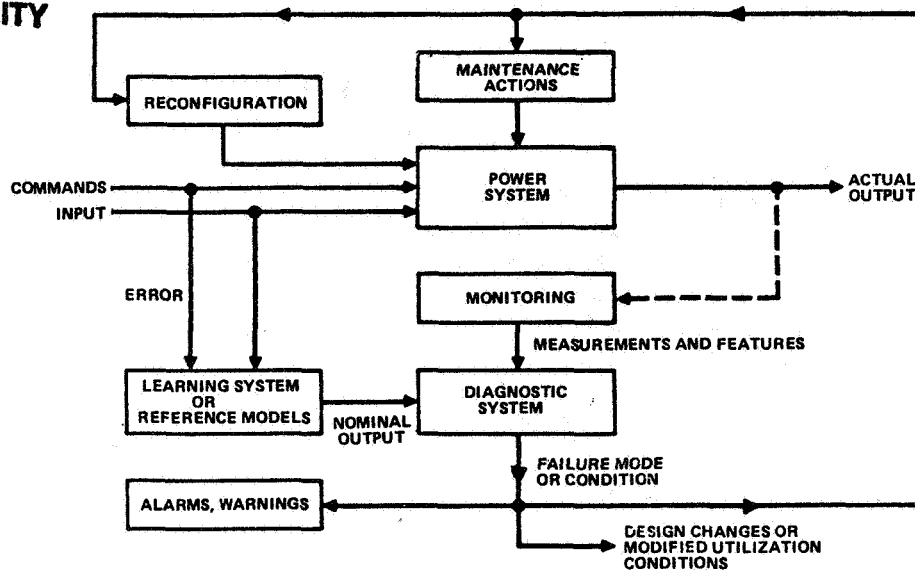

Fig. 3. Failure Detection and Diagnostics

Fig. 4. Diagnosis and Performance Monitoring

reduced power quality becomes noticeable in real-time and historical data, a power component diagnostician will begin a clearly defined list of troubleshooting procedures to localize the problem. However, if the typical troubleshooting techniques do not yield a solution, the experience of the diagnostician is required to initiate "nonroutine" troubleshooting. It is this type of knowledge that enables performance at an exceptional level. Based on years of experience, a highly skilled test/maintenance technician would develop the following traits:

● Familiarity with procedures and documented maintenance manuals

● Understanding of the relationships between observed symptoms and the malfunction of specific orbital replacement units (ORU)

● Intuitive understanding of how the system works

● Intuitive understanding of how the system will behave when certain subsystems or ORUs fail

The high level of performance of experts suggests that, confronted with a problem, they analyze the problem in a structured manner rather than randomly trying all possible alternatives. The identification of these tasks is crucial to the success of modeling the associated reasoning processes distinctly, and

modeling them as independent modules is extremely important to achieving a high degree of performance and modularity for future expansion and modification of the power system. Some of the above tasks identified are interpretation, diagnosis, troubleshooting, repair, design, planning and scheduling, monitoring, and reasoning with functional models. It is essential that an expert system be used that has a knowledge of the other subsystems and/or is capable of communicating through a shared-knowledge interface.

Failure detection, isolation, and reconfiguration (FDIR), known in the power industry as contingency management, deals with the problem of detecting deviations from normal behavior (which will be designated "failure modes") in a specified complement of system components (sensors, actuators). It includes isolating the particular component that has failed, and reconfiguring the power system by providing switching to redundant components or power buses. Recent developments in FDIR methods that may be applicable to the evolving EPS expert system can be conceptualized as consisting of four separate related stages:

1. Residual generation (using state estimation/model following)

2. Information collection (FMEA, and criticality of failure modes)

3. Decision making (determine if 1 and 2 indicate a failure mode

559

4. Reconfiguration (determine how the power system can be reconfigured in an "optimal" fashion to maintain satisfactory performance)

In modern avionics systems, failure detection and built-in testing (BIT) are the cornerstones used to signal when the principal path of information or data flow should be switched to an alternate backup path to preserve proper overall system performance (Ref. 18). While on-line BIT is usually used to rapidly uncover catastrophic or hard failures, other more sophisticated techniques (based on modern control/estimation/decision theory) are utilized to detect more subtle, or "soft" drifting-type failures. These are the failures that do not necessarily cause the system to shut down entirely but may still degrade system performance with the passing of time. The following detailed issues must be considered when implementing this kind of FDIR algorithm:

1. Nature of the soft failure (i.e., its type and severity)

2. Observability of the failure's effect within the measurement (i.e., degree of perceptibility)

3. Length of time required to accumulate enough data to register the presence of the failure in the presence of background disturbances such as quantization effects, sensor noises, response of the station data management system (DMS), computer cycle speed, etc.

4. Degree of distinguishability from other types of failures for unambiguous failure isolation

5. Ease of corrective actions (e.g., switching to an alternative analytically or functionally redundant path on-line or postponing repair until back at the maintenance module)

These considerations are important factors that contribute to an overall failure detection isolation and reconfiguration policy. However, the action of failure detection is fundamental to every system reconfiguration policy, and the

technological areas of failure/event detection are undergoing rapid change as new ideas enter the field.

Trend Analysis

Trend analysis requires the collection of data over long periods of time for detailed evaluation. Trend analysis operates on a single data series at a time, either by smoothing or estimating the parameters characterizing the shape of the time-dependent curve. Alarms result from extrapolations of such curves and from comparisons with fixed or variable thresholds. Trended data allows experts (expert systems) to make an intelligent decision on how much longer a failing component, or assembly, may be operated safely.

Since trend analysis relies heavily on historical data, the ability to store and retrieve data quickly is critical, particularly in space applications.

How much data should be analyzed is also important because there is little point in amassing so much data on a particular piece of equipment that retrieval becomes a major problem. There are several approaches that will continue to be evaluated at and beyond IOC:

1. Updating by exception – In this approach, after the baseline data have been verified over a period of time, the monitoring function continues but new data are not added unless it is at variance with the previous information.

2. Discarding old data – This approach is to save all new data and to discard or average away old readings. This procedure is currently favored and can easily be implemented on a computer-controlled diagnostic system.

Many techniques may be used to detect trends: slope computations, inflection point checks, least-square curve fitting, and state estimation methods. It is desirable, however, especially if large amounts of equipment are being monitored to maintain a low computation burden. For this reason, the simpler computational algorithms, such as the least-squares approach, are preferred options. Generally, a

curved or sloping line indicates a gradual degradation with time, and scattered data outside the expected distribution indicate a failure.

Other techniques, such as the minimal least-squares estimator and the Kalman-Bucy filter, which can be used to enhance or magnify the detectability of failure modes, are being investigated at Rocketdyne for their potential use in the EPS. They may prove to be powerful tools that can be implemented within the current design framework. But, no matter which methods are determined to be applicable, to respond to the anticipated Space Station maintainability requirements, collection and analysis of trend data will be implemented to the fullest degree feasible.

Conclusions

The importance of reliability, stability, and automatic operation of the electrical power system for all other subsystems on the Space Station makes it a prime candidate for the application of artificial intelligence techniques. Analysis of crew and ground-based resources have emphasized the need for advanced automatic control of the EPS at IOC, with an advisory diagnostic capability in an expert system for trend analysis, fault prediction, and component maintenance. Research at Rocketdyne and NASA/LeRC is proceeding in all of these areas to determine the optimum hardware and software implementations for such functions. The results of these efforts could do much to enhance the productivity of the Space Station as well as future manned space activities.

References

1. Pau, L. F., DTH, Rormosen 56, Kaarup DK 4540 Faarevejle, Denmark, "Survey of Expert Systems for Fault Detection, Test Generation and Maintenance," Expert Systems, The International Journal of Knowledge Engineering, Vol. 3, No. 2, April 1986.

2. Hendelman, D. A. and R. F. Stengel, Princeton University, Department of Mechanical & Aerospace Engineering, "Combining Quantitative and Qualitative Reasoning in Aircraft Failure Diagnosis," AIAA 85-1905, pp. 366-376.

3. Fink, P. K., J. C. Lusth, and J. W. Duran, "A General Expert System Design for Diagnostic Problem Solving," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 5, pp. 553-560, September 1985.

4. De Kleer, Johan, Xerox PARC, Intelligent Systems Laboratory, Palo Alto, CA, "How Circuits Work," Artificial Intelligence 24, pp. 205-280, 1984.

5. Stengel, R. F., "Artificial Intelligence Theory and Reconfigurable Control Systems," Princeton University, 30 June 1985.

6. Leinweber, D., "Real-Time Expert Systems for Space Station Process Control," LISP Machine, Inc., Los Angeles, California,."

7. Malik, J. and T. Lance, "An Expert System for Fault Management and Automatic Shutdown Avoidance in a Regenerative Life Support Subsystem," Paper #85-0333, ISA 1985.

8. Divita, Turner, P. R., "Autonomous Spacecraft Design Methodology," Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California.

9. Adams, T. L., B. D'Ambrosio, M. R. Fehling, and S. Schwartzberg, "An Artificial Intelligence Approach to Autonomous Power System Maintenance and Control," 1985 Society of Automotive Engineers, Inc.

10. Kusic, G. L., Computer-Aided Power System Analysis, Prentice-Hall, New Jersey, pp. 347-368, 1986.

11. Hirsch, Abraham, "Tagged Architecture Supports Symbolic Processing," Computer Design, pp. 75-78 and 80, June 1984.

12. Kraus, T. W. and T. J. Myron, "Self-Tuning PID Controller Uses Pattern Recognition Approach," Control Engineering, June 1984.

13. Klos, L. C., J. A. Edwards, and J. A. Davis, "Artificial Intelligence—An Implementation Approach for Advanced Avionics," AIAA 83-2401, pp. 300-307.

14. NASA Technical Memorandum 84766, _Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy, Volume I - Executive Overview_, Advanced Technology Advisory Committee, Submitted to United States Congress, 1 April 1985.

15. Zimmerman, W. F., J. Bard, and A. Feinberg, "Space Station Man-Machine Automation Trade-off Analysis," JPL Publication 85-13, NASA Jet Propulsion Laboratory, California Institute of Technology.

16. Stevenson, William D., _Elements of Power System Analysis_, McGraw-Hill, 1982, pp. 373-421.

17. NASA Johnson Space Flight Center, Minutes of Meeting, Space Station Program Level B SIB, 23 April 1986, p. 7.

18. Pau, L. F., "Failure Diagnosis and Performance Monitoring," _Control and Systems Theory_, Vol. 11, Marcel Dekker, Inc.

A METHODOLOGY FOR MULTIPLE RULE SYSTEM INTEGRATION

AND RESOLUTION WITHIN A SINGULAR KNOWLEDGE BASE

Frank N. Kautzmann, III, Ph.D.
MITRE Corporation
1120 NASA Road 1
Houston, Texas 77058

Topics: Philosophical and Scientific Foundations,
AI Architectures and Languages, Knowledge
Representation, Mathematical Logic, Proof
Theory, Program Verification, Set Theory

ABSTRACT

Expert Systems which support knowledge representation by
qualitative modelling techniques experience problems, when called
upon to support integrated views embodying description and
explanation, especially when other factors such as multiple
causality, competing rule model resolution, and multiple uses of
knowledge representation are included.   MITRE Corporation, in
conjunction with several directorates of NASA at Johnson Space
Center, is currently developing a series of prototypes to
demonstrate the feasibility of automating the process of systems
engineering, design and configuration, and diagnosis and fault
management. It is important to study these processes as they
pertain to the analysis of design trade-off issues, knowledge
capture and Space Station planning and assessment. The MITRE
effort will involve not only a generic knowledge representation;
it must also support multiple views at varying levels of
description and interaction between physical elements, systems,
and subsystems. Moreover, it will involve models of description
and explanation for each level.   This multiple-model feature
requires the development of control methods between rule systems
and heuristics on a meta-level for each expert system involved in
an integrated and larger class of expert system. This paper
describes the broadest possible category of interacting expert
systems and proposes a general methodology for the knowledge
representation and control of mutually exclusive rule systems
within the context of a single knowledge representation scheme.

INTRODUCTION

NASA has been directed by Public Law 98-371 to automate, as much as technically and economically feasible, appropriate aspects of space flight. This agency-wide charter necessarily involves the investigation of contemporary AI processes and methods. The generic focus is primarily on the planned Space Station, although current Space Shuttle operations evaluation is relevant due to the wealth of expert knowledge already in place. The majority of tasks that are candidates for automation involve expert knowledge. The sub-field of expert systems within AI can thus be seen as a fruitful and primary methodology for investigation. Secondarily, it is a relatively small step to determine that prototypes for each separable expert system will be required. There are two reasons for this conjecture. First, prototypes are excellent pragmatic mechanisms for the demonstration of feasibility. Second, because of the necessary interaction with domain experts in prototyping an expert system, expert knowledge is translated into a form that is formalized for future examination and use as expressed in the form of rules and logical structures. This benefit is important due to a current concern within NASA and industry about the loss of retiring experts' knowledge. By moving from an expert's knowledge to a reproducible form of expressing that knowledge, such fears are alleviated. Examples of candidate knowledge areas to capture within NASA will include design engineering, systems engineering, fault diagnosis, testing and configuration management. A current AI project within MITRE addresses the design engineering knowledge capture process utilizing an amalgam of CAD/CAM and expert systems technology [1].

It is reasonable to conjecture that separable, standalone single domain expert systems are but an initial phase along the process of providing, say, a Systems Engineering Assistant. That such an Assistant will necessarily embody and interact with the single-domain expert systems at various levels is not to be overlooked as a logical consequence. By further extension of the preliminary conjecture, it can be seen that there are at least three possible areas in which the development of such a class of "Assistant" expert systems would be useful:

. Systems Engineering
. Design and Configuration
. Diagnosis
  - Fault Management
  - Testing

Assistant-type experts in each of the three areas will require interaction with separate single-domain expert systems for two reasons. First, because each of the three areas must also provide facilities for simulation and modelling. Second, because knowledge about how to perform satisfactorily involves knowledge concerning the inter-relationships between expert domains. Thus, from within a Space Station context, each of the three areas must share elements and interact with the following ten major single-domain expert systems:

. Communications & Tracking
. Data Management System
. ECLS - Environmental Control and Life
  Support
. Electrical Power
. Guidance and Navigation
. Man
. Propulsion
. Thermal
. Structure
. Payloads

The ten systems have further taxonomy at the sub-system and component level. Additionally, there are multiple cross-links between subsystems and components at the juncture of components and subsystem interactions [Figure 4]. The former three areas and the separable single domain expert systems must interact on the boundaries of overall system performance as an integrated expert system, as noted. At a minimum, each of the ten single domain expert systems must share a common knowledge representation with the larger class of expert systems and with each other.

PROBLEM STATEMENT

The initial problem statement for the success of the larger "Assistant" class of expert systems involves specifying the current tasks and goals of the three major functional areas. From this point onward, "Expert System" shall refer to an expert system that embodies the performance objectives of the three major areas. "System" shall refer to smaller domain-specific expert systems with "subsystem" referring to elements and components of the specific local system.

The major goal of each expert system may be viewed as determining the performance of the complete system, subsystem or element in some given configuration and state, given specified inputs and specified outputs. Each expert system's task area is the determination of system performance with differing perspectives or views.

● In Systems Engineering activities, the goal is the determination of precisely how and under what conditions the subsystems and the parts can operate together to produce, preserve, or achieve some desired outputs or conditions of the overall system. Thus a methodology is required, along with a knowledge representation, that can be combined with various models of behavior to simulate alternate system configurations; thereby determining how these alternate configurations respond to various inputs and demand constraints. Secondly, an analysis of the interfaces, interactions, or connections among the subsystems, with particular emphasis on problem identification and constraint violations at the interface level, is required.

● In the "configuration" domain of the Design and Configuration activities, the goal is the determination of whether or not it is possible to structure or configure a system in such a fashion that it preserves or achieves a desired set of outputs or conditions. This determination must be made within given resource constraints, such as time or power, along with expected inputs and demands. Again, there is a need to model and simulate possible configurations with varying inputs and resources. The solution space relative to resources may be less constrained than that of the Systems Engineering domain. The area of interest and emphasis is on internal couplings or structures relative to alternative resources for achieving a given structure or series of system goals. The "design" domain of the Design and Configuration activities involves planning for a correction to a configuration, a repair, or fault-tolerating action relative to a system, subsystem element or alternative(s).

● In Diagnosis - Fault Management activities, the goal is the determination of the precise configuration, including fault modes, with appropriate inputs or conditions, that could produce a given set of outputs or conditions. The Fault Detection emphasis is on determining configuration states that elicit a fault or a series of fault conditions, and then isolating the respective faulty component states along with their state-dependent interactions and transfer functions. Corrections are actions on the system, operating in con-text, that permit changing either its state or its response in some predicted fashion, given that the assumptions about the system's prior configuration are correct.

● In Diagnosis - Testing activities, the goal is the generation and validation of test hypotheses wherein a system response is predicted based upon changing configurations, which are themselves based upon conditional inputs or states. The tests are actions on the system to evaluate system response relative to

conditional, unknown or hypothetical prior conditions or states. Predicted sets of expectations as to system behavior/response are also alternative sets of expectations that are hypothesized in order to effectively perform a differentiating test relative to actual responses.

Common Elements

The common elements of the three performance goals require movement from a given current configuration to a desired configuration (or alternate) that meets specifications, goals, or needs relative to a higher goal or to some hierarchy of goals, inputs or states.

The integrated expert system must accommodate the description, determination, design and analysis of system, subsystem and component configurations relative to constraints and goals specified either as inputs or outputs. This involves a knowledge representation that supports the above modelling and simulation activities, each with it's respective different emphasis. This requirement further decomposes into the requirement for a knowledge representation that accommodates these features:

● System, subsystem, and components must change state and internal configuration. This requires some transfer function to define such changes across boundaries; and also within a hierarchy of descriptions and explanations.

● The transfer functions must be able to characterize the internal processes of the system, subsystem, or component, given a particular state or confiuration of the same.

● Additional functions are necessary to describe coupling, direction or interactions that can characterize the constraints, parameters, or specifications on the required inputs and outputs and their respective directions of coupling.

● Goals of the system, subsystem, or components must be translated from and between all elements relative to a system, subsystem, or component state/configuration. Determination of side effects and sub-goal decomposition is also required.

GENERAL APPROACH

The conjectured task naturally divides into two phases. The initial phase will involve the construction of preliminary single-domain expert system prototypes in the ten areas already noted along tradi- tional lines. The non-traditional aspect of

567

the first phase would involve the designing and utilization of a knowledge representation scheme that is capable, in principle, of being extended for future single-domain expert systems and sub-; systems. The degree of difficulty in performing the first task is minimal, although it may be time consuming. It should also be noted that rapid prototyping schemes with alternate knowledge representations would inhibit the success of this phase. If, for example, one selects a knowledge representation scheme that works for but one of the ten single-domain expert systems, then all others to that point must be redone, or enlarged to accommodate the variance. Thus, success for this phase is requires initial selection of a valid knowledge representation scheme in order to avoid permutational reworking for each separable system. This step requires little else but good systems engineering practices and data abstraction in the selection process.

The second phase also has requirements for knowledge representation at a higher level, since it must involve cross-domain expert system interaction at subsystem and component levels and presupposes an initial knowledge representation adequate to support such a functional requirement. Since the generation of multiple single-domain expert systems has been done many times before, this paper concentrates on the second aspect of the conjectured prototype and proposes a suggested methodology for achieving cross-domain expert system interaction.

This second phase's primary objective involves the generation of a single methodology to achieve interaction among closely related cross-domain separable expert systems with multiple views, models, performance requirements, and transforming operations capable of operating at multiple levels of description and explanation. Achieving this requires, in part, adopting forms of knowledge representation that can operate at multiple meta-levels. These forms must be, beyond object-oriented semantic and syntactic specific, of a sufficient formal and mathematical nature in order to support the defined problem space transformational requirements. This further requires that the knowledge representation and the transforming processes themselves be valid. 'Validity' involves four formal requirements: consistency, completeness, soundness, and precision. 'Consistency' requires that similar answers produce similar results. 'Completeness' requires that everything true is derivable. 'Soundness' requires that everything derivable is true. 'Precision' is required within ES's that deliver probabilistic or qualified judgements. The ability of the prototype methodology and knowledge representation to meet the criteria of validity enables, in principle, the completed system to be transformed vis-à-vis alternate methods, to other representations that are equally valid across related domains as well as within the single domain.

Background to Approach

A strong parallel exists between automated program verification and specification techniques and single-domain expert systems. At the lowest level of parallel, there are merely linguistic and data typing parallels. Within the mathematical and logical foundations the parallels are more striking and offer an insight into cross-domain expert systems interaction. That the higher class of expert systems outlined herein, share the same elements with single-domain expert systems is also true by extension. That the parallel claim is the case is evidenced by Figure 1. The further language-based parallel involving syntax and semantics between the two areas is also apparent. So are the formal and philosophical foundations which involve mainly mathematical logic(s) and set theory. The long tradition in philosophy of a distinction between "semantic" and "syntactic" in many areas of inquiry does not necessarily relate, except within a limited domain, to a linguistic theory of meaning. That the linguistic model, along with further investigations in "natural language" processing, is pre-eminent in expert systems research is self-evident. That the linguistic focus is useful is not in dispute for knowledge capture. The extension of such a model throughout expert systems may account for the lack of acknowledging the parallel domains in formal systems, with their own syntactic and semantic issues, by expert system researchers.

That this parallel has not yet been either drawn explicitly or extensively investigated before for either single-domain expert systems, and most importantly, for expert system cross-domain interaction, is evident by the conspicuous absence to such parallels within contemporary AI expert system's literature and journals. [The closest parallel to date is between automatic theorem proving techniques used as a heuristic method within expert systems].

The reasons for this absence are possibly manifold. It is tentatively suggested that the absence is accounted for by the "natural language" processing model for mind that preempts other models by AI researchers. Likewise, non-Philosophers, namely cognitive psychologists, linguists, anthropologists, and to a certain extent, computer scientists, generally maintain a belief that the expert system knowledge acquisition process, as a process, offers a window to the mind. Noam Chomsky's work in linguistics has offered a model of species differentiae between man and the brutes. The salient feature is seen as centered upon man's inherent ability to acquire natural language capabilities. Ergo, this becomes the ontological basis for the central focus of the "natural language" processing theory of mind in expert system

research and cognitive psychology. This model also embodies the underlying and deeper, philosophically, linguistic theories of meaning.

Further, AI research in expert systems focuses upon confirmation of (competing) natural language acquisition models in order to further substantiate in turn, deeper models of "reasoning". These deeper theories in turn rest upon related hypotheses concerning the nature of mind. The epistemological bases in turn refer to underlying metaphysical theories of existence, structure and causality. The further foundational appeal is within Philosophy of Science and to a general model theory of meaning expressed as a linguistic model.

That there are important underlying similarities with the philosophical, logical and mathematical nature of expert systems themselves and the argued parallel for possible future progress in cross-domain expert system interaction based upon automatic program verification and theorem proving techniques warrants further in-depth investigation by AI.

An additional reason why the fact that this parallel has not been drawn before by expert system AI researchers can be possibly accounted for by the recharacterization of the formal methods that have yielded results in the field of expert systems into a form that is foundationally seductive to AI researchers for their broader scope of investigation of human reasoning.

To wit, "Forward chaining", "Backward chaining", or "Forward reasoning" and "Backward reasoning" mean nothing more than the utilization of the predicate logic techniques of modus ponens and modus tollens.

For example, every single-domain expert system "inference engine" is but a separate instance of modus ponens used as a control structure invoking other instances of either modus tollens or modus ponens operating at the rule level, which in turn, operates using variable assignments at the object level of description for binding. That there exists a fundamental contradiction when these facts are conjoined with claims from the same re-searchers that formal methods yield no promise for cross-domain expert system interaction is obvious. It is suggested here that what is amiss is the underlying, unstated premise that only confirmation of theories, in some model building or paradigmatic sense, related to human-centered reasoning models of explanation, are use-ful for the stated purposes of AI. That a strong parallel exists between the functioning underlying structures of formal logical methods, mathematical logic and set theory and the domain of actually working single-domain expert systems is not to be

refuted. Expert systems work precisely because of this continuity and dependence upon the underlying mathematical-logical structures. This is particularly true when such systems "reason about themselves" in some meta-level fashion such as exemplified in qualitative modelling.

From a differing perspective, the absence of such an explicit parallel by research- ers in theorem proving and program veri-fication to expert systems might be based upon the recognition by the former re-searcher's of the domain expert's inter-action in generating an object-oriented, for example, semantic domain(s). Or alternatively, precise recognition that verification and theorem proving deal with an inherently mathematical structure and thus lack the linguistic model confirmational orientation of expert systems.

The parallels operating from the domain expert's knowledge, once it has been captured and formalized (logically) into rules, into an object-oriented domain, which can then be operated on syntactically, will be developed from the perspective of similar formal parallels in automated program verification and theorem proving. That such can be applied to expert system's cross-domain interaction will be further exemplified from the argued methodological premise.

As a preliminary in this direction, Figure 1 is offered as an exemplification of all single-domain expert systems, and by ex-tension, to cross-domain expert systems. Methods of search and knowledge-based techniques for reasoning and the representation of knowledge are addressed. The figure is particularly important relative to the proposed general methodology. The Philosophical foundations are presented as a map to ascertain the relative position of issues and methodologies referenced within expert systems.

Outline of Approach

The initial approach for the conjectured prototype utilizes a variety of models of description and explanation over a wide variety of types. These include logical models, analogue models, semi-formal or mathematical models, simplifying models, and theoretical models as seen in Figure 4. In order to support this wide variety, multiple versions of declarative encoding for knowledge representation were be selected. In declarative encoding semantics are used in a few global procedures. By formally transforming between levels of descriptions and explanations (meanings), the problem of different domains will be reduced to formal representation and transformation at a meta-level. A fruitful first step is in selecting a knowledge

571

representation method that will ground the proposed mathodology within the realms of logic and mathematics. Semantic nets are ideal for this purpose. There are two reasons for selecting semantic nets for initial knowledge representation:

. Semantic networks are powerful. Proof exists that they are Turing-complete [2]; thus methods in finite automata are available for future use.

. Rapid prototyping of alternate knowledge schemas is permitted [3]. (This is a side benefit and not central to the proposed expert system methodology, except in the cases of the underlying single-domain expert systems).

Rule System Representation

Once the expert's knowledge has been captured in the form of rules, trans-formational consistency demands that the rule system's formal representation be also Turing-complete. LISP is known to be Turing complete and will be selected as the language of choice. The general method of rule application/selection for heuristics follows 'production rule systems', a formalism proposed by Emil Post in 1943. ["Heuristics" in this sense refers to control methods, not to special meta-rules within the rule classes]. That production rule systems are also known to be Turing-complete has been established [4]-[6]. Knowledge representation, rule writing, and heuristics thus all share, at a minimum, the computationally powerful and sufficient aspects of Turing-completeness, including recursion.

In order to achieve this common aspect, embedded rule systems in commercial AI development tools need to be avoided unless they are provably known to be extensible and complete with respect to the Lambda Calculus and also contain a definitive pathway to LISP. [That the Lambda Calculus extension is also a requirement fits within contemporary research in program verification, as elsewhere referenced].

A further advantage of using production systems for rule representation is that this method does not rule out the possibility and speed advantages of later transformation using procedural encoding following the proposed methodology. Production systems provide an effective bridge between the declarative and procedural encoding methods. It is by means of this bridge that future migration of an existing expert system so designed, can migrate cross-language [and cross-hardware], as will be outlined.

Initial Knowledge Architecture

A conceptual prototype representational scheme is given as an overview in Figure 4. In the figure the models, components, systems and subsystems are objects. The principles utilized across models refer to methods of transformation in a hierarchy of analogical pattern-based series of meta-rules [7]. The constraints on models are also pattern-based at the meta-level and object-oriented at the primary description level. Goals relate to inputs, outputs, states and/or conditions. State-history relates to operational status of the component or subsystem and is a tree-oriented directed acyclic graph, or an alternative abstract algebra state machine [8].

Pattern Matching

The models, as identified in Figure 4, are called based upon either states, inputs, outputs, goals or conditions. It is important to recognize the element of analogical pattern-matching and reasoning for the models' selection and its application.

Furthermore, the various types of 'models', (for example, qualitative models), have an algebra of operators for single-point causal analysis and sequencing. That such a single-point causal model is not sufficiently rich enough to address all issues arising from cross-domain expert systems forces the requirement for additional model types. Other models of causality may, for example, be required. However, all model's operators form, at a higher level, a unique syntax and semantic for later transformation. They are formal structures, in and of themselves, apart from the initial object-oriented domain and form the basis for valid sets of transformational operators in each domain.

Abstractly, what is most important are the patterns of reasoning applied within a given rule system or domain in terms of successful heuristics. Patterns of reasoning are themselves formal structures; such structures are capable of examination by a wide variety of formal mathematical-logical methods.

These formal structures are symbolic in that they permit more than purely formal deduction. It is important to note that there are two levels of reasoning involved. The first, with respect to the actual knowledge domain, and the second with respect to the transformational aspects of that same knowledge into forms acceptable for all levels and for all views.

This second aspect of reasoning requires the development of problem-solving procedures which take into account the robustness

of the separate rule systems and heuristics. This requirement places a further restriction on the development of problem-solving procedures; they must guarantee the mechanization of formal cross-domain proof procedures [9]-[11].

Satisfaction of the proof procedure requirement involves devising effective problem-solving methods based on the structure of the reasoning model, and the associated system of knowledge representation. The ease with which it is possible to put such methods to work will depend upon how systematically the formal properties of each reasoning model's domain has been analyzed.

SPECIFIC APPROACH

The general problem space of the Space Station prototype relates to natural language domains of expertise and also the more mathematical-logical aspects of formal languages and semantics as seen in Figure 2.

Knowledge representation, using semantic nets and declarative encoding of rules within domains, with common tree-oriented data structures, permits capturing the patterns of success involved in each state or condition, as well as the rules and heuristics. This permits specification of the patterns, and the associated facts in the data structures, to be modelled as proof-assertions, as seen in Figure 2.

Figure 3 outlines a general methodology for the transformation between logical systems while preserving the semantics of each expert system domain. The stylized diagram involves the use of transformational grammar terminology to describe levels of syntactic processing, although transformational formalisms are not used to define the processing. Source text, produced by a single-domain or otherwise, expert system, either as output or input, via the proper logical system, is concrete syntax. Parsed concrete syntax is termed surface abstract syntax following surface structures in transformational grammars. These translations are context-free; i.e., variables are still variables in the same order following these two steps. Surface abstract syntax is transformed in non-context-free fashion into deep abstract syntax. As is required in the process of semantic analysis, each declaration of symbols and variables is processed and every occurrence of a symbol is identified with the information in its declaration. Logical transformations are performed as structural transformations. [12]

Techniques for reasoning by analogy vis-a-vis proof-theoretic techniques should permit aspects of common-sense reasoning to apply at the knowledge based object levels in conjunction with

validation of cross-domain proof-assertions. This would involve the development of equivalent means of traversing both the rule base and the object attribute-values, as outlined.

The integration of cross-domain knowledge in the form of meta-level rules is viewed as an investigation into domain-specific syntactic and semantic language transformations, involving higher-order logics, such as combinatory logic, or the S-K combinator calculus for the meta-level representation of heuristics and rules [13]-[16].

The formal proof-theoretic aspects of this level of formalism can, in principle, accommodate specification and verification, design validation, and generate search methods and 'knowledge-based' reasoning about oneself in closely related knowledge domains automatically. Fault detection and analysis can be performed at a higher level than the domain level, given that the relationships between elements as behavior rules can be formalized and transformed, using analogies of function and structure.

Updates to states can be viewed as tree-insertions in a derived search/resolution graph of the actual domain specific rules and heuristics. Traversing the graph, then transforming the sequence into the rule system domain, permits viewing the overall system according to interest, intent or relevant model.

Obtaining these results involves creating a separate proof-system for the transformational rules, patterns and sequences, taking advantage of the formal-mathe-matical nature of the prototype's semantic nets and production rule systems in the form of a high level language analyzer and interpreter.

In particular, understanding logical-con-sequence semantics does not require the construction of specialized models in lattice theory or category theory [17]. Nor does it involve, as is the case with abstract data types, initial or final algebras [18].

Specifically, a rule system, for a given state, given inputs, outputs, goals, plus interactions or couplings, can be viewed as a given set of assertions akin to a declarative program. The logical consequences of such a set of assertions are all the additional assertions that must be true whenever the assertions of the rules are true. Strictly speaking, an assertion in the form of A = B can be seen as a logical consequence of a set F of equations or transformational operators if and only if, in every algebraic interpretation for which every equation in F is true, A = B is also true. This can be verified at the object level in each knowledge domain.

There are a wide variety of logical languages that have be used to write assertions of rules, conditions and states, such as the first order predicate calculus as implemented by PROLOG. Such logical (formal) language constructs as output, may be combined with other assertions, to assist in non-context-free solutions or verifications of prior states or conditions with associated trees and mappings reducing the search space. It is possible in principle to utilize such a language of transforms to assist in rule verification at the domain level.

Meta-rules are also non-context-free. It is possible to relate the patterns that are embedded in these meta-rules in a context-free implementation as meta-level heuristics and rules. The danger of using the meta-rules themselves as search/resolution mechanisms must not be over-looked. Their use is limited to assisting transformations between domains using the formalisms that are inter-derivable and reducible to the domain-specific rules as forms of proof mechanization techniques. These techniques themselves provide the firing sequence (heuristics) of domain specific rules as meta-level heuristics.

SUMMARY AND CONCLUSIONS

A methodology is being investigated for cross-domain interaction between related expert systems utilizing a single knowledge representation. The methodology utilizes various formal and logical techniques for examining rule systems and associated object representations.

A methodology for deriving states, conditions and transforms between levels of behavior and function involves utilizing the Turing-complete aspects of semantic nets and production systems of a formal and logical-mathematical nature at several degrees of abstraction above each sub-system, using the meta-level approach is suggested as fruitful for cross-domain expert system interaction resolution. That such an approach also holds promise as a general methodology for the addition of rules into existing single-domain expert systems and the attendant modification of inference engines is also noted.

It is anticipated that the best foundation for a symbolic computing formalism is probably a layered language approach, each layer containing a subset of symbols that produce all of the desired determinate computations and transformations, and something from the S-K combinator calculus [19] to be used in the infrequent cases where analysis of truly indeterminate behavior is required.

576

That there may be other layers of disciplined behavior that should be covered by simple sub-languages, rules and associated graphs that produce the desired associated transforms with an attendant search/solution sequence, as part of the proof-procedure of validation at higher levels of abstraction, is admitted.
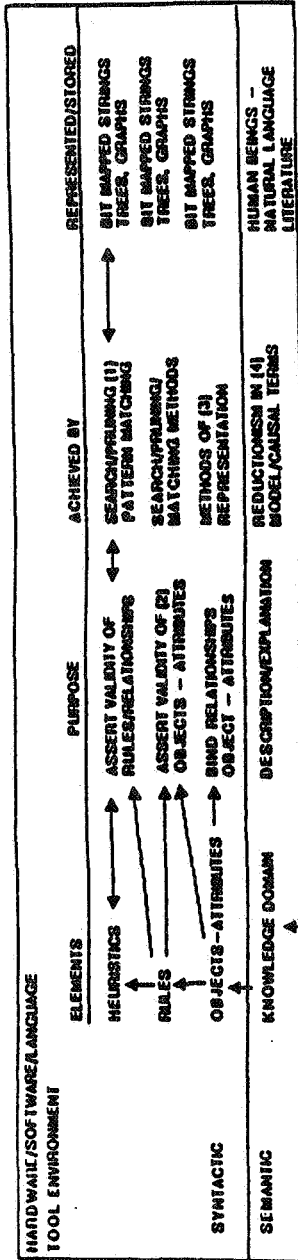
ACKNOWLEDGEMENT

REFERENCES

(1)   Crouse, K., and Spitzer, J., "Design Knowledge Bases for the Space Station", Robotics and Expert Systems, 1986, Proceedings of ROBEXS '86, Sponsored by Instrument Society of American and ISA's Clear Lake-Galveston Section, June 5-6, 1986.

(2)   Nilsson, N., Principles of Artificial Intelligence, Springer-Verlag, 1982.

(3)   Hayes-Roth, F., Watterman, D. A., and Lenat, D. B., Building Expert Systems, Addison-Wesley, 1983.

(4)   Post, Emil L., "Introduction to a General Theory of Elementary Propositions", American Journal of Mathematics, Vol. 43, pp. 163-185, 1921.

(5)   Post, Emil L., "Formal Reductions of the General Combinatorial Decision Problem", American Journal of Mathematics, Vol. 65, 1943.

(6)   Anderson, J., Memory and Thought, Erlbaum Associates, 1976.

(7)   Chouraqi, E., "Construction of a Model for Reasoning by Analogy", Proceedings of ECAI-82, Orsay, France, 1982.

(8)   Scott, D. S., and Strachey, C., "Toward a Mathematical Semantics for Computer Languages", Proc. Symp. on Computers and Automata, Polytechnic Institute of Brooklyn, Vol. 21, pp. 19-46, 1971.

(9)   Gentzen, G., "Untersuchungen uber das logische Schliessen", Mathematische Zeitschrift, Vol. 39, pp. 176-210, 405-431, 1934-1935.

(10)   Ajsukiewicz, K., "Classification des Raisonments", <u>Studia Logica II</u>, Warsaw, Poland, 1955.

(11)   <u>Logic, Form and Function: The Mechanization of Deductive Reasoning</u>, Elsevier North-Holland, 1979.

(12)   O'Donnell, Michael J., <u>Equational Logic as a Programming Language</u>, MIT Press, Cambridge, Massachusetts, 1985.

(13)   Giannini, P., and Longo, G., "Effectively Given Domains and Lambda Calculus Semantics", Preprint, <u>Dipt. Informatica</u>, Corso Italia, Vol. 40, 56100 Pisa, Italy.

(14)   Cohen, L. J., Pfeifer, H., and Podewski, K. P., (eds), <u>Logic, Methodology and Philosophy of Science VI</u>, North Holland, Amesterdam, 1982.

(15)   Suppes, P., et al., <u>Logic, Methodology and Philosophy of Science IV</u>, Studies in Logic 74, North-Holland, Amsterdam, 1973.

(16)   "Combinatory Logic as a Semigroup", abstract, <u>Bulletin American Mathematics Soc.</u>, Vol. 43, pp. 333, 1937.

(17)   Terlouw, J., "On Definition Trees of Ordinal Recursive Functionals: Reduction of the Recursion Orders by Means of Type Level Raisings", <u>J. Symbolic Logic</u>, Vol. 47, pp. 395 403, 1982.

(18)   Stenlund, S., <u>Combinators, Lambda-Terms, and Proof Theory</u>, D. Reidel Publishing Company, Dordrecht, Holland, 1972.

(19)   Goguen, J. A., "Abstract Errors for Abstract Data Types", <u>IFIP Working Conference on Formal Description of Programming Concepts</u>, E. J. Neuhold (ed), North-Holland, 1977.

(20)   Clarke, T. J., Gladstone, P. J. S., MacLean, C. D., and Norman, A. C., "SKIM-The S,K,I Reduction Machine", <u>1980 LISP Conference</u>, Stanford University, pp. 128-135, 1980.

FIGURE 1 EXPERT SYSTEMS ABSTRACT REPRESENTATION

EXPERT SYSTEM

HARDWARE/SOFTWARE/LANGUAGE
TOOL ENVIRONMENT

ELEMENTS	PURPOSE	ACHIEVED BY	REPRESENTED/STORED
HEURISTICS	ASSERT VALIDITY OF RULES/RELATIONSHIPS	SEARCH/PLANNING (1) PATTERN MATCHING	BIT MAPPED STRINGS TREES, GRAPHS
RULES	ASSERT VALIDITY OF (2) OBJECTS – ATTRIBUTES	SEARCH/PLANNING/ MATCHING METHODS	BIT MAPPED STRINGS TREES, GRAPHS
OBJECTS–ATTRIBUTES	BIND RELATIONSHIPS OBJECT – ATTRIBUTES	METHODS OF (3) REPRESENTATION	BIT MAPPED STRINGS TREES, GRAPHS

SYNTACTIC

SEMANTIC

KNOWLEDGE DOMAIN — DESCRIPTION/EXPLANATION — REDUCTIONISM IN (4) MODEL/CAUSAL TERMS — HUMAN BEINGS – NATURAL LANGUAGE LITERATURE

PHILOSOPHICAL FOUNDATIONS

EPISTEMOLOGY
- NATURE/SCOPE OF KNOWLEDGE
- PRESUPPOSITIONS AND BASES
- RELIABILITY OF CLAIMS OF KNOWLEDGE

METAPHYSICS
INQUIRY INTO THEORIES OF:
- EXISTENCE, THING PROPERTY, EVENT
- MIND/MATTER/ SPACE/TIME
- CAUSATION/CHANGE
- PARTICULARS/ UNIVERSALS

LOGIC/MATHEMATICAL LOGIC
FORMAL SYSTEMS
- PROOF THEORY
-- SET THEORY
-- MATHEMATICS
FORMAL METHODS
- DEDUCTION
- INDUCTION

METHODS OF KNOWLEDGE
VALIDATION – TRUTH/ MEANING

THEORIES OF TRUTH
- CORRESPONDENCE
- COHERENCE
- PRAGMATIC
- SEMANTIC
- PERFORMATIVE
- EXISTENCE

(4) SCIENCE
- PHYSICS
- PSYCHOLOGY

THEORIES OF MEANING
- REFERENTIAL
-- MEANING AND REFERENCE
-- REFERENCE AND SEMANTIC RELATIONS
- IDEATIONAL
- STIMULUS – RESPONSE
- NATURAL LANGUAGE
-- MEANING – USE
-- MEANING AND LINGUISTIC ACTS
-- USES OF WORDS
-- RULES AND REGULARITIES
-- CRITERIA OF SYNONYMY

(1) METHODS OF SEARCH
- TOP–DOWN, BOTTOM–UP,
- TREE/GRAPH TRAVERSALS FLATTENED TREE
- VARIANTS OF DEPTH VS. BREADTH

(2) METHODS OF REASONING
- CLASSICAL LOGIC
- NON–CLASSICAL LOGICS
- PLAN GENERATION
- NON–MONOTONIC AND COMMON–SENSE REASONING
- PATTERN–DIRECTED INFERENCE

(3) METHODS OF REPRESENTATION
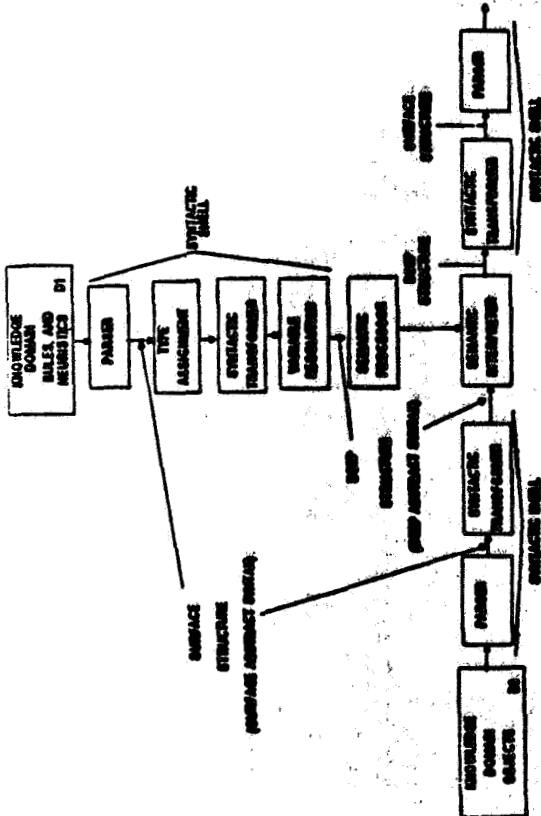- SEMANTIC NETS
- PRODUCTION SYSTEMS
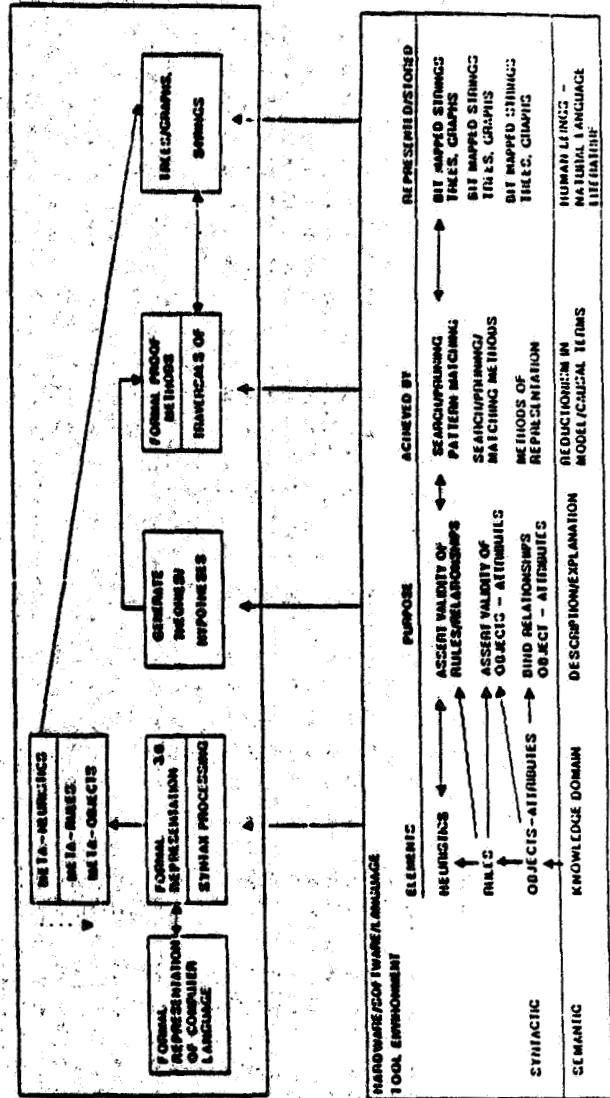- ANALOGUE REPRESENTATIONS
- FRAMES
- SEMANTIC TRIPLES, ETC.
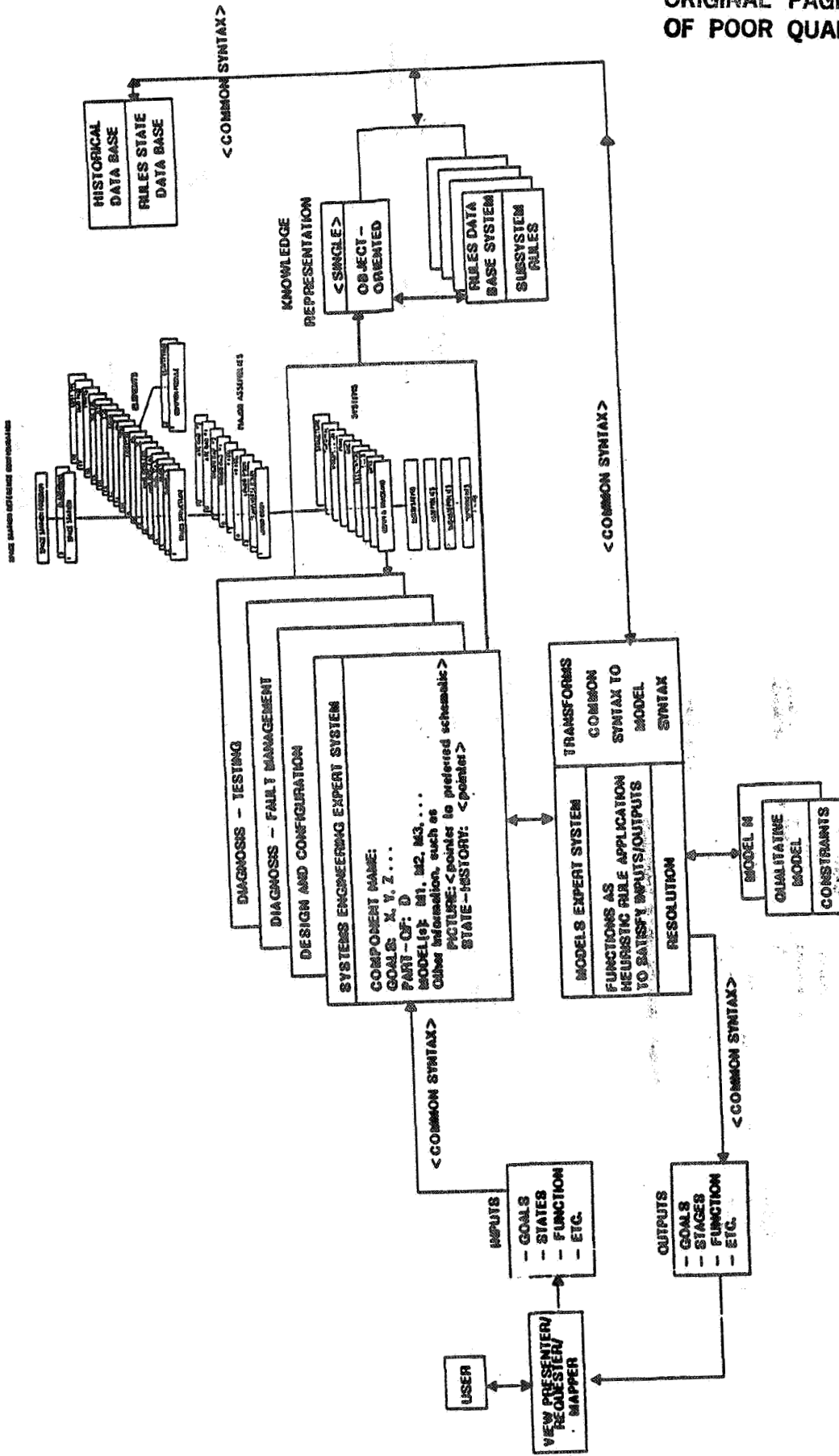
FIGURE 2   ABSTRACTED META-LEVEL TRANSFORMATION

FIGURE 4 PROTOTYPE EXPERT SYSTEMS OVERVIEW

Blackboard Architectures and Their
Relationship to Autonomous Space Systems

Allison Thornbrugh

Martin Marietta Denver Aerospace
P.O. Box 179
Denver, Colorado 80201

Abstract

The blackboard architecture provides a powerful paradigm for the autonomy expected in future spaceborne systems, especially SDI and Space Station. Autonomous systems will require skill in both the classic task of information analysis and the newer tasks of decision-making, planning and system control. Successful blackboard systems have been built to deal with each of these tasks separately --- data fusion in speech understanding [Erman81] and planning of errands with OPM [Hayes-Roth79]. The blackboard paradigm achieves success in difficult domains through its ability to integrate several uncertain sources of knowledge.

In addition to flexible behavior during autonomous operation, the system must also be capable of incrementally growing from semi- autonomy to full autonomy. The blackboard structure allows this development. This paper will discuss the blackboard's ability to handle error, its flexible execution, and variants of this paradigm as they apply to specific problems of the space environment.

Automated Planning for Autonomous Systems

In engineering increasingly autonomous systems, we must concentrate on choosing an appropriate operating paradigm. Proposed spaceborne systems such as Space Station and SDI require continuous reliable operation. The environment will vary over time. Plans must be proposed internally, during operation to dynamically adapt system operations and consistently meet mission objectives throughout the system lifetime. These requirements focus our interest on automated planning.
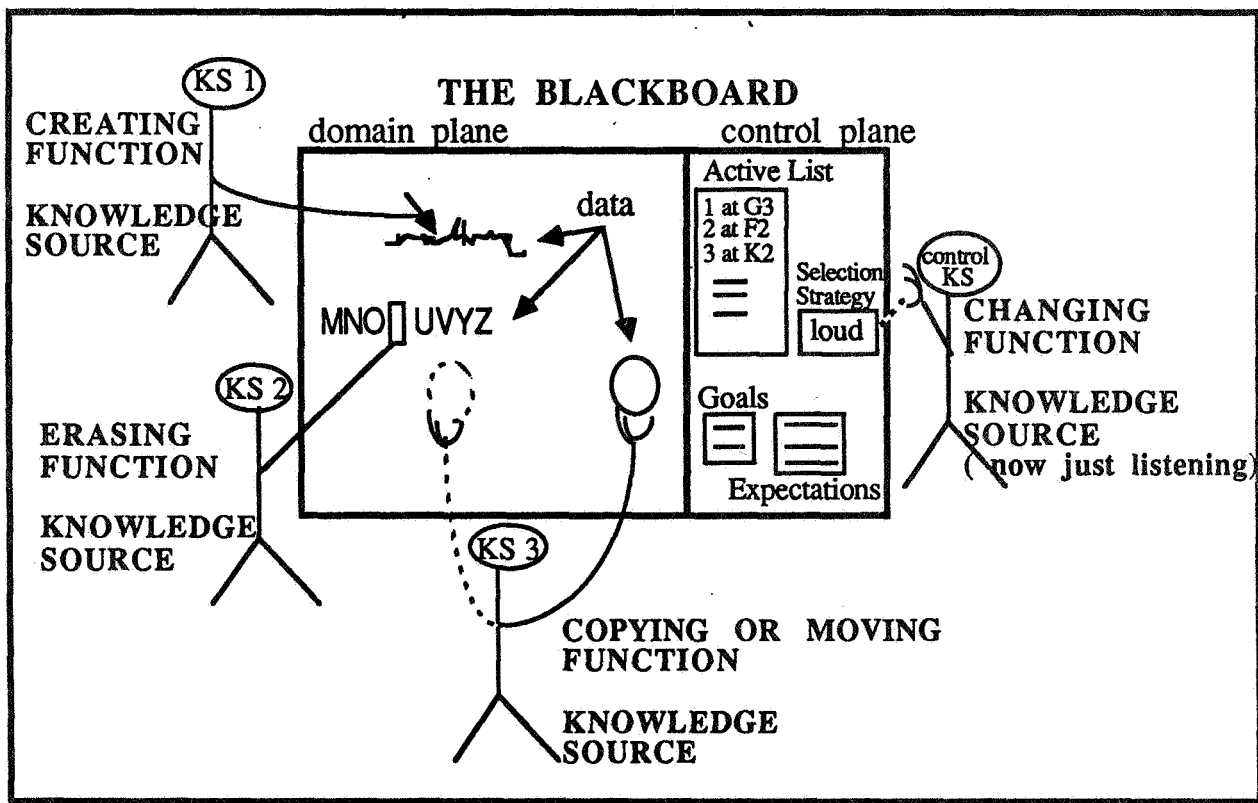
This paper extends "planners" to include time-dependent functionality such as prediction, resource allocation and context management. These factors affect the successful achievement of mission goals over the expected long lifetime of the autonomous system. Major concerns are efficient organization of multiple subsystems and adaptive management of finite resources. Planning to effectively manage these interrelated operations will be an important feature for an autonomous system.

System state determination for a large autonomous system such as Space Station or an SDI Battle Manager will be distributed among subsystems in an hierarchical manner. Effective management of such a scheme is critical to system operations.

At each functional level in the hierarchy, one system at the next level would be responsible for control of state determination for lower levels. Eventually, in this hierarchy, the ground becomes the next logical step to aggregate system state. If true autonomy is to be achieved, an intelligent replacement for these ground-based functions will be required. The planner paradigm is a helpful model for automating management, prediction and stabilization presently performed by man. In the following sections of this paper, a planner model will be presented in the abstract. A planning domain will be discussed separately. That domain is the top-level system executive for a complex autonomous space operation.

The Blackboard Planner

Planners based upon the blackboard architecture provide for adaptive control management and the ability to handle erroroneous data or sub-optimal analysis methods to solve difficult problems [Hayes-Roth79] [Erman81] [Nii86] . Active objects within the blackboard paradigm are called *knowledge sources* . *Knowledge sources* communicate through a global knowledge base, *the blackboard*. By posting, receiving and changing messages from this *blackboard* many possibilities are available to the system of executing *knowledge sources* (KS's). See Figure 1 for a simplified and general example of a blackboard system. A blackboard planner is not static. Any planner for an autonomous mission must be able to continuously adapt to the current context. Within the blackboard paradigm, plans can be "opportunistically" derived from current data input, current control strategy, or the current highest ranked goal(s).

Blackboard Architecture with Types of Knowledge Sources
Figure 1

A basic tenet of the blackboard architecture is modularity. *Knowledge sources* can be added, removed or replaced with ease because their only interaction with each other is specified completely and exactly through *the blackboard.* Specifications for autonomous planning systems may be ill-specified. The modular nature of a blackboard development environment is thus advantageous to develop gradually increasing expertise in these difficult areas. To investigate trade-offs during development, a blackboard planner allows one to include many approaches, encoded as multiple, cooperating *knowledge sources.* If this redundancy is available in the executable system, then the planner can vary coordinations as the runtime situation changes.

It is also possible to partition *the blackboard* into different levels of abstraction, multiple compartments, contexts, or time slices. In doing this, efficiencies specific to the particular planning domain can be achieved. In addition, partitioning the triggering area often allows multiple *knowledge sources* to execute in parallel. The blackboard architecture is an interrupt system rather than a polling system and thus performs better. Modular paradigms (e.g. rule-based systems) often do not show a significant increase in performance with an increase in the amount of knowledge. The blackboard does permit increasing performance through its emphasis on knowledge-based coordination.

Handling Uncertainty

An autonomous system will need to incorporate multiple sources of uncertain knowledge. The data may contain errors; some *knowledge sources*, for the sake of speed, may not be optimal. The blackboard architecture allows the effective combination of these uncertain sources into a competent decision-making tool.

Multiple control strategies (see Table 1) can create multiple lines of inference within blackboard planner operation. Messages on *the blackboard* are often termed hypotheses, or the hypothesis. At an intermediate planning stage there might still be several competing answers to the problem. Multiple lines of inference coordinate efforts in generating and eliminating different hypotheses to achieve a better final answer. For example, a predictive algorithm could be given boundaries to work within through previous reduction of a higher level goal. Or, multiple compartments performing similar computations could pool hypotheses in order to feedback improvements to their individual results.

Control Strategy	Possible Knowledge Source
Bottom-Up	Data Fusion
Top-Down	Goal-to-Subgoal
Island-Building	Nearest Neighbor

Table 1

Blackboard planners tolerate faults well and degrade gracefully. These traits are directly related to the property of modularity. Because the blackboard architecture is

modular, depth can be added to the planning. Deep reasoners are characterized by smaller, more general reasoning steps which allow a broader range of problems to be covered, usually at the cost of more steps and thus less speed. To avoid making a brittle planning system, depth can be specifically added as back-up to more sharply focused, shallower knowledge. Back-up through depth or through multiple shallow reasoning paths is a characteristic of blackboard planners and allows them to continue gracefully as variants of base test problems arise.

Autonomous System Executive Fault Protection

A major problem encountered in autonomous operation is management of state determination. Control is often distributed through many levels and perturbed by faults. The challenge is to plan appropriate fault recovery mechanisms under changing conditions in the environment and in the system itself.

The most strategic recovery configuration is that based on current context, current assumptions. The space of available contexts varies. Indeed more than one context may be viable at the same time. Multiple contexts map to multiple hypotheses on the *blackboard*. This array of choices of "the truth" can be maintained within the blackboard architecture to provide an adaptable framework for fault isolation and recovery planning.

Multiple coordinating methods of state determination are essential because of the critical nature of fault protection in autonomous systems. The blackboard planner's ability to coordinate multiple sources of knowledge provides the environment to develop and execute these methods of state determination. Truth maintenance provides one model for state determination [Doyle80] [deKleer86]. Relationships between a context of assumptions and a hypothesis would be built into *the blackboard* and modules written to maintain these contexts. For example, truth maintenance *knowledge sources might* update dependencies between data or choose a current context. This blackboard planner restructuring would,at the very least, permit dynamic system resiliency based upon recovery mechanisms which have been assigned to different contexts.

The blackboard paradigm permits further enhancements. For example, the context-based recovery mechanisms could be aided by some sort of output data normality testing. Whether any subsystem is performing according to expectations could be, for instance, answered by pre-assigned bounds , by a system of alert statuses incorporated into the contexts, or perhaps with a "suspicion" *knowledge source* that reviews strange behavior in hopes of isolating otherwise undetectable faults.

Some Example Fault Protection KS's

1. Alert Status Change
2. Functional Output Testing
3. Normality Audit
4. Suspicious Behavior Audit

Table 2

Knowledge sources, such as those in Table 2, combined with a context or truth maintenance model, will provide system executive fault protection software that dynamically changes to meet new situations.

Evolution to Full Autonomy

Functions like determining suspicious behavior are ill-specified for automation. Many human functions will need to remain in man's hands while automation of other, more concrete modules proceeds. The blackboard planner's modularity allows autonomy to develop during a period of adjustment. The "suspicion" *knowledge source* for example can be implemented as a Man Machine Interface (MMI) in the beginning, allowing developers to experiment with and record their reactions to different situations and actions before encoding these internally. This type of human placeholder is called "man-as-a-knowledge-source" because, as an active object within the blackboard planner, the interface would be a *knowledge source* .

The evolution of placeholders to decision-making *knowledge sources* can only take place if the planner is explainable. An explanation facility is easily encompassed within the blackboard paradigm by storing a history of actions performed. Because control features such as KS choice strategy can be explictly stored on *the blackboard* [Hayes-Roth84,85], the system can not only explain what it did, but also why. With the simultaneous existence of such an explainable prototype planner and a compiled executable planning system, perhaps in space, development proceeds smoothly.

Current Research Issues

Current research issues center on the viabilitiy of moving blackboard planners from the research laboratory into the real world. There, theoretical performance must be demonstrated on actual hardware and embedded within systems software. Increase in speed through parallel implementation is one key issue. Changes to existing blackboard systems' foundations in order to achieve this parallelism may be a second issue. The blackboard paradigm is inherently parallel. This is clearer if one sees that the software model is based upon the physical blackboard and its usual uses. Most implementations to date have implemented serial execution; control has thus been an attempt to find the one best KS to execute next. Although exceptions exist, [Erickson85] [Stentz85], there is still a lack of evaluations of multiple parallel architectures and their accompanying systems software trying to find the best fit to the blackboard architecture. This is an area Martin Marietta is currently investigating, supported in part by USAF contract #F30602-86-C-0062 through Rome Air Development Center.

Finding the appropriate building blocks for a parallel implementation is another underdeveloped issue. Commonly, blackboard systems are built on top of a frame system [Erickson85] [Nii82] [Nii81]. In doing so, blackboard hypothesis organization is greatly simplified because means for partitioning and typing data are already in existence. Research to date supports two reasonable approaches toward achieving parallel implementation:

1.   Continue with the current frame system basis and use a shared-memory multiprocessor. Both a global *blackboard* and a frame inheritance hierarchy are shared memory structures. This approach emphasizes memory contention problems.

2.   Adapt an object-oriented programming system or frame system especially for blackboards, where every object contains a mini blackboard planner executive. This would enable fine grained distribution of tasks and of memory contention resolution. *Blackboard* data objects would then be "intelligent" in the sense of intelligent terminals. The main problem presented by this approach is communications contentions.

Systems engineering of these approaches will require maximizing one of two parameters, either memory access organizational efficiency or connection protocols & topologies. In any specific application, islands of knowledge interaction and shared knowledge will develop which may emphasize one of these approaches above the other.

<u>Summary</u>

In conclusion, the blackboard planner paradigm seems suited to the planning problems encountered in developing autonomous spaceborne systems and should be developed further. The example planning problem of fault recovery and state determination within the multi-level system executive showed the complexity of planning for autonomy. A blackboard planner manages these complexities by combining many sources of uncertain knowledge and permitting especially difficult problems to be explored through human interaction with the planner. Explicitly describing the planning environment as changeable, e.g. a truth maintenance model, avoids falsely modeling assumptions as static for planning components of a complex autonomous system operating in a time-varying space environment. Planning must be dynamic. A static scheduling approach, even if optimal at design, will be too brittle for an autonomous application.

The blackboard architecture planner system described in this paper deserves futher investigation. Research to date has interesting implications in other areas of computer science which need exploration. Some possible future projects are:

1. Compilers for creating efficient executable code
2. Hardware architecture designs perhaps based upon the idea of active data objects
3. MMI designs based upon the blackboard architecture.

Details such as the integration of truth maintenance ideas with the blackboard framework and quantified control of feedback between separate areas of planning knowledge are both topics for further study.

References

deKleer, J., (1986) "An Assumption-based TMS," *Journal of Artificial Intelligence,* 28: 127-162.

Doyle, J., (1979) "A Truth Maintenance System," *Journal of Artificial Intelligence,* 12: 231-272.

Erickson, W.K., (1985) "The Blackboard Model: A Framework for Integrating Cooperating Expert Systems," *AIAA/ACM/NASA/IEEE Computers in Aerospace V Conference,* Long Beach, CA, Oct. 21-23, 1985.

Erman, D. L., F. Hayes-Roth, V.R. Lesser & D. Raj. Reddy, (1981) "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Readings in Artificial Intelligence,* eds B. Webber & N. Nilsson, Tioga Publishing, Palo Alto, CA, pp 349-389.

Hayes-Roth, B., F. Hayes-Roth, Stan Rosenschein & Stephanie Cammarata, (1979) "Modeling Planning as an Incremental, Opportunistic Process," *IJCAI-79,* vol 1, pp. 375-383.

Hayes-Roth, B., (1984) "BB1: An Architecture for Blackboard Systems That Control, Explain, and Learn About Their Own Behavior," HPP Report 84-16, Stanford University.

Hayes-Roth, B., (1985) "Blackboard Architecture for Control," *Journal of Artificial Intelligence,* 26: 251-321.

Nii, H. Penny, N. Aiello, C. Bock & W.C. White, (1981) "Joy of AGE-ing: An Introduction to the AGE-1 System", HPP Report 81-23, Stanford University.

Nii, H. Penny, E. Feigenbaum, J. Anton & A.J. Rockmore, (1982) "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *The AI Magazine,* Spring 1982, pp 23-35.

Nii, H. Penny, (1986) "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *The AI Magazine,* Summer 1986, pp 38-53.

Stentz, A. & S. Shafer, (1985) "Module Programmer's Guide to Local Map Builder for ALVan," Technical Report Carnegie-Mellon University, Computer Science Department, August.

"Space Station Automation Study: Autonomous Systems and Assembly, Final Report," MMDA, November 1984.

DESIGN CONSIDERATION IN CONSTRUCTING HIGH PERFORMANCE EMBEDDED KNOWLEDGE-BASED SYSTEMS (KBS)

Authors: Shelly D. Dalton
          Philip C. Daley

Martin Marietta Denver Aerospace
Denver, Colorado

ABSTRACT

As the hardware trends for artificial intelligence (AI) involve more and more complexity, the process of optimizing the computer system design for a particular problem will also increase in complexity. Space applications of knowledge-based systems (KBS) systems will often require an ability to perform both numerically intensive vector computations and real-time symbolic computations. Although parallel machines can theoretically achieve the speeds necessary for most of these problems, if the application itself is not highly parallel, the machine's power cannot be utilized. A scheme is presented here which will provide the computer systems engineer with a tool for analyzing machines with various configurations of array, symbolic, scaler, and multi-processors. High speed networks and interconnections make customized, distributed, intelligent systems feasible for the application of AI in space. The method presented in this paper can be used to optimize such AI system configurations and to make comparisons between existing computer systems. It is an open question whether or not, for a given mission requirement, a suitable computer system design can be constructed for any amount of money. Additionally, significant cost and performance risk can occur which will be avoided by careful adherence to guidelines presented in this paper. This work is supported, in part, by Air Force contract F30602-86-C-0062 from the Rome Air Development Center.

INTRODUCTION

Historically, computer systems which were used for space applications focused primarily on support of the flight package functions: attitude and control, guidance and navigation, thermal control and power control. Scientific processing for experimental data collection and manipulation, was mainly limited to ground located systems which need not be real-time. Space Station and Strategic Defense Initiative (SDI) are proposing systems which will not only require enormous amounts of space-based, real-time scientific processing, but will also incorporate knowledge-based systems (KBS) into the embedded system. The science package for Space Station is expected to be 3-4 times as large as the flight package; while SDI applications such as surveillance, acquisition,

PRECEDING PAGE BLANK NOT FILMED

tracking and kill assessment (SATKA), battle management and weapon control/fire control will require a science package 10-100 times as large as the flight package. An involved design approach must be taken if these systems are to meet both performance and KBS requirements.

FUTURE SYSTEMS FOR SPACE

Although software and hardware environments have become richer and more efficient, integration of these environments must be carefully engineered. For Space Station and SDI, a heterogeneous mix of serial, symbolic (KBS) and vector instructions will be required. For example, processing of external commands could use conventional serial instructions, the tracking and pointing of an antenna array could use vector instructions, and fault recovery could be implemented in an expert system using symbolic manipulation. For optimum processing, each of these instructions sets could be housed in a separate processor (or multi-processor) specialized for that type of instruction, linked in either a tightly or loosely coupled heterogeneous system. While, conceptually, a distributed system is easy to manage, many difficulties exist in meeting the system software and the communications requirements. Analyses of various configurations also becomes increasingly difficult as the size and complexity of the system grows.

Due to the relative simplicity of the science package in the past, the design of on-board data management system was straight-forward. One-CPU serial processors were used which were rated less than 1-MIP. For example, the ATAC-16MS computer used by NASA's Magellan and Galileo is rated at 0.5 MIPS [ATAC, 79]. Most programs consisted of a few thousand lines of fixed-point instructions running at 0.001-0.1 MIPS. Design considerations were limited to trading-off rated speeds of space-qualified processors. For ground systems, high performance computers could be evaluated by performing analysis on these homogeneous systems [Hockney, 84], [Mohan, 84] and benchmarking.

ANALYSIS OF SYSTEM REQUIREMENTS

Today, the varying types of instructions required in the system (serial, symbolic and vector) must be analyzed with respect to quantity, structure and complexity [Cook, 84]. Also, various configurations of heterogeneous processors (pipelined, symbolic, array and multi-processors) with their communications schemes must be analyzed for data flow and effective speeds. Even at the highest level of analysis, MIPS-ratings (million instructions per second) for serial instructions cannot be compared with FLOPS-ratings (floating point operations per second) for floating point operations on another machine and LIPS-ratings (logical inferences per sendon) for symbolic (Prolog) operations on a third. Any rated speed must also be scruntinized with respect to the sustained speed which the machine can deliver when operating system services are also being performed in conjunction with the application. For most machines, actual performance is approximately one-tenth rated speed [Dongarra,

85]. For array and multi-processor systems, actual performance will be even smaller due to time needed for overhead and synchronization of tasks. Systems design is now far from simple.

AMDAHL'S LAW

Machines expoiting parallelism are also constrained by a principle well known among computer architects—the so-called "Amdahl's Law". Amdahl's Law states that the overall speed-up which can be obtained in a parallel system is inversely proportional to the fraction of the application which must be performed serially [Kibler, 85]. For example, if 10% of the code is serial, then the maximum speed increase over a single processor machine is 10 times, regardless of the number of processors in the parallel machine. This is antagonistic to the commonly held belief that any performance level can be reached if enough parallelism is used in the machinery. Every application has an inherent limitation on speed-up, so the degree the application is susceptible to parallelism must be understood before the hardware is choosen.

IMPACT OF KBS

While previous space-based systems have had well defined requirements which fit easily into available hardware, current requirements for space systems which will include KBS are inherently "soft" and tend to tax the performance of available hardware. The "softness" of the requirements is due, in part, to the difficulty in expressing, a priority, the compute requirements for functions normally performed by humans and their interactions with large ground systems. These functions are now proposed to be achieved via KBS running in space. Also, KBS has in the past rarely been embedded, even in ground systems. The wealth of management guidelines, specification guidelines, and design considerations which are well known for conventional embedded systems have not yet been modified to incorporate KBS [Daley, 85].

Unfortunately, the modifications in systems engineering which must be made for future space systems go well beyond those necessary for KBS. Fault tolerance and trustedness cannot just be added as an applique to design specifications as was done in the past. The requirements for the applications alone may approach the limits of hardware technology. Additional systems services which are not integrated with the application could easily reduce performance to an unexceptable level. Therefore, fault tolerance, KBS, parallel processing, and trustedness must be integrated from design conception in order to develop requirements with some degree of confidence and achievability.

TYPES OF ARCHITECTURES

Today's system engineering must involve a deep understanding of the type of architecture available for high performance computers. Fault tolerance can be integrated well with array and multi-processors. Pipelined architectures may be needed for image

593

computations and stack architectures for lisp processing. The luxury of using an architecture which is tailored to the application may well be achievable for distributed systems.

Array processors, such as the GAPP built by Martin Marietta, expoit fine grain parallelism for applications such as signal and image processing, and computations involving vectors or matrices which are large compared to the number of processing elements. Each processing element of the array is fairly simple, being controlled synchronously by a central control unit and/or a front-end processor. These processors are not efficient for problems in KBS which need asynchronous operations that are not highly structured.

Multiprocessors, such as the Butterfly built by BBN, can run asynchronously and have attracted a great deal of attention from the AI community. The processing elements can vary in power from arithmetic logic units (ALU) to small VAX's. Configurations are available which specialize in areas such as semantic networks or floating point operations, using local and/or global memories. Unfortunately, software development is very difficult on most of these machines, and it is difficult to distribute tasks to effectively use the hardware. Care must be taken to find an architecture which complements the task structure of the application, not one which merely provides enormous computing power.

Originally, pipelined computers (such as CRAY's) where used to achieve high performance. While these computers can still out-perform most multi-processor configurations, maintenance and environmental supports are often too complex for space. Their pipelines divide computations such as floating point operations into stages. Elements of vectors follow each other through the pipeline. The pipeline is less efficient when it is being filled or emptied than when it achieves steady-state of a full pipeline. When at steady-state, maximum speed-up is proportional to the number of pipeline stages. Since KBS applications do not use large vectors, pipelined processors do not provide enormous speed increases, but instruction pre-fetch pipelines can be utilized by almost any type of application.

The type of architecture which has been most effective for KBS is stack-oriented architecture. Stacks are used to hold lists and results of function calls in stack buffers which do not require explicit addresses. Since addresses are not as important in AI as in conventional languages such as Fortran, associative array processors and content addressable memory are also currently being researched as environments for KBS. Since software cost currently exceeds hardware costs, the ease of programming still makes stack-oriented architectures the most viable choice for many KBS applications.

DATA MANAGEMENT

The emphasis on speed for high performance architectures tends to overshadow the need for efficient data management by the

operating system. Immense processing power is useless if it must wait on the operating system and the I/O channels.

Data management will vary for the different architectures. For KBS, type checking and garbage collection functions must be provided [Hirsh, 84]. For processors which share memory, the data integrity problem is even greater than what is involved with virtual meory, and can quickly become intractible. Not even virtual memory management is simple, since most common approaches are based on disks which cannot be easily used in space systems. Memory problems with space systems cannot be easily fixed by adding, a posteriori, more memory or operating system software. As with the application software, the complexity of the operating system software will be high, so small deviations in requirements could cause huge reductions in performance.

From a hardware point of view, a distributed system which provides parallel machines for vector and matrix operations, stack machines for KBS and conventional machines for serial processing is becoming fairly simple. From a software point of view, operating systems do not yet exist which can distribute and control a mix of instructions over a heterogeneous hardware environment. Until such an operating system is developed, the system must be loosely coupled with the developer constantly aware of the internal interface capabilities.

CONCLUSION

Hardware is becoming available to provide for enormous increases in performance in space for the future. Likewise, software techniques such as KBS are providing means to drastically increase functionality for space systems. If systems engineering does not rise to the challenge of providing means to effectively design and control these new, complex systems, then future space systems may face additional programmatic risks or degradation of performance.

REFERENCES

[ATAC 79]   ATAC-16MS Principles of Operation, Applied Technology Coporation, June 1979

[Cook 83]   Cook, Stephen A., "An Overview of Computational Complexity," Communications of the ACM, 26 (1983) P 401-408

[Daley 85]   Daley, Phil C., "Knowledge-Based Systems: Systems Engineering and Management Issues," Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, 1985

[Dongarra 85]   Dongarra, J. J., "Performance of Various Computers Using Standard Linear Equations Software in a Fortran Environment," Mathematics and Computer Science Division, Argonne National Laborator, Oct. 11, 1985

[Hirsh 84]   Hirsh, Abraham, "Tagged Architecture Supports Symbolic Processing," Computer Design, June 1, 1984

[Hockney 84]   Hockney, R. W., "Performance of Parallel Computers, High Speed Computation, J. S. Kowalik, ed., 1984

[Kibler 85]   Kibler, Dennis F. and Conery, John, "Parallelism in AI Programs", IICAI, 1985

[Mohan 84]   Mohan Joseph, "Performance of Parallel Programs: Model and Analysis," PhD. Dissertation, Carnegie-Mellon University, 1983

Validation of Expert Systems

Rolf A Stachowitz and Jacqueline B Combs

Lockheed Missiles & Space Company, Inc.

Software Technology Center, O/96-01, B/30E

2124 E St. Elmo Rd.

Austin, Texas, 78744

Abstract[1]

The validation of expert systems (ES's) has only recently become an active AI research topic. Current approaches have concentrated mainly on the validation of rule properties (basically syntactic) of such systems. Our effort improves on current methods by also exploiting the structural and semantic information of such systems.

To increase programmer productivity, more and more companies have begun exploiting the advent of AI technology by developing applications using ES shells or other AI-based high-level program generators.

Whereas papers and books on validating traditional software abound, the validation of ES's has received considerably less attention in the AI literature. Rare exceptions are Nguyen et al. [3] and Suwa et al. [4].

This lack of attention is also reflected in the "standard" AI terminology where *Validation of ES's* is used synonymously with *Evaluation of the Quality (or Performance) of ES's* rather than with *VV&T* (see, for example, the index in Buchanan/Shortliffe [1]). Most existing, commercially available ES shells correspondingly provide, if at all, only rudimentary support for validating ES's. Thus **KEE** (Knowledge Engineering Environment, IntelliCorp) mainly supports *legal values* and *legal numeric ranges*; the sophisticated **ART** (Automated Reasoning Tool, Inference Corporation), so far, does not provide any validation tools beyond straight-forward syntax checking.[2]

A notable exception is **LES**, (Lockheed Expert System). Like **KEE**, it supports *legal values* and *legal numeric ranges*. It also provides a program, *Check*, which detects potential errors in ES rules, such as *dead-end clauses*, *unreachable clauses*, *irrelevant clauses*, *cycles* in rules and certain cases of *inconsistency* and *incompleteness* (Perkins et al [2]).

In our opinion even the LES effort suffers from the fact that only very little use is made of semantic information or metaknowledge in validating ES's. Beginning in 1986, the AI group of the Lockheed Software Technology Center (LSTC) at Austin has started work on **EVA** (Expert Systems Validation Associate) which makes use of metaknowledge to validate ES's.

In the remainder of this paper, we outline the architecture, functionality, and future goals of **EVA** and describe the features that have been implemented *for* **ART** and, partially, *in* **ART**, the ES shell used at LSTC.

PRECEDING PAGE BLANK NOT FILMED

References

1. B.G. Buchanan, E.H. Shortliffe (Ed.). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Company, Reading, MA, .

2. *LES User's Manual.* Lockheed Missiles & Space Company, Inc., 1986.

3. T.A. Nguyen, W.A. Perkins, T.J. Laffey, D. Pecora. An Expert Systems Knowledge Base for Consistency and Completeness. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 1985, pp. 375-378.

4. M. Suwa, A.C. Scott, E.H. Shortliffe. "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System". (1982), 16-21.

A NONLINEAR FILTERING PROCESS DIAGNOSTIC SYSTEM FOR THE SPACE STATION

Raymond R. Yoel

Boeing Aerospace Company
Space Station Program
Huntsville, Alabama 35807

Dr. M. Buchner
Dr. K. Loparo
Arif Cubukcu

Case Western Reserve University
Systems Engineering Department
Cleveland, Ohio 44106

ABSTRACT

This paper will present a nonlinear filtering process
diagnostic system, terrestrial simulation and real-time
implementation studies, and discuss possible applications
to Space Station subsystem elements.

There is the need for computer based process diagnostic systems on
the Space Station. Certain processes within Space Station
subsytems, e.g., Thermal Control System (TCS) and Process Material
Management System (PMMS), can not be continuously monitored by the
astronauts. Further, there are numerous failure modes where the
time necessary for an astronaut to detect and isolate a failure is
greater than the time for those failures to result in serious,
even disastrous conditions. Throughout the terrestrial chemical,
oil, utility, and other process industries, computer based process
diagnostic systems are being implemented at an increasing rate.
The field of process diagnostics has provided many techniques,
with each having distinct advantages and disadvantages.

The objectives for implementing a computer based process
diagnostics system are dedicated monitoring and quick detection
times. Further, the system can not give failure mode response
under normal conditions, i.e., false alarms. Once the reliability
of the system is questioned, lack of confidence can result in slow
or incorrect action by astronauts or automatic control systems.

At Case Western Reserve University, a process diagnostic system
using model based nonlinear filtering for systems with random
structure has been shown to provide improvements in stability,
robustness, and overall performance in comparison to linear filter
based systems. A sub-optimal version of the nonlinear filter
(zero-order approximation filter, or ZOA filter, similar to the
multiple model filter introduced by Magill) was used in simulation
studies, initially, with a pressurized water reactor model and
then with water/steam heat exchanger models. Finally, a real-time
implementation for leak detection in a water/steam heat exchanger
was conducted using the ZOA filter and heat exchanger models.

This nonlinear filtering process diagnostic system can be applied
to Space Station subsystem elements which have process parameters
with random structure. Some possible applications are:

    TCS: heat exchangers, valves, sensors, piping
    PMMS: separators, storage tanks, valves, sensors, piping

DESCRIPTION

The integration and implementation of an artificial intelligence system
that controls two robots, a vision system, tactile sensors, and a
supermini process control computer in R & D robotic applications is
discussed. The system uses an Expert System developed in a Common-Lisp
environment which can solve a variety of problems using a rule based
system. This 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.
The expert system resides within a real-time process control system.

ABSTRACT

REAL TIME AI EXPERT SYSTEM FOR ROBOTIC APPLICATIONS

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 are 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 effect, the system
can learn new rules. The expert system is interfaced through touch
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 information required to direct the robots to complete the desired
task. The presentation discusses the mechanisms that the expert system
uses to solves problems (goals), the different rule data bases, 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.

John F. Follin, Staff Engineer, Robotics and Heuristic Process Control

GA TECHNOLOGIES
PO BOX 85608
SAN DIEGO, CA 92138
(619) 455-4405

COMPUTER USED :   MV/4000 with disk, tape, line printer, and operator
                   consoles

EQUIPMENT INTERFACED THROUGH COMPUTER :

                   GCA/Par overhead robot
                   Prab cylindrical robot
                   Machine vision Genesis 2000 CCD vision system
                   CAMAC data acquisition systems (digital and analog)
                   Speech synthesizer
                   Voice Recognition
                   Megatek Graphics Display Station
                   Graphic terminals fitted with touch sensors
                   Joystick controller unit for robot control
                   Tactile touch sensors for GCA/Par robot

MATHEMATICAL ALGORITHMS FOR APPROXIMATE REASONING

John H. Murphy, Seung C. Chay and Mary M. Downs
Westinghouse R&D Center
Pittsburgh, PA 15235

ABSTRACT

Most state-of-the-art expert system environments contain a single and often ad hoc strategy for approximate reasoning. Some environments provide facilities to program the approximate reasoning algorithms. However, the next generation of expert systems should have an environment which contains a choice of several mathematical algorithms for approximate reasoning. To meet the need for validatable and verifiable coding, the expert system environment must no longer depend upon ad hoc reasoning techniques but instead must include mathematically rigorous techniques for approximate reasoning. In this paper, we review popular approximate reasoning techniques including: certainty factors, belief measures, Bayesian probabilities, fuzzy logic, and Shafer-Dempster techniques for reasoning. Then we concentrate on a group of mathematically rigorous algorithms for approximate reasoning that could form the basis of a next generation expert system environment. These algorithms are based upon the axioms of set theory and probability theory. To separate these algorithms for approximate reasoning, various conditions of mutual exclusivity and independence are imposed upon the assertions. Approximate reasoning algorithms presented include: reasoning with statistically independent assertions, reasoning with mutually exclusive assertions, reasoning

with assertions that exhibit minimum overlap within the state space, reasoning with assertions that exhibit maximum overlap within the state space (i.e. fuzzy logic), pessimistic reasoning (i.e. worst case analysis), optimistic reasoning (i.e. best case analysis), and reasoning with assertions with absolutely no knowledge of the possible dependency among the assertions. To further separate these approximate reasoning algorithms, a robust environment for expert system construction should include the two modes of inference: modus ponens and modus tollens. Modus ponens inference is based upon reasoning towards the conclusion in a statement of logical implication, whereas modus tollens inference is based upon reasoning away from the conclusion. These algorithms, when consistently applied, allow one to reason accurately with uncertain data. The above environment can also replicate most state-of-the-art expert system environments which provides a continuity between the current expert systems which cannot be validated nor verified and future expert systems which should be both validated and verified.

ABSTRACT

REAL-TIME SPACE SYSTEM CONTROL
WITH EXPERT SYSTEMS

David Leinweber, Ph.D
Lowell Hawkinson
LISP Machine Inc.
Los Angeles, California   90045

John Perry, Ph.D.
OAO Corporation
Los Angeles, California   90245

Many aspects of space system operations involve continuous
control of real-time processes. These processes include
electrical power system monitoring, pre-launch and ongoing
propulsion system health and maintenance, environmental and
life support systems, space suit checkout, on-board
manufacturing, and vehicle servicing including satellites,
shuttles, orbital manuvering vehicles, orbital transfer
vehicles and remote teleoperators. Traditionally, monitoring
of these critical real-time processes has been done by trained
human experts monitoring telemetry data. However, the long
duration of future space missions and the high cost of crew
time in space creates a powerful economic incentive for the
development of highly autonomous knowledge-based expert control
procedures for these space systems.

In addition to controlling the normal operations of these processes, the expert systems must also be able to quickly respond to anomalous events, determine their cause and initiate corrective actions in a safe and timely manner. This must be accomplished without excessive diversion of system resources from ongoing control activities. Any events beyond the scope of the expert control and diagnosis functions must be recognized and brought to the attention of human operators. Real-time sensor-based expert systems (as opposed to off-line, consulting or planning systems receiving data via the keyboard) pose particular problems associated with sensor failures, sensor degradation and data consistency, which must be explicitly handled in an efficient manner. A set of these systems must also be able to work together in a cooperative manner.

This paper describes the requirements for real-time expert systems in space station control, and presents prototype implementations of space system expert control procedures in PICON (process intelligent control) for real world examples. PICON is a real-time expert system shell which operates in parallel with distributed data acquisition systems. It incorporates a specialized inference engine with a specialized scheduling portion specifically designed to match the allocation of system resources with the operational

requirements of real-time control systems. Innovative knowledge engineering techniques used in PICON to facilitate the development of real-time sensor-based expert systems which use the special features of the inference engine are illustrated in the prototype examples.

The paper concludes with a discussion of new facilities being incorporated into the PICON system, including automatic rule generation and diagnostic capabilities based solely on descriptive frames.

A FLEXIBLE SEARCH STRATEGY FOR PRODUCTION SYSTEMS

Pradip Dey
S. Srinivasan
K. R. Sundararaghavan
University of Alabama at Birmingham, AL 35294

Most problems considered to be solvable by expert systems have very large search space. It is, therefore, imperative to use efficient search strategy in expert system tools. Thus, OPS5 uses a kind of hill-climbing which is very efficient. However, hill-climbing is inadequate for many problems because it is one of the least dependable search strategies. It fails in ridges, and local maximum. In order to make the search efficient and adequate one can (1) adopt best-first search instead of hill-climbing or (2) modify hill-climbing with intelligent backtracking. There are some serious problems in application of best-first in expert systems. Therefore, the second alternative is adopted. It is implemented in a production system called PRO2 embedded in C running on UNIX. We call the search hill-tracking. It has the advantage that it reduces search space by using the hill-climbing function and avoids the deficiencies of hill-climbing by recovering from ridges, and local maximum by intelligent backtracking.

PRO2 is a general purpose tool for developing expert systems. This is a rule based production system with an effective, intelligent and flexible backtracking control mechanism, which makes the system more dependable. In case the goal state is not reached by following a path the system backtracks to an earlier point and tries alternative paths. But ordinary cronological backtracking is grossly inefficient. PRO2 has two features which allow efficient backtracking: (1) In each recognize-act cycle, at most three rules are selected by conflict resolution strategies; the best one is fired and the other two are held in an ordered set as points of backtracking. This feature reduces the number of alternatives to be tried in the event of a backtracking. (2) The system backtracks only after it fails to provide a suitable solution without backtracking. Thus, if PRO2 reaches a dead-end then it will backtrack to an earlier point and try one of the untried rules saved in the conflict resolution phase. The system also backtracks if the user is not satisfied with the solution and requests for alternatives. Thus, this system provides a greater opportunity to find an acceptable solution if the user is not satisfied with the earlier solution provided. Most production systems are either inefficient or inadequate to search alternative paths. In space applications where safety is of prime importance PRO2 will have a competitive edge over other tools, because it is efficient, flexible, and adequate for building highly dependable expert systems.

PRECEDING PAGE BLANK NOT FILMED

Semantic Based Man-Machine Interface for Real-Time Communication

M. Ali and C.-S. Ai
Knowledge Engineering Laboratory
The University of Tennessee Space Institute
Tullahoma, Tennessee 37388

ABSTRACT

A flight expert system (FLES) has been developed to assist pilots in monitoring, diagnosing and recovering from in-flight faults. To provide a communications interface between the flight crew and FLES, a natural language interface (NALI) has been implemented. Input to NALI is processed by three processors: 1) the semantic parser. 2) the knowledge retriever, and 3) the response generator. First, the semantic parser extracts meaningful words and phrases to generate an internal representation of the query. At this point, the semantic parser has the ability to map different input forms related to the same concept into the same internal representation. Then the knowledge retriever analyzes and stores the context of the query to aid in resolving ellipses and pronoun references. At the end of this process. a sequence of retrieval functions is created as a first step in generating the proper response. Finally, the response generator generates the natural language response to the query.

FLES's knowledge-base consists of temporal as well as non-temporal knowledge. The temporal knowledge is mainly concerned with the order and timing of events. Component failures. sensor failures and abnormal situations are a few examples of events. The non-temporal knowledge is concerned with the structural and diagnostic aspects of the flight domain. The architecture of NALI has been designed to process both the temporal and non-temporal queries. Provisions have also been made to reduce the number of system modifications required for adapting NALI to other domains. This paper describes the architecture and implementation of NALI.

PRECEDING PAGE BLANK NOT FILMED

The Concurrent Common Lisp Development Environment

1 Summary

A discussion of the Concurrent Common Lisp Development Environment on the iNTEL Personal Super Computer (iPSC) is presented. The advent of AI based engineering design tools has lead to a need for increased performance of computational facilities which support those tools. Gold Hill has approached this problem by directing its efforts to the creation of a concurrent, distributed AI development environment. This discussion will focus on the development tools aspect of the CCLISP environment. The future direction of Gold Hill in the area of distributed AI support environments is also presented.

2 Outline of Talk

1. AI techniques are providing a basis for current generation Engineering design tools

2. As with other AI applications, these tools are being limited by current generation computational facilities

3. Gold Hill is removing those limitations by developing a concurrent development environment which allows increased performance for the standard AI tools.

4. The original vision:
   - Bring modern AI development tools to the market on generic micro-processor based systems.

   - Result: Golden Common Lisp Developer on 80286 based machines.

   - Facilitate Distributed AI support environments as an extension to the stand-alone environment

   - Result: GCLISP-Network and CCLISP

   - Provide Access to multiple, loosely coupled nodes via message passing semantics.

5. The search for the appropriate vehicle led to iNTEL Corp, and the iPSC.

6. Under a joint development agreement, iNTEL and Gold Hill are bringing the first of the new generation, concurrent AI tools environments to market.

7. Product development is well underway, with the current alpha version being demonstrated at this conference.

8. Common Lisp is the basic development tool.

9. Extended to support messages passing via streams which are used to communicate with other processors in the iPSC.

PRECEDING PAGE BLANK NOT FILMED

10. The familiar development tools, including compilers, steppers and computation analysis facilities are extended to allow control of sets computations within the iPSC.

11. Access to the CCLISP environment from external Lisp based workstations. (Symbolics, TI Explorer, and IBM-AT's)

12. A short example of application in CCLISP

13. The future:
    a. Providing a uniform application interface to support the distribution of computations to external machines

    b. based upon open systems/actor technologies

    c. Provide distributed AI knowledge bases and reasoning tools.

    d. Insure that environment available in future concurrent architectures is also available in networks of generic workstations.

AUTOMATED SOFTWARE DEVELOPMENT WORKSTATION

Dale A. Prouty
Philip Klahr
Inference Corporation
Los Angeles, CA. 90045

ABSTRACT

A workstation is being developed that provides
a computational environment for all NASA engineers[1]
across application boundries, which automates
reuse of existing NASA software and designs, and
efficiently and effectively allows new programs/
designs to be developed, catalogued, and reused.
The generic workstation is made "domain specific"
by specialization of the user interface, capturing
engineering design expertise for the domain, and
by constructing/using a library of pertinent
information. The incorporation of software
reusability principles and expert system technology
into this workstation provide the obvious benefits
of increased productivity, improved software use
and design reliability, and enhanced engineering
quality by bringing engineering to higher levels
of abstraction based on a well tested and class-
ified library.
Keywords: software workstation, software reuse,
          design reuse

INTRODUCTION

The workstation applies artificial intelligent (AI)
automated reasoning technology to the problems
associated with efficient construction of pro-
cedural software. The designs and procedural
software developed are catalogued or classified
for reuse in later design efforts, so that the
two main goals of the workstation, software reuse
and engineering design automation, are achieved.
At this writing, the project is underway but not
yet completed. The completion of the project is
approximately two years hence, and follows six
months of initial work. Thus this is a report
on the progress, approach, and conceptual
principles being applied in the early phases of
effort.
Software reuse is not a new theme. Much research
effort has gone into understanding the problems
and appropriate approaches for software reuse[1-3]
The feasibility of this workstation concept is
enhanced by incorporating the fruits of that
research. Two noteworthy conceptual results are
that software should be "reused" at the highest
level of abstraction possible and that addressing
narrow problem domains lends the greatest potential
for success.

This workstation incorporates these results first,
by providing a domain-specific "requirements
language" so that workstation interactions may
proceed from engineering requirements, and second
through supporting customizations of a general
purpose shell for specific problem domains.

The highest possible level of engineering software
design automation is one where requirements are
input and executable code produced with only an
essential amount of user interaction to assure
accurate mapping of 1) requirements into a formal
specification of the problem, and 2) implementation
of the specification. If the implementation is
constrained to use a very limited set of existing
code then the user interactivity may be further
minimized, but with the obvious drawback of
constraining design flexibility based on the
available software.

APPROACH TO SOFTWARE REUSE

Two general categories of software reuse have been
discussed [1], "building blocks" and "transforma-
tion". This workstation is a hybrid attempt at
initially supporting use of existing software
subroutines, but providing an increasingly more
useful transformational capability as the work-
station evolves to achieve the end result of
maximizing reuse through transformation which is
generally believed to be the more successful route
to effective software reuse.

A transformation approach, simply put, reuses
transformational capabilities as opposed to
existing modules that are pre-canned to fit needs.
A simple example being applied in the early phases
of the workstation is in computer mathematics/
algebra packages. These programs, SMP[2] and
MACSYMA[3], can take analytic mathematic express-
ions and generate procedural code for them auto-
matically in Fortran.

The Fortran code may be regenerated based on
incrementally changed specifications of the
analytic expression, i.e. based on reuse of the
transformational capabilities of these mathematics
programs.

THE DESIGN PROBLEM

The basic engineering approach to design is one
of stating problem requirements in a requirements
language, then appropriately translating this
language into a formal specification language.
This translation process typically requires
interactive user involvement to assure correct
understanding of the intended requirements, both
by the user and the system. Then based on this
formal specification, an implementation process
ensues to produce source code in traditional
programming languages, such as Fortran, C, or Ada.

The entire design process of requirements to
implemented code may be captured in a "design
data structure" which supports problem description
at multiple levels of abstraction and implementa-
tion simultaneously. It is not necessary for the
entire problem description or design process to be
at the same level of specification or implementa-
tion concurrently. It is possible that some parts
of the design would be fully implemented, some
fully specified, and others still at various levels
of definition and abstraction. The reuse of
previous designs then can be achieved based on
stored data structures.

The general design data structures support rule-
based search of various solution pathsfrom
requirements to specification and specification
to implemented code, or, that is, various designs.
A finalized design consists of a particular
path through the data structure, and the result
is an executable application. Various parts of
the data structure are saved by users for future
design efforts so that previously encountered
designs, specifications, or implementations may
be recognized and linked in as new design
scenarios are developed.

REQUIREMENTS LANGUAGE

Successful design is only achieveable if the
user requirements are clearly understood. To
make the workstation successful at understanding
the stated problem, the language available to the
user for expression should be limited in
vocabulary and structure. Thus, the workstation
requirements definition language covers a narrow
domain with restricted input syntax. Users
thereby avoid having to learn a general purpose
language that is foreign to their problem domain.
These narrow domain or limited purpose languages
can themselves by structurally based on a general
purpose language, but the individual user does
not have to be aware of this property.

TOWARD FORMAL SPECIFICATION

A parser accepts input requirements and translates
them into specification. The language is used to
allow identification of both the domain objects
mentioned in the requirements and the constraints
imposed on them. Once the objects are identified,
the specified constraints together with stored
knowledge-based constraints restrict the trans-
lation to formal specification. Also, the

workstation can suggest types of requirements that
might allow further specification to occur and
indentify ambiguous or contradictory inputs.
Eventually, the user has accurately stated the
requirements for formal specification. The
translation then implements requirement in a
formal specification language.

SPECIFICATION LANGUAGE

The specification language allows for modelling
of multiple levels of problem abstraction and
their refinement in domain specific areas. The
language used has a special purpose syntax to
assure accurate representation. The specification
language includes constructs for a "what-
description" and a "how-description" of engineering
problems. The "what-description" constructs allow
domain objects to be represented. The "how-
description" constructs allow problem solving
strategies to be represented and are automated as
much as possible from stored domain expertise.

IMPLEMENTATION METHODS

The method of implementation of the engineering
specification can be manual, by general trans-
formation, based on domain-specific rules, or in
the simplest case by direct correspondence with
library functionality. The implementation
approach taken can lead to different implementa-
tions of the same specification. A simple
example of this is where library subroutines
and modules (sets of subroutines) are both
available. The implementation may then piece
together explicitly required subroutines, or use
the more general but less efficient higher level
modules.

Once the specification has been implemented in a
language (possibly intermediate) it may be
desirable to translate this code into a target
language, such as Ada, C, or Fortran.

During the implementation phase, a history of
effort is maintained so that high level opera-
tions may be performed when possible. For
example, it should be possible to make incremen-
tal changes to the specification and determine
the effect of the changes to minimize reimplemen-
tation efforts.

STORED DESIGN DATA STRUCTURES

The stored design data structures take several
forms including subroutines, various levels of
abstracted logical structures (flow charts or
block diagrams), and pieces of specification.
A library of existing (Fortran) subroutines are
classified into a library DBMS for reuse and
perusal. Additional data structure elements
are classified into the system to support design
work until a sufficient amount of engineering
domain expertise had been collated to support a
specific domain of design activity. The adequate
classification of library information is a
fundamental basis for successful reuse of
existing elements. Both keyword systems and
general knowledge-based structures retaining
information about the design data structure
elements stored are utilized.

CURRENT WORKSTATION EFFORTS

Initial efforts are a demonstration of concepts to software development automation. The efforts during this phase result in delivery of all essential parts of the workstation design just discussed - a simple requirements language, translation to formal specification, specification language, and implementation in Fortran. The breadth of capabilities is limited in scope as might be expected. The objective is to provide NASA with a clear demonstration of an approach to software reuse and engineering design automation and could result in a fully developed and deployable system. The initial workstation is being developed on the Symbolics[4] Lisp Machine using ART[5] Automated Reasoning Tool.

The requirements language supports a very restrictive syntax for a narrow application domain - NASA's Flight Design Software. The narrow language allows requirements statement such that design constraints may be propagated and domain objects identified. A knowledge engineering effort incorporates rule-based constraints for the users based on NASA designer expertise in an effort to assist in the requirements to specification translation process.

The initial specification language consists of a graphical-flow language. Given an initial limited selection of available library routines, an engineer will not always succeed in full problem specification. After all, this is a software "development" workstation. This will be the typical state of affairs during engineering design. Some specified need would be unavailable. However, if further refinement of the specification is possible, then the workstation may be able to suggest several pieces of this lower level specification supported by library or design date structure functionality.

Finally, the code implementation of the workstation is directly in Fortran. The workstation supports automatic handling of all routine interfacing and executable module linking. Also, it is possible to request certain results presentation facilities for plotting, output, etc.

For demonstration purposes the "workstation shell" is utilized by the design team of knowledge engineers, to build a shell customization for NASA Flight Design engineers. Efforts at domain specific knowledge engineering are being maximized to demonstrate technological capabilities for a rule-based approach to Flight Design Software use.

[4]Symbolics, Inc.

[5]ART-Automated Reasoning Tool of Inference Corp., L.A., CA.

SUMMARY AND CONCLUSIONS

A Workstation targeted at improving software reusability is under development for NASA. This workstation is to accomplish the goals of engineering design automation and effective software reuse. The approach taken integrates expert system technology and software reusability principles in a hybrid reuse system which encompasses both reuse of building blocks of existing subroutines, as well as transformational capabilities to generate code in traditional programming languages such as Fortran for engineering designs. The workstation is customized for narrow engineering disciplines to allow reasonable expectations of success.

REFERENCES

(1) IEEE Software Engineering, Special Issue on Software Reusability, Vol. SE-10 No.5, September 1984.
(2) IEEE Software Engineering, Special Issue on Artificial Intelligence and Software Engineering, Vol. SE-11, No.11, November 1985.
(3) IEEE Computer, Design for Adaptibility, Vol. 19, No. 2, February 1986.

AI Tools in Computer Based Problem Solving

Arthur J. Beane
Digital Equipment Corporation

July 31, 1986

The use of computers to solve value-oriented, deterministic, algorithmic problems, has evolved a structured life cycle model of the software process. The symbolic processing techniques used, primarily in research, for solving non-deterministic problems, and those for which an algorithmic solution is unknown, have evolved a different model, much less structured. Traditionally, the two approaches have been used completely independently.

With the advent of low cost, high performance 32-bit workstations executing identical software with large minicomputers and mainframes, it became possible to begin to merge both models into a single extended model of computer problem solving.

This paper describes the implementation of such an extended model on the Digital Equipment Corporation VAX family of micro/mini/mainframe systems. Examples in both development and deployment of applications involving a blending of AI and traditional techniques will be given.

Networking & AI Systems: Requirements & Benefits

I. OPEN SYSTEMS require Networks

- Evolving multi-vendor systems   (IBM, DEC, Lotus)

- Inconsistent knowledge & databases   (DB2, IMS, dBASE)

- Decentralized decision making   (international)

II. DELIVERY SYSTEMS require Networks

- access to corporate databases

- integration with desktop applications

- price/performance optimization

III. Foundations for CONCURRENT ARCHITECTURES

- Resource sharing   (network services)

- Synchronization   (protocols)

- Load balancing   (flow control)

- Fault Tolerance   (error recovery)

IV. Example Applications

- Training   (CLUG RBBS)

- Development   (GHC, Beckman, Honeywell)

- Delivery   (Perkin Elmer, PMS)

The price performance benefits of network systems is well documented. The ability to share expensive resources sold timesharing for mainframes, department clusters of minicomputers, and now local area networks of workstations and servers.

In the process, other fundamental system requirements emerged. These have now been generalized Open System requirements for hardware, software, applications and tools. The ability to interconnect a variety of vendor products has led to a specification of interfaces that allow new techniques to extend existing systems for new and exciting applications.

As an example of the a message passing system, local area networks provide a testbed for many of the issues addressed by future concurrent architectures: synchronization, load balancing, fault tolerance and scalability.

Gold Hill has been working with a number of vendors on distributed architectures that range from a network of workstations to a hypercube of microporcessors with distributed memory. Results from early applications are promising both for performance and scalability.

An Expert System for Natural Language Processing

John F. Hennessy
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342

Abstract

This paper proposes a solution to the natural language processing problem that uses a rule-based system, written in OPS5, to replace the traditional parsing method.

The advantages to using a rule-based system are explored. Specifically, the extensibility of a rule-based solution is discussed as well as the value of maintaining rules that function independently. Finally, the power of using semantics to supplement the syntactic analysis of a sentence is considered.

Introduction

Traditional approaches to natural language processing are based upon standard compiler technology. That is, the language to be parsed is first defined. Then attempts are made to match data, entered in the form of sentences, to the structure that is derived from the language definition. The primary deficiency of this method is demonstrated when a sentence that does not match the predefined structure is encountered: the parse fails — the sentence cannot be "understood". This can occur when either a valid sentence is input that has not been accounted for or when a non-grammatical sentence is entered.

Overview

The system I am developing consists of three phases: a parser, an error corrector, and a semantic analyzer. The parser, written in Lisp, attempts to parse the input sentence using purely syntactic rules. The error corrector and the semantic analyzer, both written in OPS5, are designed as expert systems. The error corrector is invoked if the parser fails. After an attempt is made to correct the sentence, control is returned to the parser. The output of the parser is given to the semantic analyzer.

Implementation Strategy

During the parsing phase, morphological analysis is performed and dictionary definitions for the individual words making up the sentence are extracted from the lexicon. The grammar definitions are maintained in a separate file to ease maintenance. The definitions are deliberately kept as simple as possible. Consequently, both the error corrector and semantic analysis modules handle the sentence analysis process. The defined grammar for the parser is as follows:

S = NP + VP
   A sentence consists of a noun
   phrase and a verb phrase

NP = SNP + {PP}*
   A noun phrase consists of a simple
   noun phrase and any number of
   optional prepositional phrases

SNP = {DET} + {ADJ}* + Noun
   A simple noun phrase consists of an
   optional determiner (a, an, the),

any number of optional adjectives, and a noun

PP = Prep + SNP
A prepositional phrase consists of a preposition and a simple noun phrase

VP = Verb + {NP}
A verb phrase consists of a verb plus an optional noun phrase

where braces indicate optional items and the asterisk indicates zero or more occurrences.

The words in the lexicon are provided with a list of features, both syntactic and semantic, which includes the part of speech and other features, such as whether it is an animate or inanimate object, whether countable, or if color applies. No attempt is made to define an orthogonal or complete set of features.

The parser also maintains a list of all noun phrases encountered and passes that list to the semantic analyzer in order to resolve pronoun references.

The task of the error corrector is to restructure the original input into a grammatically acceptable form, using relatively simple rules. For example, if two consecutive nouns appear in the sentence, then treat the first noun as an adjective (e.g., "The house boat is docked"). Other rules handle such things as misplaced modifiers (e.g., "In the car the man plays a piano"). Some rules just throw out duplicate words – a common typo (e.g., "The man in in the boat"). Note that these rules are not necessarily restricted to syntax. That is, a semantic rule determined that "in the car" modified "man" and not "piano" in the previous example.

Many of the rules defined in the error corrector handle

conditions that could be processed by other means. For example, an ATN (augmented transition network) is often implemented in the parser. An ATN is a process of transformations that preserves the meaning of the sentence and checks for consistent features, such as verb agreement. An ATN, for example, could provide a mechanism for the parser to convert an imperative sentence into the equivalent declarative sentence. In the system I am developing, imperative sentences are handled in the error corrector by a rule which inserts the word "you" in front of sentences beginning with a non-question verb. Although ATNs have been shown to handle many cases, they complicate the parsing. Consequently, ATNs have been avoided in my research.

The semantic analysis module relies on a frame-based representation of the dictionary. This module resolves pronoun references using the noun history maintained by the parser. This module also attempts to resolve ambiguities and determines the "correctness" of the sentence (e.g., rejecting "Colorless green ideas sleep furiously"). The semantic analysis module shares many rules with the error corrector module. For example, a syntactically valid version of the above sentence, "The man plays a piano in the car" yields the same result: it is the man, not the piano, who is in the car.

Summary

I chose OPS5 as the primary implementation language for my research because it is a rule-based, forward-chaining language. Thus, it maps directly to the processing paradigm I am developing. Although OPS5 does not directly support the frame-based structure needed for the dictionary representation, I implemented this using a

small set of rules.

OPS5 seems to be a good choice not only because of its rules, but also because of the ease with which it interacts with other languages. Rules are independent thus, new rules can easily be added to expand the analysis capability. Two major benefits result. First, the control structure is never modified, thus providing a system that is easier to maintain and to extend. Second, rules for semantic analysis can be executed in conjunction with the syntactic rules. Rules are invoked as the sentence structure dictates, not as the parsing algorithm demands. By treating the natural language processing problem in this manner, it is possible to concentrate on the analysis without getting bogged down in the implementation details.

The success of the current system, as tested by the successful interpretation of the examples cited in this paper and other test cases, indicates that my initial premise is correct. That is, natural language understanding can be modeled by an expert system, enabling syntactic and semantic analysis to be combined in a manner analogous to the way natives of the language process text.

As more rules are added to the expert system portions, it seems clear that the grammar required by the parser can be kept to a very small subset of valid English. The error corrector and semantic analysis modules may be able to be combined because they share many common rules.

Although my research is ongoing, I believe that the expert system technology offers a major step in solving the natural language processing problem.

References

Barr, Avron and Feigenbaum, Edward A., The Handbook of Artificial Intelligence, Vol. 1, William Kaufman, Los Altos, CA, 1981.

Brownston, Lee et al., Programming Expert Systems in OPS5, Addison-Wesley, Reading, MA, 1985

Feigenbaum, Edward A. and Feldman, Julian (editors), Computers and Thought, McGraw-Hill, New York, 1963

Harris, Mary Dee, Introduction to Natural Language Processing, Reston Publishing, Reston, VA, 1985.

Keenan, Edward L., Formal Semantics of Natural Language, Cambridge University Press, London, 1975

Leister, Mark, Introductory Transformational Grammar of English, Holt, Rinehart and Winston, New York, 1971

Partee, Barbara H., Montague Grammar, Academic Press, New York, 1976

Expert System Technology as a Data Processing Tool

John F. Hennessy
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342

Abstract

This paper focuses on the expert system technology as a data processing tool. Several existing applications are analyzed. Their original definition as pure expert systems is contrasted with their current status as integrated systems. The architectural requirements needed to support such a heterogeneous environment are given.

Introduction

A pure expert system consists solely of rule-based processing. The solution can be expressed entirely in terms of the AI tool used to implement the expert system. Systems of this type are often advisory systems: a user poses a questions and receives an answer. In contrast, an integrated system contains rule-based processing, but its rules are supplemented by routines that are common to traditional data processing. Often the inputs come from another process, such as a process control system. The outputs may be required by another program, which need not be an expert system.

This paper examines three problems which required an AI solution, but which had special requirements that necessitated integrated system solutions.

Overview of Problem 1

In 1978, when Digital Equipment Corporation began
shipping the first VAX computers (11/780), they realized
that they were facing a potential problem. Digital was,
and still is, in the business of providing custom
computer systems — computers built to the needs of the
customer. No two orders are alike. Then each order
required an engineer to check the configuration to insure
that the machine could be built; that the specified
configuration could be supported; and that the order was
complete (e.g. contained sufficient cabling, power,
etc.). This was a time-consuming, people-intensive task.
If a method was not found to speed up the process, either
DEC's ability to ship systems in a timely manner would be
impeded or DEC would have to resort to offering standard
configurations.

A decision was made to automate the checking task. This
was attempted using traditional programming techniques.
The task, though seemingly well defined, proved too
difficult to implement: the data was constantly changing
— more parts were becoming available, while other parts
were changing or becoming obsolete. In addition, for any
given order, there were multiple "correct"
configurations. Understandably, the initial project
failed.

The AI Solution

Carnegie-Mellon University proposed attacking the problem
by building an expert system, and Digital funded the
effort. In December 1978, development began at CMU. A

research prototype, consisting of 250 rules, was demonstrated the following April. In January 1980, Digital began using the system for all VAX orders and in January 1981, full development commenced.

The result was XCON, a system for configuring computers. It was developed using a rule-based expert system implemented in OPS5, because the problem could not be solved traditionally. Two earlier attempts failed, not because the problem was not understood, but because the problem was ill-defined: requirements were constantly changing and, for any given system, many configurations were correct. Now that the problem has been solved, it is possible to look at the solution and identify areas that do have algorithmic solutions. As a result, XCON today, although still primarily written in OPS5, has sections written in seven traditional programming languages. Hence, it is now an integrated system.

Overview of Problem 2

Digital Equipment Corporation, after developing the XCON system, which insured that orders could be manufactured and supported, needed to aid its sales force in generating correct orders. The problem was that a sales representative could write an order that appeared to meet the customer's requirements, but that could omit certain components required for a complete system. That is, the sales rep could err in configuring a complete system, but the omission might not be obvious. If an order were submitted and later found by XCON to have errors, that information would have to be transmitted back to the sales representative and then back to the customer and a new corrected order placed. This resulted in time delay and frustration for the customer and for Digital.

633

C-8

The AI Solution

The basic functionality of XCON, a batch system, was taken and given an interactive, menu-driven front end. Consequently, the sales representative can specify a generic part, such as a particular class of disk drive, and the system will automatically generate the proper part plus all the necessary additional parts (for example, the correct disk controller). As a result, orders are being placed with a higher degree of accuracy than was possible before.

Overview of Problem 3

A major bank needed a system to help transportation managers minimize float by quickly clearing "non-on-us" checks, that is, checks drawn against other financial institutions. The current method required a transportation manager to analyze reports of the previous months' business. Assuming the same mix of checks for the current month, he had to determine the quickest way to clear the checks. Furthermore, he had to decide which checks should be sent to the issuing bank, to an area Federal Reserve, or to a local Federal Reserve branch. The problem involved the volume of checks (both in number and dollar amount), various processing charges, flight schedules, and interest rates. Although the problem could have been handled by conventional programming techniques, the volume of data made any real-time value of the information unlikely. When the non-numeric factors were entered – bad weather, flight delays, airport closings – no traditional system could handle the problem. An expert system was the only solution.

634

The AI Solution

A prototype expert system was designed that accepted the check information as it became available. The system stored the various presentation charges of the other banks and the Federal Reserve, accessed data from the Official Airline Guide (OAG), and accepted data changes (e.g., a cancelled flight) from the transportation managers. This data, combined with the heuristics, the expert knowledge, of the transportation managers, resulted in a prototype system that demonstrated that the difficult task could be handled.

Summary

Each of the above problems has been solved with an expert system using a variety of AI tools. A large part of each system is implemented using standard programming tools that supplement the AI code. That is, the XCON system uses standard database query languages to extract part information from a networked database. The XSEL system makes heavy use of a forms interface to interact with the user. The banking system uses Fortran for numerical calculations.

The key to each of these systems is that, although developed as expert systems, they were implemented on a traditional, general-purpose computer architecture. Thearchitecture provides easy interface with other languages while allowing the use of conventional development aids, such as code and module management tools, debuggers, and performance analyzers.

The expert system technology allows significant problems to be solved. Futhermore, the expert systems are useful because they easily fit into the standard data processing environment.

References

Feigenbaum, Edward A. and McCorduck, Pamela, The Fifth Generation, Addison-Wesley, Reading, MA, 1985

Harmon, Paul and King, David, Expert Systems, John Wiley, New York, 1985

Hayes-Roth, Frederick et al., Building Expert Systems, Addison-Wesley, Reading, MA, 1983

Scown, Susan J., The Artificial Intelligence Experience: An Introduction, Digital Press, Maynard, MA, 1985

Waterman, Donald A., A Guide to Expert Systems, Addison-Wesley, Reading, MA, 1986

Next-Generation Space Manipulator

P. Brunson, W. Chun, and P. Cogeos
Advanced Automation Technology (Robotics) Group
Martin Marietta Denver Aerospace

Abstract

In 1977, Martin Marietta Corporation, Denver Division, designed and built the Protoflight Manipulator Arm (PFMA) for Marshall Space Flight Center. It is one* of two space-qualified manipulators, the other being the Shuttle Remote Manipulator System (RMS). Since then, technology has advanced considerably in terms of components such as electric motors, control electronics, materials, sensors, and end effectors.

This paper will present a conceptual design for the next-generation manipulator of space applications. The next-generation manipulator and the PFMA will be described in more detail. These differences could have a major influence on the construction, testing, and performance of a space arm. Assessed in detail are these technologies and their effect on the design.

Servicing is an important goal of robotics in space. Parameters such as environment, type of task, time sequence, and dexterity will affect the arm and its ability to accomplish its mission. Requirements such as these are important considerations in the design of the next-generation space arm.

Introduction

The PFMA[1] is a very good design for a space manipulator. Unfortunately, it was never flown and is now nearly ten years old. At the time, the arm exemplified state-of-the-art technology. For its particular size, its use proved its principle, however, it lacked an adequate control system and architecture.
This problem is now being reversed by the Intelligent Robotics Systems Study (IRSS) program (see Fig. 1).

The PFMA (Fig. 2) is a general-purpose manipulator with distributed actuators. This paper outlines the kinematics, components, drive technology, arm segments and end effector, manufacturing/assembly, tests, and performance necessary to upgrade the arm to become the next-generation space manipulator.

Kinematics

Seven degrees-of-freedom (DOF) have proven useful in avoiding the singularities inherent in a 6-DOF arm. When adding that seventh DOF, there are three possibilities[2]: (1) a 4-DOF wrist, (2) an upper arm roll, or (3) a pitch/yaw elbow. The PFMA has an upper arm roll. This removes the majority of the singularities and enables the arm to work in both a horizontal plane as well as a vertical plane. Figure 3 is a schematic of the proposed kinematics, including a compact wrist.

The work envelope of the wrist could be improved on. By using distributed actuators, the three axes of the wrist cannot be constructed concurrently. This results in having the wrist being neither compact nor dexterous. As an improvement, we suggest using a wrist design of Mark Rosheim[3] (see Fig. 4). The wrist has all three axes concurrent. The design is simple and well engineered. The motors for the wrist are located in the forearm. By relocating the motors closer to the base of the manipulator, its dynamics are improved.

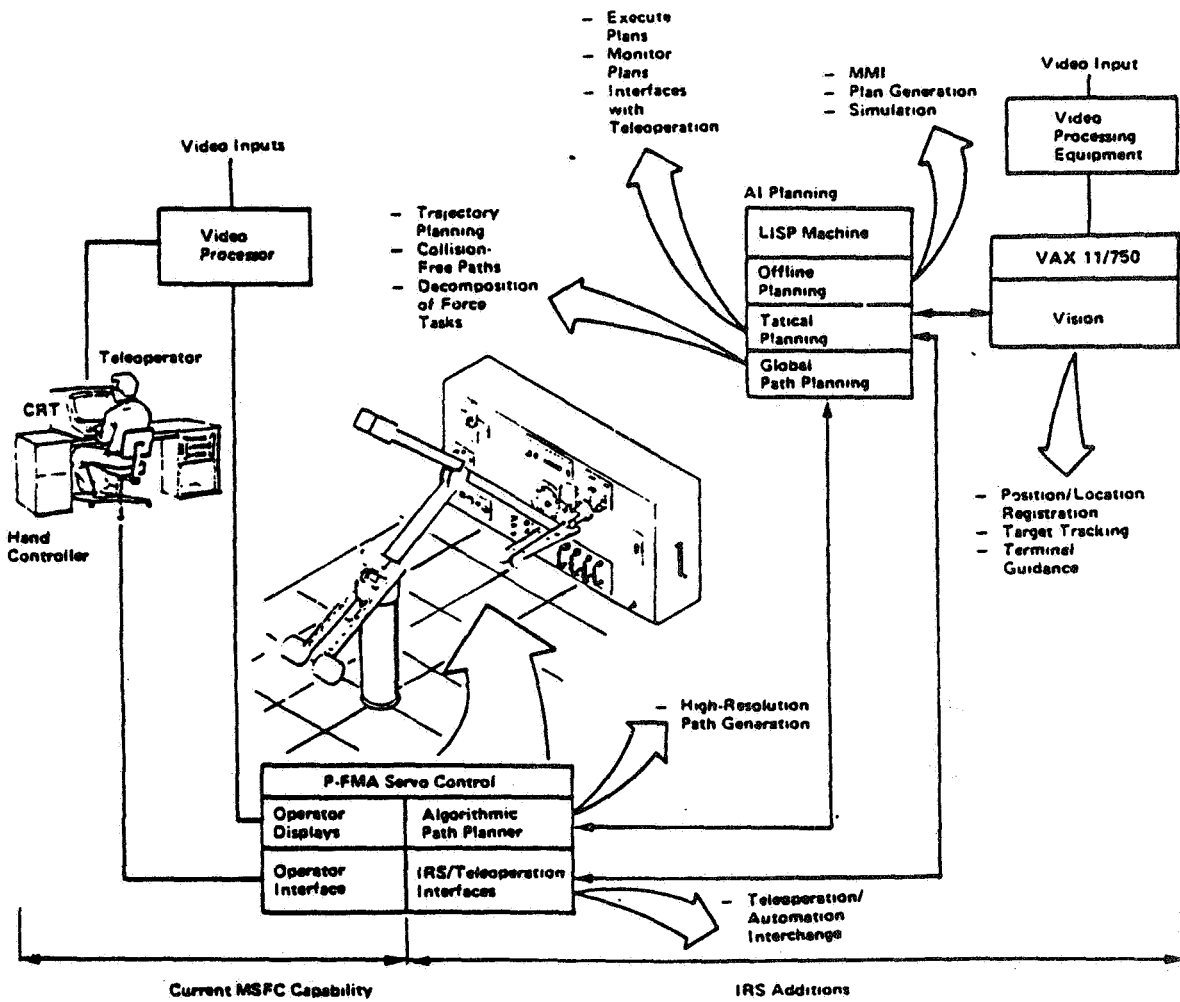*The arm was built to flight specifications and one of the motor drives completed space qualifications.
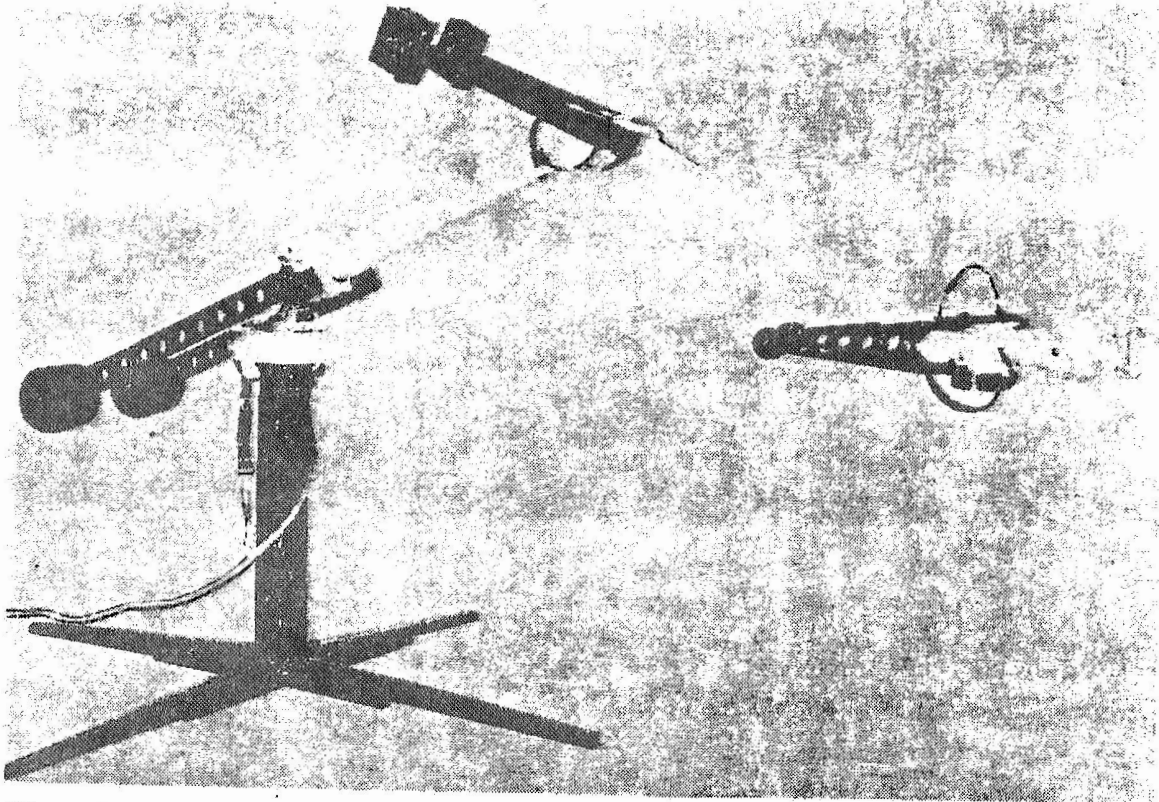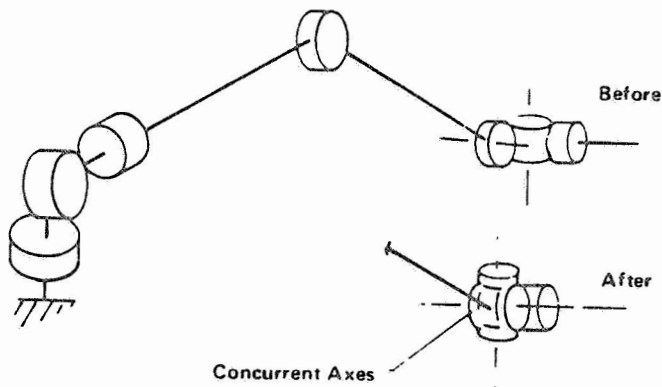
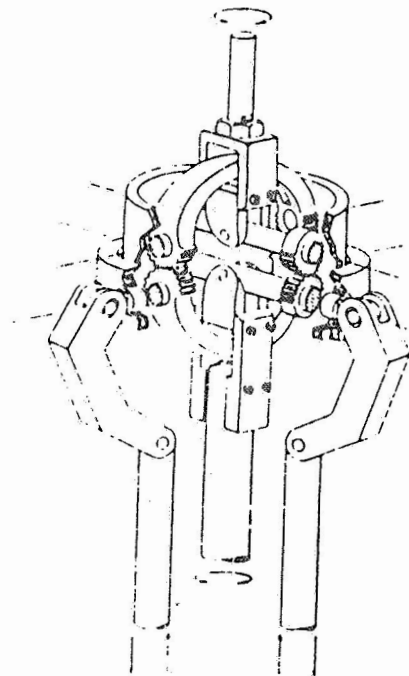Figure 1 IRSS Program

Figure 2 The PFMA

Figure 3 Arm Kinematics

Figure 4 Rosheim Wrist

639

The PFMA uses a brush-type, d-c motor using Samarium-Cobalt magnets. Another rare-earth magnet, Neodymium-Iron-Boron[4] promises more torque for the same frame size. Figure 5 compares the maximum energy product of several magnet materials. Although Neodymium-Iron-Boron has the highest rating, its output has not been consistent and will require further testing. Samarium-Cobalt is still the standard for high-torque motors.
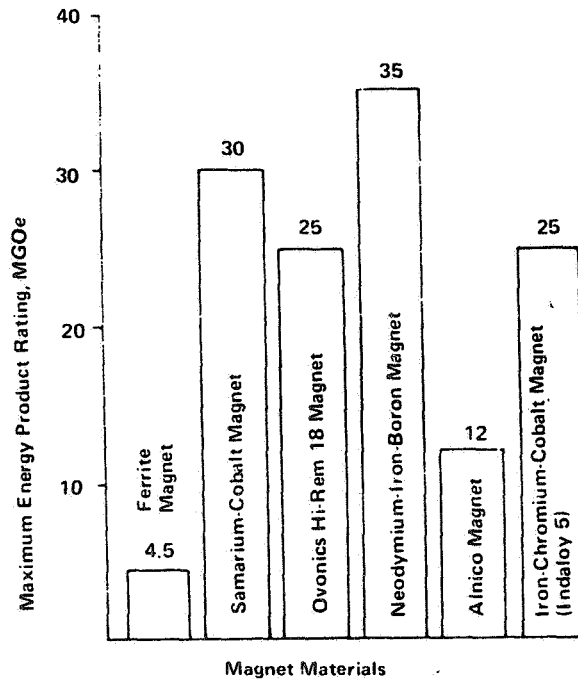

Figure 5 Maximum Energy Product of Different Magnet Materials

The motor should be brushless. With no brushes to wear, reliability will increase, while periodic maintenance will be reduced. Brush debris will be eliminated as well as arcing, which is a common problem with brush motors.

There are several devices that can be used to provide joint angle information to the arm controller. These are:

1) Brushless resolvers,
2) Inductive couplings,
3) Optical encoders.

Each of these devices can be designed with high, inherent accuracies to enable precise end point positioning. None of them contain rubbing surfaces, such as brushes, that would introduce frictional torque, limited life resulting from wear, or become a source of electrical noise. Being analog devices, both the resolver and inductive coupling require analog-to-digital conversion electronics, which can reduce the accuracies of these devices somewhat. The optical encoder provides a digital output and therefore does not suffer this problem. All of these devices can be qualified for space use and indeed have been used in this environment before. Packaging for each is available in a variety of configurations and would not limit their use.

At this time, the next-generation space arm would not use tachometers at each of the manipulator joints; instead, rate information will be derived from the individual joint angle position sensors. This approach will reduce the complexity of each drive along with its overall size and weight. In addition, the wire count will be reduced by at least two per drive along with the reduction of power to operate tachometers as a result of not using these devices.

The low level of torque required to backdrive each of the joint actuators will necessitate a friction brake so each drive can be locked when required. These brakes must exhibit fast response, long life, and no backlash when engaged. Simple devices using spring-energized friction pads are capable of generating large torques and braking energy from a small package. These devices are fail-safe in that the spring ensures brake lockup in the event of a power failure. An electrically activated coil and armature provide the release power to allow joint rotation. Brake and drive power will be controlled individually to allow each joint to "freewheel" as desired. This characteristic has been very useful during tasks such as aligning and engaging a close fitting pin or dowel.

Drive Technology

The dual-path transmission is a mature technology. One motor drives two identical gear trains that are sprung against each other at the final output ring gear (see Fig. 6). The preloading of the gears is accomplished through the use of a split gear hub. The result is a very high-precision gear train without backlash.

A manipulator such as the next-generation space arm will use a host of electrical components and subsystems that require power to operate plus receive/send signals back to the controller or operator's console. Supporting these components requires a large number of wires that must be run, in some instances, the entire length of the arm. Routing these wires along or through the manipulator links poses no real problem. The difficulties arise when the wires must be routed across the bend and roll joints. This must be done in such a fashion that it does not restrict joint range of motion, degrade the quality of the signal passing through the conductor, or introduce excessive torque that the joint drive must overcome. Slip rings have been used, but are often a source of electrical noise, friction, and limited life. A highly reliable substitute for the slip ring in limited rotation applications is a twist capsule. These devices eliminate contact resistance variation because there is no sliding contact. Flexible tapes provide the connection between the rotating and stationary circuits. The tapes provide constant circuit resistance that is not degraded by wear, shock, or vibration. The twist capsule's life can be measured in the millions of cycles, can be fabricated from low outgassing materials, and good signal isolation can be achieved.[5] Figure 7 illustrates how a twist capsule might be incorporated into the joint drive package.

All gear design, especially precision gear designs, will be depreciated or even rendered completely useless by improper choice of material. The primary objective of the gear material in the next-generation space arm will be to provide machinability to obtain a precise profile and then to retain this precision throughout the gear's life against stress, wear, and environmental effects. Desirable gear properties include the following qualities:

o   Machinability,
o   Stability,
o   Surface finish,
o   Rigidity,
o   Wear Resistance,
o   High Strength,
o   Fatigue Resistance,
o   Shock Resistance,
o   Corrosion Resistance,
o   Temperature Stability,
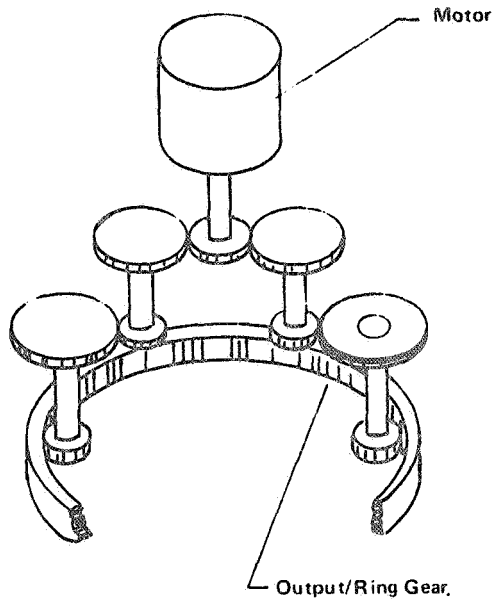o   Internal Damping,
o   Cost,
o   Availability.

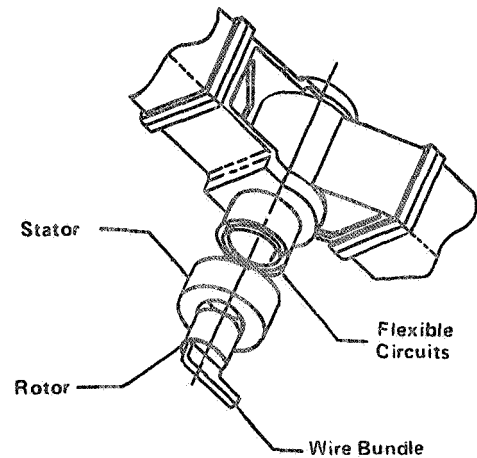Figure 6 *Dual-Path Transmission*

Figure 7 *Twist Capsules*

In precision gear trains, the first five items are the most important, while some tradeoffs can be made with the other properties.[6]

This paper will make no attempt to define what the optional material will be for each of the joint drive gears. The above information is presented to ensure the reader understands the many variables to be considered during the gear design process. Each must be carefully evaluated to ensure a drive design with the strength, life, and precision required of a manipulator in a space environment.

Bearings are a key element in the design of any high-precision mechanism. They can influence a number of drive requirements including:

1) Internal alignments and relationships,
2) Transmission errors,
3) Drive stiffness,
4) Drive stiction and friction,
5) Load-carrying capacity,
6) Life.

Torque motors and joint angle sensors of the type used within each drive require very small clearances between rotating and nonrotating parts.

642

Excessive shaft runouts or free play would introduce rubbing contact, both increasing drive friction and damaging the devices.

The quality of the gears used (AGMA 12) in the reduction path demand shafts that run true and introduce no backlash or transmission errors. Drive stiffness must be controlled to ensure all resonances are well above the operating bandwidth of the of the system. Stiction and friction levels must be minimized to maximize drive controllability. All of these requirements are influenced by the bearing suspension system selected. Bearing characteristics that must be evaluated before selection include:

1) Tolerances,
2) Preloading,
3) Load rating,
4) Raceway geometry,
5) Lubrication,
6) Bearing fit-up,
7) Retainer design.

Each of these characteristics can affect one or more of the drive requirements and therefore must be carefully evaluated.

In addition, drive housing and shaft designs must share the same degree of precision as the bearing to fully exploit its capabilities. Accommodations in the design must also be made to ensure proper bearing operation throughout the temperature envelope without loss of precision or stiffness.

Arm Segments and End Effector

To provide a viable space-rated robotic arm, there are certain design criteria that must be met for the arm to function optimally.

The arm segments should include the following:

1) Be lightweight with high-strength and stiffness characteristics.
2) Allow for a "clean" method to route wires.
3) Neatly and compactly house the actuators and positioning sensors.
4) Contain joints that are simple and quickly engaged and disengaged; they should lock rigidly together.
5) Be designed to be modular.

The following is proposed to accomplish these goals:

1) The arm should be made of a mixture of continuous and noncontinuous magnesium graphite composite; continuous where machining is not necessary and noncontinuous where it is.
2) The arm segments should be hollow to allow for the wires to be run through their centers, keeping them hidden yet accessible.
3) There are several acceptable joint designs; two of the most promising designs feature a bayonnet mount or a bulkhead mount.

For the manipulator to be a viable tool, it must have a general-purpose gripper and yet be able to use several different servicing tools. Articulated hands are experimental and are only found in the laboratory. Their technology is not mature and requires additional considerations for housing the finger actuators thus complicating the wrist interface.

The most advanced gripper is the "Smart End Effector" developed by JPL[7]. The end effector incorporates an integral force sensor at its base and the jaw combines tactile with proximity data. The intermeshing jaw provides excellent

prehension. In addition to the gripper, the arm requires a system that incorporates different tools. Martin-Marietta has developed a power takeoff system with an extra motor built in the gripper that powers an assortment of compatible tools. As a result, the arm needs only the one integrated motor for all its tools. SRS Technologies[8] has taken a deeper investigation into the necessary tools for servicing.

Manufacturing/Assembly

Martin Marietta has been machining and assembling these types of drives since the mid 1970s. They have gained the technical expertise necessary to construct the various stages and to assemble the parts in a clean room to tight tolerances. Each step is very critical and built to exact specifications. Any inconsistencies or missed detail could degrade the performance of the drive. As a result, the drives are rugged and durable. Similarly, the arm segments are inspected. All devices are tested at each subassembly level. The final product is a flight-worthy system.

Testing

Testing of the arm is accomplished at two stages: (1) drive and (2) complete manipulator. The individual drives are characterized separately. Its performance is checked by studying torque versus speed curves and torque versus current curves. The plots are differentiated by varying the input voltage, load, or pulse-width modulation (PWM) signal. The tests are conducted through an Electromechanical Test Support Unit (ETSU), a Martin Marietta-designed test bench that is reconfigurable to perform a variety of tests.

In addition to performance testing, each drive is tested for mechanical and electrical friction at running and breakaway levels. The next test is to measure the drive compliance. Having passed performance tests, the drives are put through random vibration and thermal-vacuum testing. The prototype drive has one additional test: life cycle in a thermal chamber.

The next level of testing is at the assembled stage. The drive tests are repeated at the systems level. Moreover, a thermal balance test and an electromagnetic interference/compatibility test are performed. Complementing the characterization tests will be several robotic tests[9] to obtain the following data:

1) Geometrical values
   a) Workspace,
   b) Static behavior,
   c) Position accuracy,
   d) Path accuracy,
   e) Overshoot,
   f) Reproduction of the smallest steps,
   g) Synchronous travel accuracy,
   h) Long-term behavior;
2) Kinematic values
   a) Cycle Time,
   b) Speed,
   c) Acceleration;
3) Power and Noise Values;
4) Dynamic Values
   a) Force,
   b) Dynamic compliance,
   c) Dynamic behavior of the structure.

The extensive testing qualifies the design and the hardware for its mission in space.

Performance

Because of the increased strength of the motor magnets, the same diameter motor will produce greater torque. If the current torque values are sufficient, the design may favor smaller and lighter drives. Either way, the drive will produce a higher torque-to-weight ratio. Additionally, elimination of the tach generator will lighten the drive and make it more compact.

The mature, dual-path gear train exhibits zero backlash without compromising friction. The improved bearing design removes radial play and thus creates a stiffer drive. Using high-quality gears and precision assembly techniques, the gear tooth stresses can be distributed uniformly. Gear wear is reduced and thus its life extended. Brushless motors further extend life and reduce maintenance.

The approximately 100 to 1 gear ratio is very backdriveable. The higher resolution position sensor is combined with the IRSS control architecture, resulting in a high-performance arm (Fig. 8).
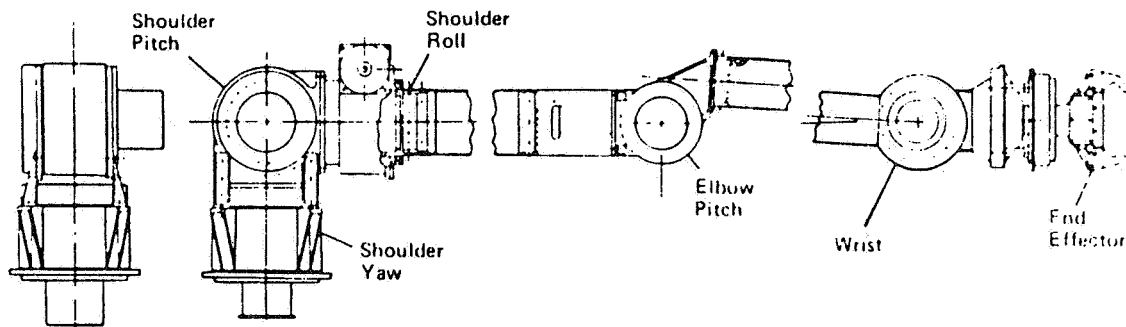

Figure 8 Next-Generation Space Manipulator

Conclusion

Upgraded components increase the performance of the individual drives. The final product is a very precise positioning arm that is characterized by a high torque-to-weight ratio. Similar upgraded drives have been built, tested, and space-qualified by Martin Marietta.

A compact wrist gives the manipulator increased dexterity to maneuver in its work envelope. Advanced materials make the arm segments stiffer without sacrificing weight or a slim profile. New concepts in attaching the various components lends the arm to a modular approach. Moreover, the use of twist-capsules eliminates noises that are inherent in existing slip rings.

The elimination of the tach generator and the placement of the drive electronics close to each drive reduces the number of power and signal wires. The smaller wire bundles can now be routed more efficiently.

Finally, the incorporation of the aforementioned upgrades on the PFMA will result in a very high-performance manipulator system for space. Combined with the control system and computer architecture of the IRSS program, the arm can be available in a very short timeframe. It has the potential to be the next space manipulator.

Acknowledgments

The authors wish to thank Bill Britton of Martin Marietta, Denver, for encouragement and helpful discussions.

References

1. Protoflight Manipulator Arm (PFMA). MCR-77-201, Final Report, Martin Marietta, Denver Division, April 1977.

2. J.M. Hollerbach: "Optimum Kinematic Design for a Seven Degree-of-Freedom Manipulator." Preprints 2nd International Conference, Robotics Research, Kyoto, Aug. 19-23, pp 349-356.

3. M. Rosheim: Robot Wrist Development. NASA Conference on Robotics and Flexible Automation, Jan. 14, 1986.

4. K. Dekker: "New Materials for Improved Motor Designs." Motion, May/June 1986, pp 6-13.

5. Poly-Twist Capsules Catalogue. Poly-Scientific, Blacksbury, Virginia.

6. G. Michalec: Precision Gearing: Theory and Practice. John Wiley and Sons, 1966.

7. A. Bejczy, E. Kan, and R. Killion: "Integrated Multisensory Control of Space Robot Hand." Preprint Proceedings of the AIAA Guidance and Control Conference, Colorado, Aug. 19-21, 1985.

8. J. Cody: Interchangeable End Effector Tools Utilized on the PFMA-Task 1 Final Report. Refer to Marshall Space Flight Center, MFS-27125.

9. H. Warnecke, R. Schraft, and M. Wanner: "Performance Testing." Handbook of Industrial Robotics, Chapter 10. Simon NOF (editor), John Wiley and Sons.

DMES

Distributed Module Expert System

By

Steven C. Purinton
Caroline K. Wang

MARSHALL SPACE FLIGHT CENTER

Description

DMES is an expert system concerned with the execution of a realtime training simulation on a distributed system.

History

A facility was developed at Marshall Space Flight Center to provide training for the Spacelab payload crew and mission specialists. The training facility originally consisted of a two processor Host with shared memory and a Spacelab mockup. Communications links between the Host computers and the mockup provided for serial ASCII, display, discrete, and analog data. Modules functionally representing the experiment computer I/O unit, operating system, and application software as well as payload hardware, and environment were developed. A control task was developed to provide for the selection of a training configuration, module scheduling, mode control, and monitoring of the simulator. Modules are components of the simulator and may be a single task or multiple tasks with a single function.

As the Payload Crew Training Complex evolved, a telemetry communication link to the Payload Operation Control Center, a shuttle aft flight deck workstation, scene generation, and two processors were added to the host. Training sessions were conducted using one processor dedicated to communications and two processors hosted the experiment modules, system modules, and control tasks. Figure 1 depicts this system.

The sequence of execution was complex because some of the processing was serial and other parallel. Some modules, such as environment, time, and, input/output are required to execute only once between instances of an experiment or application software module. Multiple instances would (or could) mean the omission of an event in the simulator. This condition occurs when execution is not predictable as with multiple processors or a timesliced system. Similarily, multiple instances of an experiment or application would cause effects as wasting time, a mixture of input-output data, or other undesirable problem.

With three active processors as Host, a training session configuration would evolve using experience, intuition, and testing. The order of execution and location for execution would be the parameters varied. The verification process could consume several days of testing.

The simulation has been transfered to a single VAX and expansion to a distributed system is in progress. The training simulator is composed of serial and parallel modules and must also deal with network delays and the problem of updating a centralized data base. This growth presents the facility with a much more complex problem when it comes to determining a configuration which will execute in real time. In response to this problem an expert system is being developed to allow the analysis of a training session.

Implementation

The expert system loads the knowledge base, which consists of the following items:

1. Number of processors
2. Default tasks
3. Number of experiment modules
4. Processor location for each experiment module
5. Data dependance for each module
6. Execution time for each module
7. Network delays
8. Database read/write
9. Basic cycle time

A knowledge editor allows input and adjustment of these items.

A control task is started in the first processor, this in turn schedules the execution of other tasks and modules within the first processor and schedules the execution of the control task in additional processors. A cycle is complete when all modules in all processors have finished execution. Execution is defined as being active for the amount of time specified in the knowledge base. Any database accesses from the other processors will extend the active time by the amount of access time. A failures would be a database access out of sequence or too much time used during a cycle.

Enhancements

The operating system is not part of the expert system, but rather the control task for the simulator is modeled. As the expert system is refined it may be necessary to include the operating system and it effect as elements. Different methods to update or distribute the database need investigated. Also distribution changes in functions such as display updates or I/O can be included. Accuracy and resolution can be improved as needed to make the expert system useful. Randomness can be used in some areas to improve problem detection.

PAYLOAD CREW TRAINER

SPACELAB/AFT FLIGHT DECK MOCKUPS

ANALOG/DISCRETE DISPLAY/KEYBOARD

ANALOG/DISCRETE DISPLAY/KEYBOARD

PDP 11/70 CONTROL MODELS

PDP 11/70 MODELS

SHARED MEMORY

PDP 11/24 COMMUNICATION
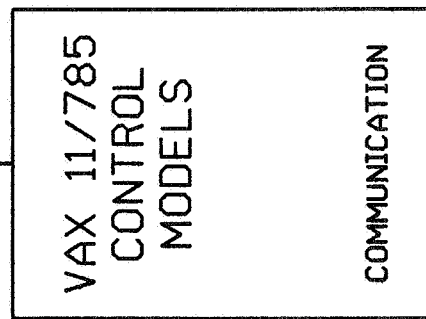
PDP 11/70 DEVELOPMENT BACKUP

DECNET

DECNET

HOSC

- PARALLEL PROCESSING
- HIGH-SPEED COMMUNICATIONS BY SHARED MEMORY
- EMULATED SHARED MEMORY TO ALLOW USE OF PARTIAL SYSTEMS

649

REPLACEMENT PAYLOAD CREW TRAINER

SPACELAB/AFT FLIGHT DECK MOCKUPS

IEEE FOR ANALOG/DISCRETE
DISPLAY/KEYBOARD

```
VAX 11/785
CONTROL
MODELS

COMMUNICATION
```

DECNET

HOSC

- ALL COMPONENTS IN A SINGLE MACHINE

- MULTIPLE SIMULATORS POSSIBLE IN A
  SINGLE MACHINE

650

DISTRIBUTED EXPANSION OF VAX-BASED PCTC

- DISTRIBUTED SYSTEM

- REMOTE CONTROL OF EXECUTION

- REMOTE UPDATING OF DATA

SPACE LAB/AFT FLIGHT DECK MOCKUPS

IEEE FOR ANALOG/DISCRETE DISPLAY/KEYBOARD

VAX 11/785 CONTROL MODELS

COMM

MICROVAX REMOTE CONTROL MODEL

MICROVAX REMOTE CONTROL MODEL
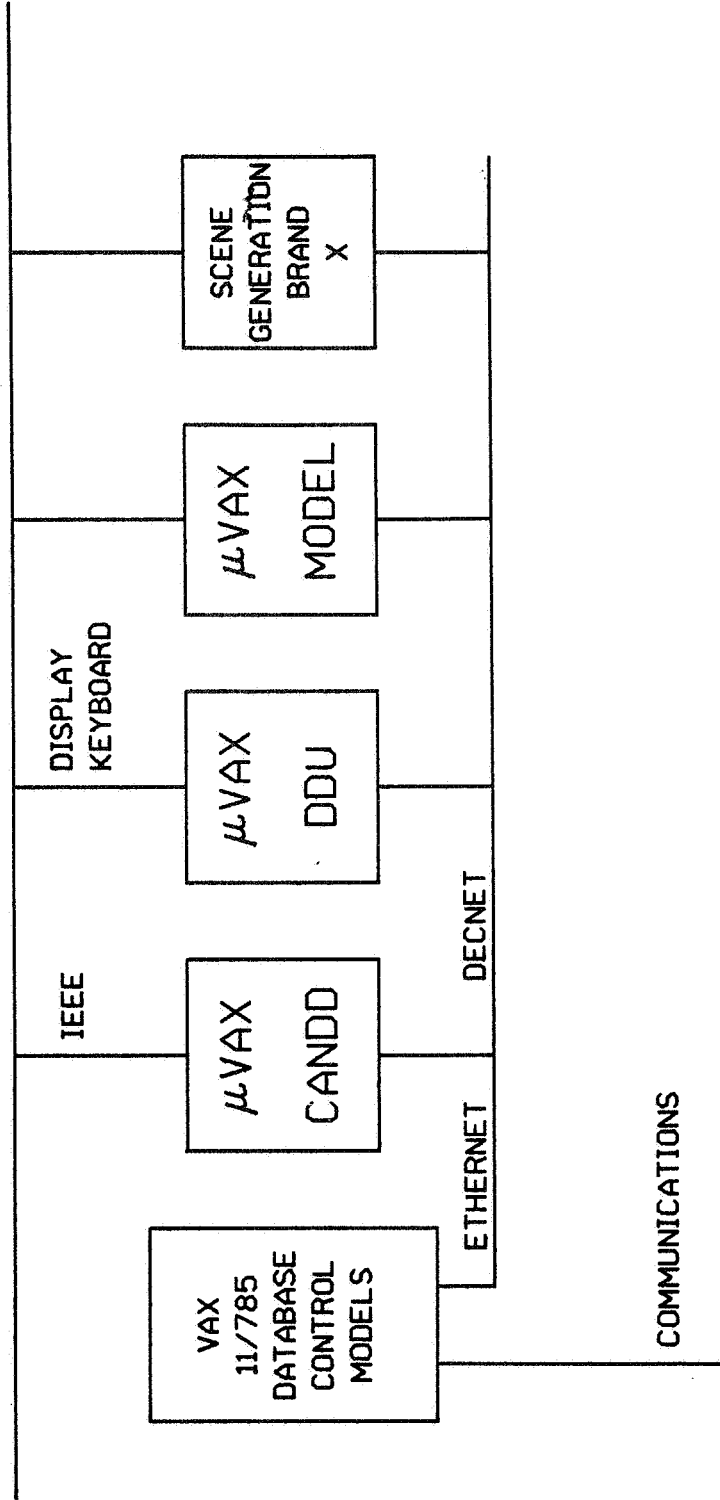
ETHERNET/DECNET

DECNET

HOSC

651

DECENTRALIZED RESOURCES

SPACELAB/AFT FLIGHT DECK MOCKUP

652

Features

Distributed Module Expert System (DMES) is a prototype frame based Expert System designed to evaluate timing and loading conditions for a digital simulator (for Payload Crew Training).

DMES was developed on Symbolics 3670.
DMES has the following features:

1. An user friendly interface utilizing the Symbolics window system capabilities to:

   a. Create information data base
   b. Display information data base
   c. Update information data base

2. Automatic knowledge base generation.

   Automatically generate the knowledge base from the information data base.
   The knowledge base includes:
   The procedure for each individual processor.
   Information for graphics display.

3. Real time graphics display

   DMES is utilized to operate one cycle per second.
   DMES Checks on the multiple processors of the simulator and searches for the current experiment for each processor and puts each experiments into the queue stack and fires the experiments on schedule.
   DMES also graphically-displays the current status of the experiments and the same time stores the detail information into the buffer for future analysis.

4. Generate DMES history file
   There is an option for writing the history file from the history buffer.

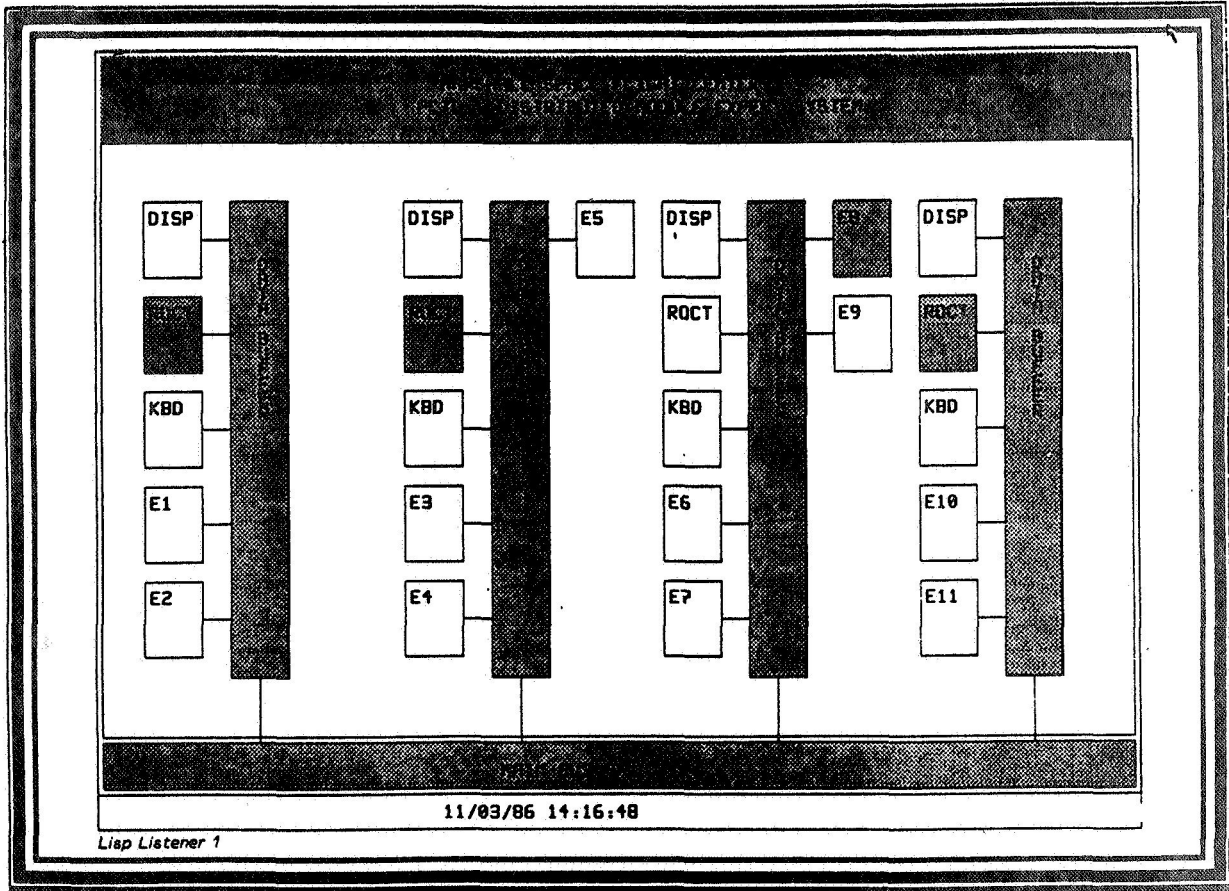5. The history file can be graphically displayed for closer analysis.

Benefits

DMES was designed by using common lisp for the expert system portion and zetalisp for a user interface window flavor techniques and graphics.
It can be transfered to other systems by rewriting the user interface and graphic display.

DMES can be helpful in studying and modeling the complex distributed system for multiple processors. It helps to quickly see the problem and also can modify the model easily.

653

CREATE DATA BASE
DISPLAY DATA BASE
UPDATE DATA BASE
EXIT

file name: c:>caroline>dnes>demo.dat

Distributed Module Expert System

file name:		c:>caroline>dmes>demo.dat
number of	processors:	4
Processor	name	p1
number of	experiments:	2
ROCT	wait time	4
	set inactive	T
	wait time	10
	set inactive	T
DISPLAY	wait time	2
	read data	T
	wait time	6
	set inactive	T
KBD	wait time	1
	write data	T
	wait time	2
	set inactive	T
Experiment	name	e1
EXPERIMENT	wait time	2
	read data	T
	wait time	4
	write data	T
	set inactive	T
Experiment	name	e2
EXPERIMENT	wait time	4
	read data	T
	wait time	6
	write data	T
	set inactive	T

Press any key for next page

COMMAND LINE

DISP			DISP		E5	DISP		RDT	DISP	
						ROCT		E9	RDCT	
KBD			KBD			KBD			KBD	
E1			E3			E6			E10	
E2			E4			E7			E11	

11/03/86 14:16:48

Lisp Listener 1

11/03/86 14:16:48 Caroline   USER: Run   • C:>caroline>dnes>dnes-history.dat 0

654

UTILIZATION OF ARTIFICIAL INTELLIGENCE TECHNIQUES
FOR THE SPACE STATION POWER SYSTEM

Thomas C. Evatt and Edward W. Gholdston

Rockwell International/Rocketdyne Division
Canoga Park, California

Abstract

Due to the complexity of the Space Station Electrical Power System (EPS) as currently envisioned, artificial intelligence/expert system techniques are being investigated to automate operation, maintenance, and diagnostic functions. A company-funded study has been conducted to investigate this technology as it applies to failure detection, isolation, and reconfiguration (FDIR) and health monitoring of power system components and of the total system. Control system utilization of expert systems for load scheduling and shedding operations has also been researched. A discussion of the utilization of artificial intelligence/expert systems for Initial Operating Capability (IOC) for the Space Station effort is presented along with future plans at Rocketdyne for the utilization of this technology for enhanced Space Station power capability.

Introduction

Automation will need to be an integral part of the Space Station Electrical Power System (EPS). At Initial Operating Capability (IOC) and beyond, the implementation of artificial intelligence technologies is envisioned at increasingly sophisticated levels to minimize the need for crew interaction and to maximize station productivity. The goal at IOC is for the power system to automatically operate, reconfigure itself in case of failure, monitor system health, and provide a diagnostic expert system to assist with maintenance, fault isolation, and component replacement (Ref. 1-9). Current research at Rocketdyne is focused on the implementation of these functions, with projections beyond IOC for the implementation of advanced artificial intelligence (AI) hardware and software as they are developed.

IOC Power System Automation

There are four basic motivating factors for inclusion of advanced automation in the station power system:

- Crew safety (hazard exposure, extravehicular activity (EVA) time, etc.)

- Crew and ground operational efficiency

- System/subsystem reliability (maintenance and replacement costs)

- Mission success (power availability for scientific experiments, etc.)

With these requirements in mind, an initial power system architecture has been formulated with capabilities that can be expanded in a series of incremental steps, each of which would improve the reliability and efficiency of the overall power system. The current EPS design consists of solar dynamic (SD) and photovoltaic (PV) power generation units, energy storage equipment, and power management and distribution systems. Figure 1 shows a schematic of the complete power system illustrating the locations of power sources, bus switching assemblies, and load points. The widely distributed nature of the sources and loads has led to a computer control architecture that reflects a similar distribution around the station. The processors are configured in a hierarchical arrangement that allows many control functions to be retained at lower, local levels, as shown in Fig. 2. At these lower levels, the power system will generally have "classical" control (algorithm control to a given setpoint), with manual flight crew override, and ground override as a last resort. Global coordination and other higher level functions, such as load scheduling, are
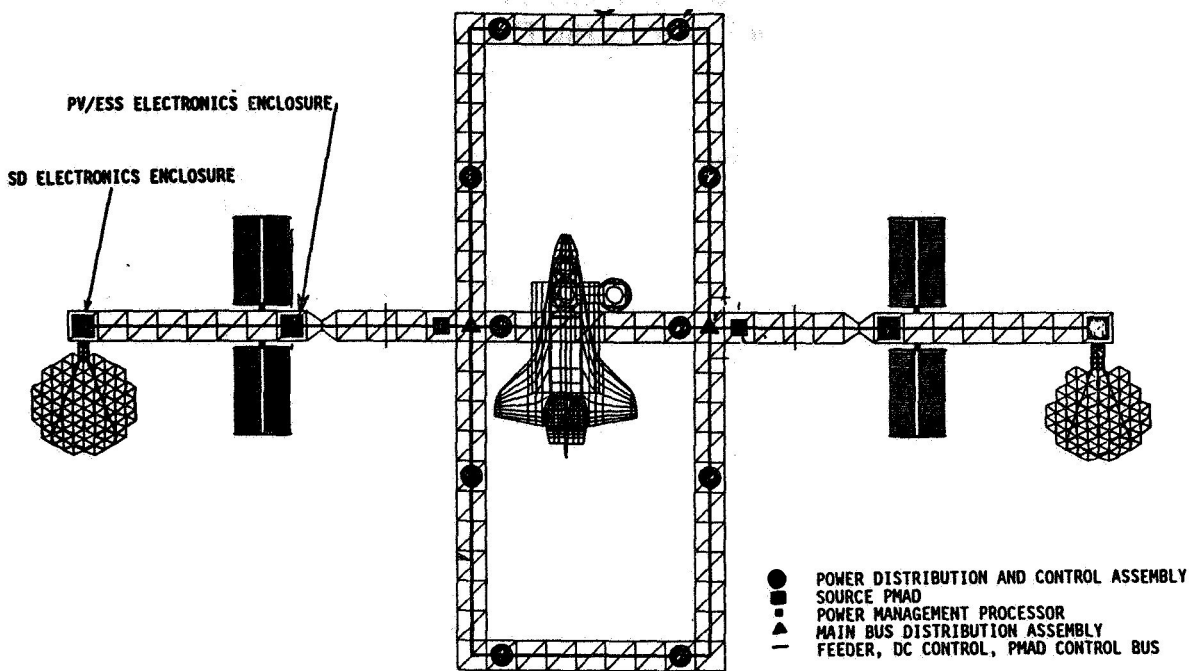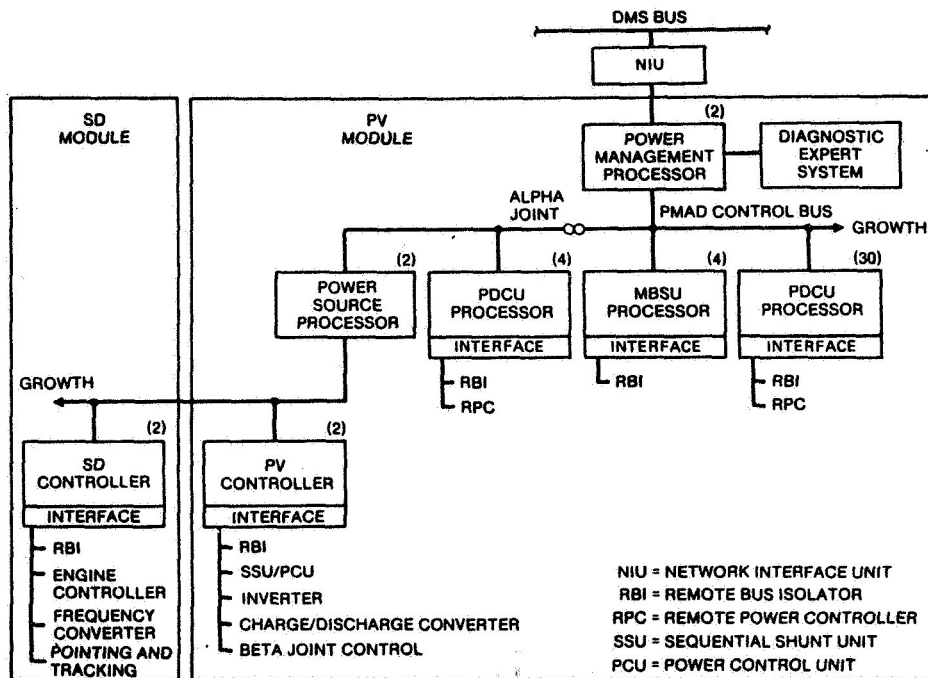
Fig. 1. Space Station EPS Components Locations

Fig. 2. Control Functions

centered in the Power Management Controller (PMC), at the top of the hierarchy. It is at this level that interaction with a dedicated expert system is being considered.

The kinds of functions being evaluated for an expert system, operating as an extension of the PMC, include advanced health monitoring and trend analysis, failure mode analysis, equipment fault prediction, load flow analysis, and state estimation. For many of these functions, the mathematic calculations required are extensive. For example, some state estimation techniques require iterative manipulation of large line-

flow matrices (Ref. 10). Because of the need for the EPS expert system to utilize such computational data, there will have to be a close coupling of information between the conventional PMC processor and the expert system, with the PMC performing most calculations, leaving the expert system free for heuristic reasoning. Studies are currently being conducted to determine the best implementation approach for different types of expert systems. A load management expert system, for example, will require real-time data and might therefore be a candidate for a conventional processor approach (Ref. 2). A diagnostic expert system, on the other hand, requires a great deal of detailed information on the system configuration, models of performance, failure modes effects analysis (FMEA), etc., in a knowledge base that would run inefficiently on a conventional processor and might require a specialized symbolic or parallel processor.

Electrical Power System Evolution

The basic architecture for future EPS configurations will include enhancements to the IOC hardware and software architecture and functionality. The addition of a specialized expert system processor, such as a compact LISP processor, is a logical first step. Early enhancements are expected to involve the development and testing of a more advanced diagnostic expert system that will be stationed on the ground and receive its data from the Space Station via telemetry. To verify the performance of a prototype expert system, two sources of system data will be utilized: one will be the actual operating data from the orbiting station, and the other will be the results of running off-nominal tests in the Rocketdyne and the NASA Lewis Research Center (LeRC) test facilities. After operational experience is obtained from day-to-day interaction with the power system (failure rates, histograms of the types of failures, etc.), the ground-based expert system could become validated and ready for implementation on the station. The iterative development process will utilize the knowledge of power system experts during the initial stages of the program well before the Space Station is operational. The NASA LeRC Space Power Facility and the Rocketdyne Space Power Electronics Laboratory will be utilized to simulate failure modes, and to exercise diagnostic strategies for early knowledge-based technology assessment.

System Architecture

The IOC architecture represents a conventional distributed and hierarchical architecture approach that greatly simplifies the difficulties typically involved in adding functional capability by utilizing specialized processors. However, it is important in the EPS IOC architecture design to allow for the direct incorporation of specialized processor architectures for the power system. Issues of how to utilize LISP (Ref. 11-13) processors are now under consideration for military applications. The Texas Instruments Compact LISP Machine developed under DARPA funding represents an important step in utilizing advanced technologies for military aircraft and autonomous vehicle applications. This processor has been designed to be fully MIL-SPEC qualified and will undergo testing within the next few years. The bus architecture has been designed to accommodate the specialized bus requirements as well as providing for the addition of 1750A processor architectures running in parallel with a compact LISP processor. This technology represents a possible growth option for the Space Station EPS, and the utilization of such a backbone architecture is a potentially logical direction. Thus, at IOC, the PMC could be configured with a conventional processor architecture with the capability of adding a LISP machine or another appropriately configured conventional processor after initial ground testing of the system is completed. Alternatively, the IOC system management controller could be removed and replaced with a program management controller actually containing the LISP machine and software.

Diagnostic and Trend Analysis

Diagnostics of key power system components represent a significant requirement for the EPS (Ref. 6, 14, and 15). The safety, risk, and associated cost issues of power system component unavailability is prompting the development of a knowledge-based diagnostic system that detects deterioration of power system components in the early stages. Since typically, a human operator must diagnose a situation by interpreting what the actual condition of the equipment is from a number of measured variables, the diagnostic knowledge of the operator will be the critical link in making decisions for power system maintenance and operation. Steady-state system stability will also be an issue when considering

657

removing any unit from the interconnected grid (Ref. 13 and 16). Projected resource needs, as well as the availability of backup units, further complicates the picture. The utility industry has tended to counter these concerns by increasing the number of variables monitored and improving the quality of the display of these variables. This, however, is in contrast to the real need of an operator, especially under decision-making stress, to know the actual condition of the equipment (what is actually malfunctioning, not just which variables are abnormal). An on-line diagnostic system would provide the Space Station crew with a significantly higher level of information to allow them to make better quality decisions, especially under rapidly changing conditions.

Failure Detection, Isolation, and Reconfiguration for Growth

The Space Station power system will grow in size and evolve making it difficult to characterize a static maintenance problem. This makes crew training in the maintenance of particular systems difficult and could result in a major dependence on ground expertise. It also makes it difficult to specifically configure generalized automated test equipment if any shared resources are used. Requirements for automatic fault detection/fault isolation have been established for meeting Space Station availability requirements (Ref. 14 and 17). A typical failure detection and isolation configuration is shown

in Fig. 3. A base of historical performance data will be required for trend monitoring and incipient failure detection of subsystems (with similar requirements for health monitoring). Specific "signature" assessment modules must be defined for the distinct failure mechanisms of different component types—mechanical, electronic, fluid, etc. Figure 4 shows a diagram of this kind of diagnosis and performance monitoring.

The large mix of fault isolation procedures results in the requirement for a procedures storage system of some sort, and "user friendly" access to these procedures. The dynamics of the EPS could result in a configuration control problem in trying to keep the procedure storage system current. This would be solved in part by the use of knowledge engineering techniques to uncomplicate and speed the search for the required procedures.

Failure detection, testing, and maintenance have been categorized as knowledge-intensive and experienced-based tasks (Ref. 1). Although test procedures and maintenance manuals normally will contain recommended detection, localization, testing, and maintenance and repair actions, their use for the EPS does not ensure completion of troubleshooting and repair tasks in a timely fashion. Skilled power system maintenance staff, apart from using test procedures and maintenance manuals, use heuristics and an understanding of how the system works to solve component problems. For example, when a symptom such as
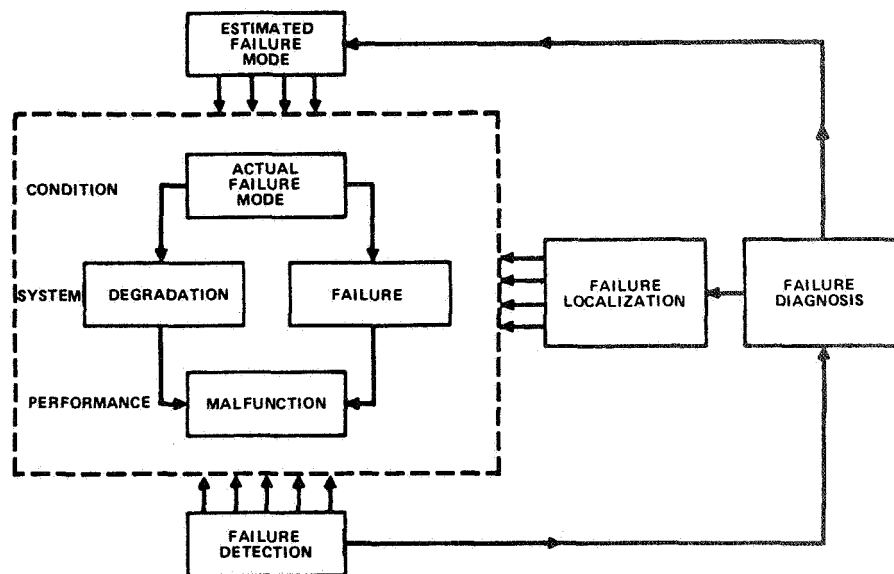

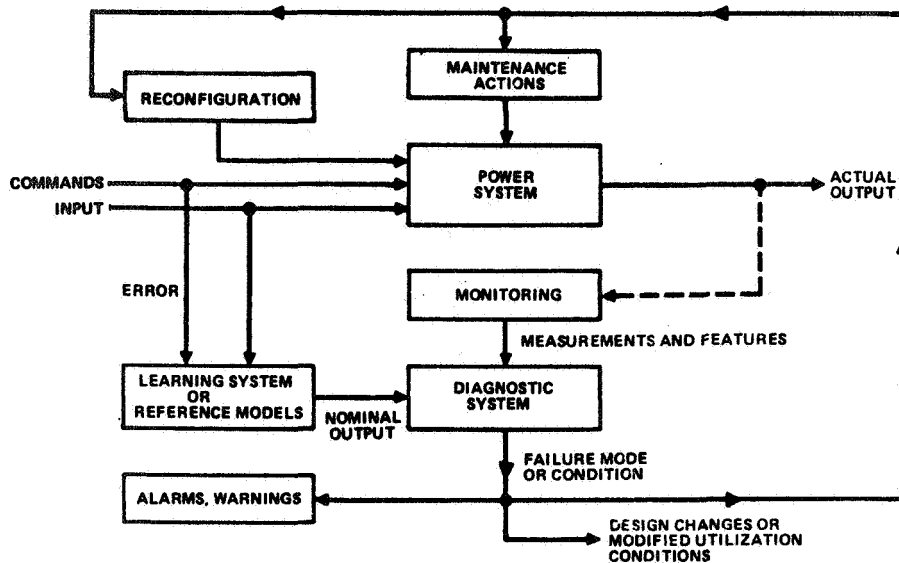Fig. 3. Failure Detection and Diagnostics

Fig. 4. Diagnosis and Performance Monitoring

reduced power quality becomes noticeable in real-time and historical data, a power component diagnostician will begin a clearly defined list of troubleshooting procedures to localize the problem. However, if the typical troubleshooting techniques do not yield a solution, the experience of the diagnostician is required to initiate "nonroutine" troubleshooting. It is this type of knowledge that enables performance at an exceptional level. Based on years of experience, a highly skilled test/maintenance technician would develop the following traits:

⊛ Familiarity with procedures and documented maintenance manuals

⊛ Understanding of the relationships between observed symptoms and the malfunction of specific orbital replacement units (ORU)

⊛ Intuitive understanding of how the system works

⊛ Intuitive understanding of how the system will behave when certain subsystems or ORUs fail

The high level of performance of experts suggests that, confronted with a problem, they analyze the problem in a structured manner rather than randomly trying all possible alternatives. The identification of these tasks is crucial to the success of modeling the associated reasoning processes distinctly, and

modeling them as independent modules is extremely important to achieving a high degree of performance and modularity for future expansion and modification of the power system. Some of the above tasks identified are interpretation, diagnosis, troubleshooting, repair, design, planning and scheduling, monitoring, and reasoning with functional models. It is essential that an expert system be used that has a knowledge of the other subsystems and/or is capable of communicating through a shared-knowledge interface.

Failure detection, isolation, and reconfiguration (FDIR), known in the power industry as contingency management, deals with the problem of detecting deviations from normal behavior (which will be designated "failure modes") in a specified complement of system components (sensors, actuators). It includes isolating the particular component that has failed, and reconfiguring the power system by providing switching to redundant components or power buses. Recent developments in FDIR methods that may be applicable to the evolving EPS expert system can be conceptualized as consisting of four separate related stages:

1. Residual generation (using state estimation/model following)

2. Information collection (FMEA, and criticality of failure modes)

3. Decision making (determine if 1 and 2 indicate a failure mode

4. Reconfiguration (determine how the power system can be reconfigured in an "optimal" fashion to maintain satisfactory performance)

In modern avionics systems, failure detection and built-in testing (BIT) are the cornerstones used to signal when the principal path of information or data flow should be switched to an alternate backup path to preserve proper overall system performance (Ref. 18). While on-line BIT is usually used to rapidly uncover catastrophic or hard failures, other more sophisticated techniques (based on modern control/estimation/decision theory) are utilized to detect more subtle, or "soft" drifting-type failures. These are the failures that do not necessarily cause the system to shut down entirely but may still degrade system performance with the passing of time. The following detailed issues must be considered when implementing this kind of FDIR algorithm:

1. Nature of the soft failure (i.e., its type and severity)

2. Observability of the failure's effect within the measurement (i.e., degree of perceptibility)

3. Length of time required to accumulate enough data to register the presence of the failure in the presence of background disturbances such as quantization effects, sensor noises, response of the station data management system (DMS), computer cycle speed, etc.

4. Degree of distinguishability from other types of failures for unambiguous failure isolation

5. Ease of corrective actions (e.g., switching to an alternative analytically or functionally redundant path on-line or postponing repair until back at the maintenance module)

These considerations are important factors that contribute to an overall failure detection isolation and reconfiguration policy. However, the action of failure detection is fundamental to every system reconfiguration policy, and the

technological areas of failure/event detection are undergoing rapid change as new ideas enter the field.

Trend Analysis

Trend analysis requires the collection of data over long periods of time for detailed evaluation. Trend analysis operates on a single data series at a time, either by smoothing or estimating the parameters characterizing the shape of the time-dependent curve. Alarms result from extrapolations of such curves and from comparisons with fixed or variable thresholds. Trended data allows experts (expert systems) to make an intelligent decision on how much longer a failing component, or assembly, may be operated safely.

Since trend analysis relies heavily on historical data, the ability to store and retrieve data quickly is critical, particularly in space applications.

How much data should be analyzed is also important because there is little point in amassing so much data on a particular piece of equipment that retrieval becomes a major problem. There are several approaches that will continue to be evaluated at and beyond IOC:

1. Updating by exception - In this approach, after the baseline data have been verified over a period of time, the monitoring function continues but new data are not added unless it is at variance with the previous information.

2. Discarding old data - This approach is to save all new data and to discard or average away old readings. This procedure is currently favored and can easily be implemented on a computer-controlled diagnostic system.

Many techniques may be used to detect trends: slope computations, inflection point checks, least-square curve fitting, and state estimation methods. It is desirable, however, especially if large amounts of equipment are being monitored to maintain a low computation burden. For this reason, the simpler computational algorithms, such as the least-squares approach, are preferred options. Generally, a

ORIGINAL PAGE IS
OF POOR QUALITY

curved or sloping line indicates a gradual degradation with time, and scattered data outside the expected distribution indicate a failure.

Other techniques, such as the minimal least-squares estimator and the Kalman-Bucy filter, which can be used to enhance or magnify the detectability of failure modes, are being investigated at Rocketdyne for their potential use in the EPS. They may prove to be powerful tools that can be implemented within the current design framework. But, no matter which methods are determined to be applicable, to respond to the anticipated Space Station maintainability requirements, collection and analysis of trend data will be implemented to the fullest degree feasible.

Conclusions

The importance of reliability, stability, and automatic operation of the electrical power system for all other subsystems on the Space Station makes it a prime candidate for the application of artificial intelligence techniques. Analysis of crew and ground-based resources have emphasized the need for advanced automatic control of the EPS at IOC, with an advisory diagnostic capability in an expert system for trend analysis, fault prediction, and component maintenance. Research at Rocketdyne and NASA/LeRC is proceeding in all of these areas to determine the optimum hardware and software implementations for such functions. The results of these efforts could do much to enhance the productivity of the Space Station as well as future manned space activities.

References

1. Pau, L. F., DTH, Rormosen 56, Kaarup DK 4540 Faarevejle, Denmark, "Survey of Expert Systems for Fault Detection, Test Generation and Maintenance," Expert Systems, The International Journal of Knowledge Engineering, Vol. 3, No. 2, April 1986.

2. Hendelman, D. A. and R. F. Stengel, Princeton University, Department of Mechanical & Aerospace Engineering, "Combining Quantitative and Qualitative Reasoning in Aircraft Failure Diagnosis," AIAA 85-1905, pp. 366-376.

3. Fink, P. K., J. C. Lusth, and J. W. Duran, "A General Expert System Design for Diagnostic Problem Solving," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 5, pp. 553-560, September 1985.

4. De Kleer, Johan, Xerox PARC, Intelligent Systems Laboratory, Palo Alto, CA,"How Circuits Work," Artificial Intelligence 24, pp. 205-280, 1984.

5. Stengel, R. F., "Artificial Intelligence Theory and Reconfigurable Control Systems," Princeton University, 30 June 1985.

6. Leinweber, D., "Real-Time Expert Systems for Space Station Process Control," LISP Machine, Inc., Los Angeles, California,."

7. Malik, J. and T. Lance, "An Expert System for Fault Management and Automatic Shutdown Avoidance in a Regenerative Life Support Subsystem," Paper #85-0333, ISA 1985.

8. Divita, Turner, P. R., "Autonomous Spacecraft Design Methodology," Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California.

9. Adams, T. L., B. D'Ambrosio, M. R. Fehling, and S. Schwartzberg, "An Artificial Intelligence Approach to Autonomous Power System Maintenance and Control," 1985 Society of Automotive Engineers, Inc.

10. Kusic, G. L., Computer-Aided Power System Analysis, Prentice-Hall, New Jersey, pp. 347-368, 1986.

11. Hirsch, Abraham, "Tagged Architecture Supports Symbolic Processing," Computer Design, pp. 75-78 and 80, June 1984.

12. Kraus, T. W. and T. J. Myron, "Self-Tuning PID Controller Uses Pattern Recognition Approach," Control Engineering, June 1984.

13. Klos, L. C., J. A. Edwards, and J. A. Davis, "Artificial Intelligence—An Implementation Approach for Advanced Avionics," AIAA 83-2401, pp. 300-307.

14. NASA Technical Memorandum 84766, <u>Advancing Automation and Robotics Technology for the Space Station and for the U.S. Economy, Volume I - Executive Overview</u>, Advanced Technology Advisory Committee, Submitted to United States Congress, 1 April 1985.

15. Zimmerman, W. F., J. Bard, and A. Feinberg, "Space Station Man-Machine Automation Trade-off Analysis," JPL Publication 85-13, NASA Jet Propulsion Laboratory, California Institute of Technology.

16. Stevenson, William D., <u>Elements of Power System Analysis</u>, McGraw-Hill, 1982, pp. 373-421.

17. NASA Johnson Space Flight Center, Minutes of Meeting, Space Station Program Level B SIB, 23 April 1986, p. 7.

18. Pau, L. F., "Failure Diagnosis and Performance Monitoring," <u>Control and Systems Theory</u>, Vol. 11, Marcel Dekker, Inc.

662

A METHODOLOGY FOR MULTIPLE RULE SYSTEM INTEGRATION

AND RESOLUTION WITHIN A SINGULAR KNOWLEDGE BASE

Frank N. Kautzmann, III, Ph.D.
MITRE Corporation
1120 NASA Road 1
Houston, Texas 77058

Topics: Philosophical and Scientific Foundations,
AI Architectures and Languages, Knowledge
Representation, Mathematical Logic, Proof
Theory, Program Verification, Set Theory

ABSTRACT

Expert Systems which support knowledge representation by
qualitative modelling techniques experience problems, when called
upon to support integrated views embodying description and
explanation, especially when other factors such as multiple
causality, competing rule model resolution, and multiple uses of
knowledge representation are included. MITRE Corporation, in
conjunction with several directorates of NASA at Johnson Space
Center, is currently developing a series of prototypes to
demonstrate the feasibility of automating the process of systems
engineering, design and configuration, and diagnosis and fault
management. It is important to study these processes as they
pertain to the analysis of design trade-off issues, knowledge
capture and Space Station planning and assessment. The MITRE
effort will involve not only a generic knowledge representation;
it must also support multiple views at varying levels of
description and interaction between physical elements, systems,
and subsystems. Moreover, it will involve models of description
and explanation for each level. This multiple-model feature
requires the development of control methods between rule systems
and heuristics on a meta-level for each expert system involved in
an integrated and larger class of expert system. This paper
describes the broadest possible category of interacting expert
systems and proposes a general methodology for the knowledge
representation and control of mutually exclusive rule systems
within the context of a single knowledge representation scheme.

INTRODUCTION

NASA has been directed by Public Law 98-371 to automate, as much as technically and economically feasible, appropriate aspects of space flight. This agency-wide charter necessarily involves the investigation of contemporary AI processes and methods. The generic focus is primarily on the planned Space Station, although current Space Shuttle operations evaluation is relevant due to the wealth of expert knowledge already in place. The majority of tasks that are candidates for automation involve expert knowledge. The sub-field of expert systems within AI can thus be seen as a fruitful and primary methodology for investigation. Secondarily, it is a relatively small step to determine that prototypes for each separable expert system will be required. There are two reasons for this conjecture. First, prototypes are excellent pragmatic mechanisms for the demonstration of feasibility. Second, because of the necessary interaction with domain experts in prototyping an expert system, expert knowledge is translated into a form that is formalized for future examination and use as expressed in the form of rules and logical structures. This benefit is important due to a current concern within NASA and industry about the loss of retiring experts' knowledge. By moving from an expert's knowledge to a reproducible form of expressing that knowledge, such fears are alleviated. Examples of candidate knowledge areas to capture within NASA will include design engineering, systems engineering, fault diagnosis, testing and configuration management. A current AI project within MITRE addresses the design engineering knowledge capture process utilizing an amalgam of CAD/CAM and expert systems technology [1].

It is reasonable to conjecture that separable, standalone single domain expert systems are but an initial phase along the process of providing, say, a Systems Engineering Assistant. That such an Assistant will necessarily embody and interact with the single-domain expert systems at various levels is not to be overlooked as a logical consequence. By further extension of the preliminary conjecture, it can be seen that there are at least three possible areas in which the development of such a class of "Assistant" expert systems would be useful:

. Systems Engineering
. Design and Configuration
. Diagnosis
  - Fault Management
  - Testing

Assistant-type experts in each of the three areas will require interaction with separate single-domain expert systems for two reasons. First, because each of the three areas must also provide facilities for simulation and modelling. Second, because knowledge about how to perform satisfactorily involves knowledge concerning the inter-relationships between expert domains. Thus, from within a Space Station context, each of the three areas must share elements and interact with the following ten major single-domain expert systems:

. Communications & Tracking
. Data Management System
. ECLS - Environmental Control and Life
  Support
. Electrical Power
. Guidance and Navigation
. Man
. Propulsion
. Thermal
. Structure
. Payloads

The ten systems have further taxonomy at the sub-system and component level. Additionally, there are multiple cross-links between subsystems and components at the juncture of components and subsystem interactions [Figure 4]. The former three areas and the separable single domain expert systems must interact on the boundaries of overall system performance as an integrated expert system, as noted. At a minimum, each of the ten single domain expert systems must share a common knowledge representation with the larger class of expert systems and with each other.

PROBLEM STATEMENT

The initial problem statement for the success of the larger "Assistant" class of expert systems involves specifying the current tasks and goals of the three major functional areas. From this point onward, "Expert System" shall refer to an expert system that embodies the performance objectives of the three major areas. "System" shall refer to smaller domain-specific expert systems with "subsystem" referring to elements and components of the specific local system.

The major goal of each expert system may be viewed as determining the performance of the complete system, subsystem or element in some given configuration and state, given specified inputs and specified outputs. Each expert system's task area is the determination of system performance with differing perspectives or views.

665

● In Systems Engineering activities, the goal is the determination of precisely how and under what conditions the subsystems and the parts can operate together to produce, preserve, or achieve some desired outputs or conditions of the overall system. Thus a methodology is required, along with a knowledge representation, that can be combined with various models of behavior to simulate alternate system configurations; thereby determining how these alternate configurations respond to various inputs and demand constraints. Secondly, an analysis of the interfaces, interactions, or connections among the subsystems, with particular emphasis on problem identification and constraint violations at the interface level, is required.

● In the "configuration" domain of the Design and Configuration activities, the goal is the determination of whether or not it is possible to structure or configure a system in such a fashion that it preserves or achieves a desired set of outputs or conditions. This determination must be made within given resource constraints, such as time or power, along with expected inputs and demands. Again, there is a need to model and simulate possible configurations with varying inputs and resources. The solution space relative to resources may be less constrained than that of the Systems Engineering domain. The area of interest and emphasis is on internal couplings or structures relative to alternative resources for achieving a given structure or series of system goals. The "design" domain of the Design and Configuration activities involves planning for a correction to a configuration, a repair, or fault-tolerating action relative to a system, subsystem element or alternative(s).

● In Diagnosis - Fault Management activities, the goal is the determination of the precise configuration, including fault modes, with appropriate inputs or conditions, that could produce a given set of outputs or conditions. The Fault Detection emphasis is on determining configuration states that elicit a fault or a series of fault conditions, and then isolating the respective faulty component states along with their state-dependent interactions and transfer functions. Corrections are actions on the system, operating in con-text, that permit changing either its state or its response in some predicted fashion, given that the assumptions about the system's prior configuration are correct.

● In Diagnosis - Testing activities, the goal is the generation and validation of test hypotheses wherein a system response is predicted based upon changing configurations, which are themselves based upon conditional inputs or states. The tests are actions on the system to evaluate system response relative to

conditional, unknown or hypothetical prior conditions or states. Predicted sets of expectations as to system behavior/response are also alternative sets of expectations that are hypothesized in order to effectively perform a differentiating test relative to actual responses.

Common Elements

The common elements of the three performance goals require movement from a given current configuration to a desired configuration (or alternate) that meets specifications, goals, or needs relative to a higher goal or to some hierarchy of goals, inputs or states.

The integrated expert system must accommodate the description, determination, design and analysis of system, subsystem and component configurations relative to constraints and goals specified either as inputs or outputs. This involves a knowledge representation that supports the above modelling and simulation activities, each with it's respective different emphasis. This requirement further decomposes into the requirement for a knowledge representation that accommodates these features:

● System, subsystem, and components must change state and internal configuration. This requires some transfer function to define such changes across boundaries; and also within a hierarchy of descriptions and explanations.

● The transfer functions must be able to characterize the internal processes of the system, subsystem, or component, given a particular state or confiuration of the same.

● Additional functions are necessary to describe coupling, direction or interactions that can characterize the constraints, parameters, or specifications on the required inputs and outputs and their respective directions of coupling.

● Goals of the system, subsystem, or components must be translated from and between all elements relative to a system, subsystem, or component state/configuration. Determination of side effects and sub-goal decomposition is also required.

GENERAL APPROACH

The conjectured task naturally divides into two phases. The initial phase will involve the construction of preliminary single-domain expert system prototypes in the ten areas already noted along tradi- tional lines. The non-traditional aspect of

the first phase would involve the designing and utilization of a knowledge representation scheme that is capable, in principle, of being extended for future single-domain expert systems and sub-systems. The degree of difficulty in performing the first task is minimal, although it may be time consuming. It should also be noted that rapid prototyping schemes with alternate knowledge representations would inhibit the success of this phase. If, for example, one selects a knowledge representation scheme that works for but one of the ten single-domain expert systems, then all others to that point must be redone, or enlarged to accommodate the variance. Thus, success for this phase is requires initial selection of a valid knowledge representation scheme in order to avoid permutational reworking for each separable system. This step requires little else but good systems engineering practices and data abstraction in the selection process.

The second phase also has requirements for knowledge representation at a higher level, since it must involve cross-domain expert system interaction at subsystem and component levels and presupposes an initial knowledge representation adequate to support such a functional requirement. Since the generation of multiple single-domain expert systems has been done many times before, this paper concentrates on the second aspect of the conjectured prototype and proposes a suggested methodology for achieving cross-domain expert system interaction.

This second phase's primary objective involves the generation of a single methodology to achieve interaction among closely related cross-domain separable expert systems with multiple views, models, performance requirements, and transforming operations capable of operating at multiple levels of description and explanation. Achieving this requires, in part, adopting forms of knowledge representation that can operate at multiple meta-levels. These forms must be, beyond object-oriented semantic and syntactic specific, of a sufficient formal and mathematical nature in order to support the defined problem space transformational requirements. This further requires that the knowledge representation and the transforming processes themselves be valid. 'Validity' involves four formal requirements: consistency, completeness, soundness, and precision. 'Consistency' requires that similar answers produce similar results. 'Completeness' requires that everything true is derivable. 'Soundness' requires that everything derivable is true. 'Precision' is required within ES's that deliver probabilistic or qualified judgements. The ability of the prototype methodology and knowledge representation to meet the criteria of validity enables, in principle, the completed system to be transformed vis-à-vis alternate methods, to other representations that are equally valid across related domains as well as within the single domain.

Background to Approach

A strong parallel exists between automated program verification and specification techniques and single-domain expert systems. At the lowest level of parallel, there are merely linguistic and data typing parallels. Within the mathematical and logical foundations the parallels are more striking and offer an insight into cross-domain expert systems interaction. That the higher class of expert systems outlined herein, share the same elements with single-domain expert systems is also true by extension. That the parallel claim is the case is evidenced by Figure 1. The further language-based parallel involving syntax and semantics between the two areas is also apparent. So are the formal and philosophical foundations which involve mainly mathematical logic(s) and set theory. The long tradition in philosophy of a distinction between "semantic" and "syntactic" in many areas of inquiry does not necessarily relate, except within a limited domain, to a linguistic theory of meaning. That the linguistic model, along with further investigations in "natural language" processing, is pre-eminent in expert systems research is self-evident. That the linguistic focus is useful is not in dispute for knowledge capture. The extension of such a model throughout expert systems may account for the lack of acknowledging the parallel domains in formal systems, with their own syntactic and semantic issues, by expert system researchers.

That this parallel has not yet been either drawn explicitly or extensively investigated before for either single-domain expert systems, and most importantly, for expert system cross-domain interaction, is evident by the conspicuous absence to such parallels within contemporary AI expert system's literature and journals. [The closest parallel to date is between automatic theorem proving techniques used as a heuristic method within expert systems].

The reasons for this absence are possibly manifold. It is tentatively suggested that the absence is accounted for by the "natural language" processing model for mind that preempts other models by AI researchers. Likewise, non-Philosophers, namely cognitive psychologists, linguists, anthropologists, and to a certain extent, computer scientists, generally maintain a belief that the expert system knowledge acquisition process, as a process, offers a window to the mind. Noam Chomsky's work in linguistics has offered a model of species differentiae between man and the brutes. The salient feature is seen as centered upon man's inherent ability to acquire natural language capabilities. Ergo, this becomes the ontological basis for the central focus of the "natural language" processing theory of mind in expert system

research and cognitive psychology. This model also embodies the underlying and deeper, philosophically, linguistic theories of meaning.

Further, AI research in expert systems focuses upon confirmation of (competing) natural language acquisition models in order to further substantiate in turn, deeper models of "reasoning". These deeper theories in turn rest upon related hypotheses concerning the nature of mind. The epistemological bases in turn refer to underlying metaphysical theories of existence, structure and causality. The further foundational appeal is within Philosophy of Science and to a general model theory of meaning expressed as a linguistic model.

That there are important underlying similarities with the philosophical, logical and mathematical nature of expert systems themselves and the argued parallel for possible future progress in cross-domain expert system interaction based upon automatic program verification and theorem proving techniques warrants further in-depth investigation by AI.

An additional reason why the fact that this parallel has not been drawn before by expert system AI researchers can be possibly accounted for by the recharacterization of the formal methods that have yielded results in the field of expert systems into a form that is foundationally seductive to AI researchers for their broader scope of investigation of human reasoning.

To wit, "Forward chaining", "Backward chaining", or "Forward reasoning" and "Backward reasoning" mean nothing more than the utilization of the predicate logic techniques of modus ponens and modus tollens.

For example, every single-domain expert system "inference engine" is but a separate instance of modus ponens used as a control structure invoking other instances of either modus tollens or modus ponens operating at the rule level, which in turn, operates using variable assignments at the object level of description for binding. That there exists a fundamental contradiction when these facts are conjoined with claims from the same re-searchers that formal methods yield no promise for cross-domain expert system interaction is obvious. It is suggested here that what is amiss is the underlying, unstated premise that only confirmation of theories, in some model building or paradigmatic sense, related to human-centered reasoning models of explanation, are use-ful for the stated purposes of AI. That a strong parallel exists between the functioning underlying structures of formal logical methods, mathematical logic and set theory and the domain of actually working single-domain expert systems is not to be

refuted. Expert systems work precisely because of this continuity and dependence upon the underlying mathematical-logical structures. This is particularly true when such systems "reason about themselves" in some meta-level fashion such as exemplified in qualitative modelling.

From a differing perspective, the absence of such an explicit parallel by research- ers in theorem proving and program veri-fication to expert systems might be based upon the recognition by the former re-searcher's of the domain expert's inter-action in generating an object-oriented, for example, semantic domain(s). Or alternatively, precise recognition that verification and theorem proving deal with an inherently mathematical structure and thus lack the linguistic model confirmational orientation of expert systems.

The parallels operating from the domain expert's knowledge, once it has been captured and formalized (logically) into rules, into an object-oriented domain, which can then be operated on syntactically, will be developed from the perspective of similar formal parallels in automated program verification and theorem proving. That such can be applied to expert system's cross-domain interaction will be further exemplified from the argued methodological premise.

As a preliminary in this direction, Figure 1 is offered as an exemplification of all single-domain expert systems, and by ex-tension, to cross-domain expert systems. Methods of search and knowledge-based techniques for reasoning and the representation of knowledge are addressed. The figure is particularly important relative to the proposed general methodology. The Philosophical foundations are presented as a map to ascertain the relative position of issues and methodologies referenced within expert systems.

Outline of Approach

The initial approach for the conjectured prototype utilizes a variety of models of description and explanation over a wide variety of types. These include logical models, analogue models, semi-formal or mathematical models, simplifying models, and theoretical models as seen in Figure 4. In order to support this wide variety, multiple versions of declarative encoding for knowledge representation were be selected. In declarative encoding semantics are used in a few global procedures. By formally transforming between levels of descriptions and explanations (meanings), the problem of different domains will be reduced to formal representation and transformation at a meta-level. A fruitful first step is in selecting a knowledge

671

representation method that will ground the proposed mathodology within the realms of logic and mathematics. Semantic nets are ideal for this purpose. There are two reasons for selecting semantic nets for initial knowledge representation:

. Semantic networks are powerful. Proof exists that they are Turing-complete [2]; thus methods in finite automata are available for future use.

. Rapid prototyping of alternate knowledge schemas is permitted [3]. (This is a side benefit and not central to the proposed expert system methodology, except in the cases of the underlying single-domain expert systems).

Rule System Representation

Once the expert's knowledge has been captured in the form of rules, trans-formational consistency demands that the rule system's formal representation be also Turing-complete. LISP is known to be Turing complete and will be selected as the language of choice. The general method of rule application/selection for heuristics follows 'production rule systems', a formalism proposed by Emil Post in 1943. ["Heuristics" in this sense refers to control methods, not to special meta-rules within the rule classes]. That production rule systems are also known to be Turing-complete has been established [4]-[6]. Knowledge representation, rule writing, and heuristics thus all share, at a minimum, the computationally powerful and sufficient aspects of Turing-completeness, including recursion.

In order to achieve this common aspect, embedded rule systems in commercial AI development tools need to be avoided unless they are provably known to be extensible and complete with respect to the Lambda Calculus and also contain a definitive pathway to LISP. [That the Lambda Calculus extension is also a requirement fits within contemporary research in program verification, as elsewhere referenced].

A further advantage of using production systems for rule representation is that this method does not rule out the possibility and speed advantages of later transformation using procedural encoding following the proposed methodology. Production systems provide an effective bridge between the declarative and procedural encoding methods. It is by means of this bridge that future migration of an existing expert system so designed, can migrate cross-language [and cross-hardware], as will be outlined.

Initial Knowledge Architecture

A conceptual prototype representational scheme is given as an overview in Figure 4. In the figure the models, components, systems and subsystems are objects. The principles utilized across models refer to methods of transformation in a hierarchy of analogical pattern-based series of meta-rules [7]. The constraints on models are also pattern-based at the meta-level and object-oriented at the primary description level. Goals relate to inputs, outputs, states and/or conditions. State-history relates to operational status of the component or subsystem and is a tree-oriented directed acyclic graph, or an alternative abstract algebra state machine [8].

Pattern Matching

The models, as identified in Figure 4, are called based upon either states, inputs, outputs, goals or conditions. It is important to recognize the element of analogical pattern-matching and reasoning for the models' selection and its application.

Furthermore, the various types of 'models', (for example, qualitative models), have an algebra of operators for single-point causal analysis and sequencing. That such a single-point causal model is not sufficiently rich enough to address all issues arising from cross-domain expert systems forces the requirement for additional model types. Other models of causality may, for example, be required. However, all model's operators form, at a higher level, a unique syntax and semantic for later transformation. They are formal structures, in and of themselves, apart from the initial object-oriented domain and form the basis for valid sets of transformational operators in each domain.

Abstractly, what is most important are the patterns of reasoning applied within a given rule system or domain in terms of successful heuristics. Patterns of reasoning are themselves formal structures; such structures are capable of examination by a wide variety of formal mathematical-logical methods.

These formal structures are symbolic in that they permit more than purely formal deduction. It is important to note that there are two levels of reasoning involved. The first, with respect to the actual knowledge domain, and the second with respect to the transformational aspects of that same knowledge into forms acceptable for all levels and for all views.

This second aspect of reasoning requires the development of problem-solving procedures which take into account the robustness

of the separate rule systems and heuristics. This requirement places a further restriction on the development of problem-solving procedures; they must guarantee the mechanization of formal cross-domain proof procedures [9]-[11].

Satisfaction of the proof procedure requirement involves devising effective problem-solving methods based on the structure of the reasoning model, and the associated system of knowledge representation. The ease with which it is possible to put such methods to work will depend upon how systematically the formal properties of each reasoning model's domain has been analyzed.

SPECIFIC APPROACH

The general problem space of the Space Station prototype relates to natural language domains of expertise and also the more mathematical-logical aspects of formal languages and semantics as seen in Figure 2.

Knowledge representation, using semantic nets and declarative encoding of rules within domains, with common tree-oriented data structures, permits capturing the patterns of success involved in each state or condition, as well as the rules and heuristics. This permits specification of the patterns, and the associated facts in the data structures, to be modelled as proof-assertions, as seen in Figure 2.

Figure 3 outlines a general methodology for the transformation between logical systems while preserving the semantics of each expert system domain. The stylized diagram involves the use of transformational grammar terminology to describe levels of syntactic processing, although transformational formalisms are not used to define the processing. Source text, produced by a single-domain or otherwise, expert system, either as output or input, via the proper logical system, is concrete syntax. Parsed concrete syntax is termed surface abstract syntax following surface structures in transformational grammars. These translations are context-free; i.e., variables are still variables in the same order following these two steps. Surface abstract syntax is transformed in non-context-free fashion into deep abstract syntax. As is required in the process of semantic analysis, each declaration of symbols and variables is processed and every occurrence of a symbol is identified with the information in its declaration. Logical transformations are performed as structural transformations. [12]

Techniques for reasoning by analogy vis-a-vis proof-theoretic techniques should permit aspects of common-sense reasoning to apply at the knowledge based object levels in conjunction with

validation of cross-domain proof-assertions. This would involve the development of equivalent means of traversing both the rule base and the object attribute-values, as outlined.

The integration of cross-domain knowledge in the form of meta-level rules is viewed as an investigation into domain-specific syntactic and semantic language transformations, involving higher-order logics, such as combinatory logic, or the S-K combinator calculus for the meta-level representation of heuristics and rules [13]-[16].

The formal proof-theoretic aspects of this level of formalism can, in principle, accommodate specification and verification, design validation, and generate search methods and 'knowledge-based' reasoning about oneself in closely related knowledge domains automatically. Fault detection and analysis can be performed at a higher level than the domain level, given that the relationships between elements as behavior rules can be formalized and transformed, using analogies of function and structure.

Updates to states can be viewed as tree-insertions in a derived search/resolution graph of the actual domain specific rules and heuristics. Traversing the graph, then transforming the sequence into the rule system domain, permits viewing the overall system according to interest, intent or relevant model.

Obtaining these results involves creating a separate proof-system for the transformational rules, patterns and sequences, taking advantage of the formal-mathe-matical nature of the prototype's semantic nets and production rule systems in the form of a high level language analyzer and interpreter.

In particular, understanding logical-con-sequence semantics does not require the construction of specialized models in lattice theory or category theory [17]. Nor does it involve, as is the case with abstract data types, initial or final algebras [18].

Specifically, a rule system, for a given state, given inputs, outputs, goals, plus interactions or couplings, can be viewed as a given set of assertions akin to a declarative program. The logical consequences of such a set of assertions are all the additional assertions that must be true whenever the assertions of the rules are true. Strictly speaking, an assertion in the form of A = B can be seen as a logical consequence of a set F of equations or transformational operators if and only if, in every algebraic interpretation for which every equation in F is true, A = B is also true. This can be verified at the object level in each knowledge domain.

There are a wide variety of logical languages that have be used to write assertions of rules, conditions and states, such as the first order predicate calculus as implemented by PROLOG. Such logical (formal) language constructs as output, may be combined with other assertions, to assist in non-context-free solutions or verifications of prior states or conditions with associated trees and mappings reducing the search space. It is possible in principle to utilize such a language of transforms to assist in rule verification at the domain level.

Meta-rules are also non-context-free. It is possible to relate the patterns that are embedded in these meta-rules in a context-free implementation as meta-level heuristics and rules. The danger of using the meta-rules themselves as search/resolution mechanisms must not be over-looked. Their use is limited to assisting transformations between domains using the formalisms that are inter-derivable and reducible to the domain-specific rules as forms of proof mechanization techniques. These techniques themselves provide the firing sequence (heuristics) of domain specific rules as meta-level heuristics.

SUMMARY AND CONCLUSIONS

A methodology is being investigated for cross-domain interaction between related expert systems utilizing a single knowledge representation. The methodology utilizes various formal and logical techniques for examining rule systems and associated object representations.

A methodology for deriving states, conditions and transforms between levels of behavior and function involves utilizing the Turing-complete aspects of semantic nets and production systems of a formal and logical-mathematical nature at several degrees of abstraction above each sub-system, using the meta-level approach is suggested as fruitful for cross-domain expert system interaction resolution. That such an approach also holds promise as a general methodology for the addition of rules into existing single-domain expert systems and the attendant modification of inference engines is also noted.

It is anticipated that the best foundation for a symbolic computing formalism is probably a layered language approach, each layer containing a subset of symbols that produce all of the desired determinate computations and transformations, and something from the S-K combinator calculus [19] to be used in the infrequent cases where analysis of truly indeterminate behavior is required.

That there may be other layers of disciplined behavior that should be covered by simple sub-languages, rules and associated graphs that produce the desired associated transforms with an attendant search/solution sequence, as part of the proof-procedure of validation at higher levels of abstraction, is admitted.
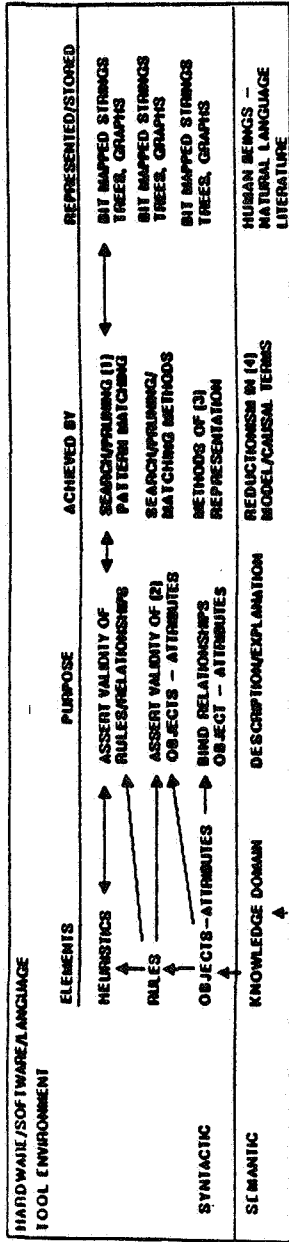
ACKNOWLEDGEMENT

REFERENCES

(1)   Crouse, K., and Spitzer, J., "Design Knowledge Bases for the Space Station", Robotics and Expert Systems, 1986, Proceedings of ROBEXS '86, Sponsored by Instrument Society of American and ISA's Clear Lake-Galveston Section, June 5-6, 1986.

(2)   Nilsson, N., Principles of Artificial Intelligence, Springer-Verlag, 1982.

(3)   Hayes-Roth, F., Watterman, D. A., and Lenat, D. B., Building Expert Systems, Addison-Wesley, 1983.

(4)   Post, Emil L., "Introduction to a General Theory of Elementary Propositions", American Journal of Mathematics, Vol. 43, pp. 163-185, 1921.

(5)   Post, Emil L., "Formal Reductions of the General Combinatorial Decision Problem", American Journal of Mathematics, Vol. 65, 1943.

(6)   Anderson, J., Memory and Thought, Erlbaum Associates, 1976.

(7)   Chouraqi, E., "Construction of a Model for Reasoning by Analogy", Proceedings of ECAI-82, Orsay, France, 1982.

(8)   Scott, D. S., and Strachey, C., "Toward a Mathematical Semantics for Computer Languages", Proc. Symp. on Computers and Automata, Polytechnic Institute of Brooklyn, Vol. 21, pp. 19-46, 1971.

(9)   Gentzen, G., "Untersuchungen uber das logische Schliessen", Mathematische Zeitschrift, Vol. 39, pp. 176-210, 405-431, 1934-1935.

(10) Ajsukiewicz, K., "Classification des Raisonments", <u>Studia Logica II</u>, Warsaw, Poland, 1955.

(11) <u>Logic, Form and Function: The Mechanization of Deductive Reasoning</u>, Elsevier North-Holland, 1979.

(12) O'Donnell, Michael J., <u>Equational Logic as a Programming Language</u>, MIT Press, Cambridge, Massachusetts, 1985.

(13) Giannini, P., and Longo, G., "Effectively Given Domains and Lambda Calculus Semantics", Preprint, <u>Dipt. Informatica</u>, Corso Italia, Vol. 40, 56100 Pisa, Italy.

(14) Cohen, L. J., Pfeifer, H., and Podewski, K. P., (eds), <u>Logic, Methodology and Philosophy of Science VI</u>, North Holland, Amesterdam, 1982.

(15) Suppes, P., et al., <u>Logic, Methodology and Philosophy of Science IV</u>, Studies in Logic 74, North-Holland, Amsterdam, 1973.

(16) "Combinatory Logic as a Semigroup", abstract, <u>Bulletin American Mathematics Soc.</u>, Vol. 43, pp. 333, 1937.

(17) Terlouw, J., "On Definition Trees of Ordinal Recursive Functionals: Reduction of the Recursion Orders by Means of Type Level Raisings", <u>J. Symbolic Logic</u>, Vol. 47, pp. 395 403, 1982.

(18) Stenlund, S., <u>Combinators, Lambda-Terms, and Proof Theory</u>, D. Reidel Publishing Company, Dordrecht, Holland, 1972.

(19) Goguen, J. A., "Abstract Errors for Abstract Data Types", <u>IFIP Working Conference on Formal Description of Programming Concepts</u>, E. J. Neuhold (ed), North-Holland, 1977.

(20) Clarke, T. J., Gladstone, P. J. S., MacLean, C. D., and Norman, A. C., "SKIM-The S,K,I Reduction Machine", <u>1980 LISP Conference</u>, Stanford University, pp. 128-135, 1980.

FIGURE 1 EXPERT SYSTEMS ABSTRACT REPRESENTATION

HARDWARE/SOFTWARE/LANGUAGE
TOOL ENVIRONMENT

	ELEMENTS	PURPOSE	ACHIEVED BY	REPRESENTED/STORED
EXPERT SYSTEM	HEURISTICS	ASSERT VALIDITY OF RULES/RELATIONSHIPS	SEARCH/PRUNING (1) PATTERN MATCHING	BIT MAPPED STRINGS TREES, GRAPHS
	RULES	ASSERT VALIDITY OF (2) OBJECTS - ATTRIBUTES	SEARCH/PRUNING/ MATCHING METHODS	BIT MAPPED STRINGS TREES, GRAPHS
SYNTACTIC	OBJECTS-ATTRIBUTES	BIND RELATIONSHIPS OBJECT - ATTRIBUTES	METHODS OF (3) REPRESENTATION	BIT MAPPED STRINGS TREES, GRAPHS
SEMANTIC	KNOWLEDGE DOMAIN	DESCRIPTION/EXPLANATION	REDUCTIONISM IN (4) MODEL/CAUSAL TERMS	HUMAN BEINGS - NATURAL LANGUAGE LITERATURE

PHILOSOPHICAL FOUNDATIONS

METAPHYSICS

INQUIRY INTO THEORIES OF:
- EXISTENCE, THING PROPERTY, EVENT
- MIND/MATTER/ SPACE/TIME
- CAUSATION/CHANGE
- PARTICULARS/ UNIVERSALS

EPISTEMOLOGY
- NATURE/SCOPE OF KNOWLEDGE
- PRESUPPOSITIONS AND BASES
- RELIABILITY OF CLAIMS OF KNOWLEDGE

METHODS OF KNOWLEDGE
VALIDATION - TRUTH/ MEANING

LOGIC/MATHEMATICAL LOGIC

FORMAL SYSTEMS
- PROOF THEORY
-- SET THEORY
-- MATHEMATICS

FORMAL METHODS
- DEDUCTION
- INDUCTION

THEORIES OF TRUTH
- CORRESPONDENCE
- COHERENCE
- PRAGMATIC
- SEMANTIC
- PERFORMATIVE
- EXISTENCE

THEORIES OF MEANING
- REFERENTIAL
-- MEANING AND REFERENCE
-- REFERENCE AND SEMANTIC RELATIONS
- IDEATIONAL
- STIMULUS-RESPONSE
- NATURAL LANGUAGE
-- MEANING - USE
-- MEANING AND LINGUISTIC ACTS
-- USES OF WORDS
-- RULES AND REGULARITIES
-- CRITERIA OF SYNONYMY

(4) SCIENCE
- PHYSICS
- PSYCHOLOGY

(1) METHODS OF SEARCH
- TOP-DOWN, BOTTOM-UP.
- TREE/GRAPH TRAVERSALS FLATTENED TREE
- VARIANTS OF DEPTH VS. BREADTH

(2) METHODS OF REASONING
- CLASSICAL LOGIC
- NON-CLASSICAL LOGICS
- PLAN GENERATION
- NON-MONOTONIC AND COMMON-SENSE REASONING
- PATTERN-DIRECTED INFERENCE

(3) METHODS OF REPRESENTATION
- SEMANTIC NETS
- PRODUCTION SYSTEMS
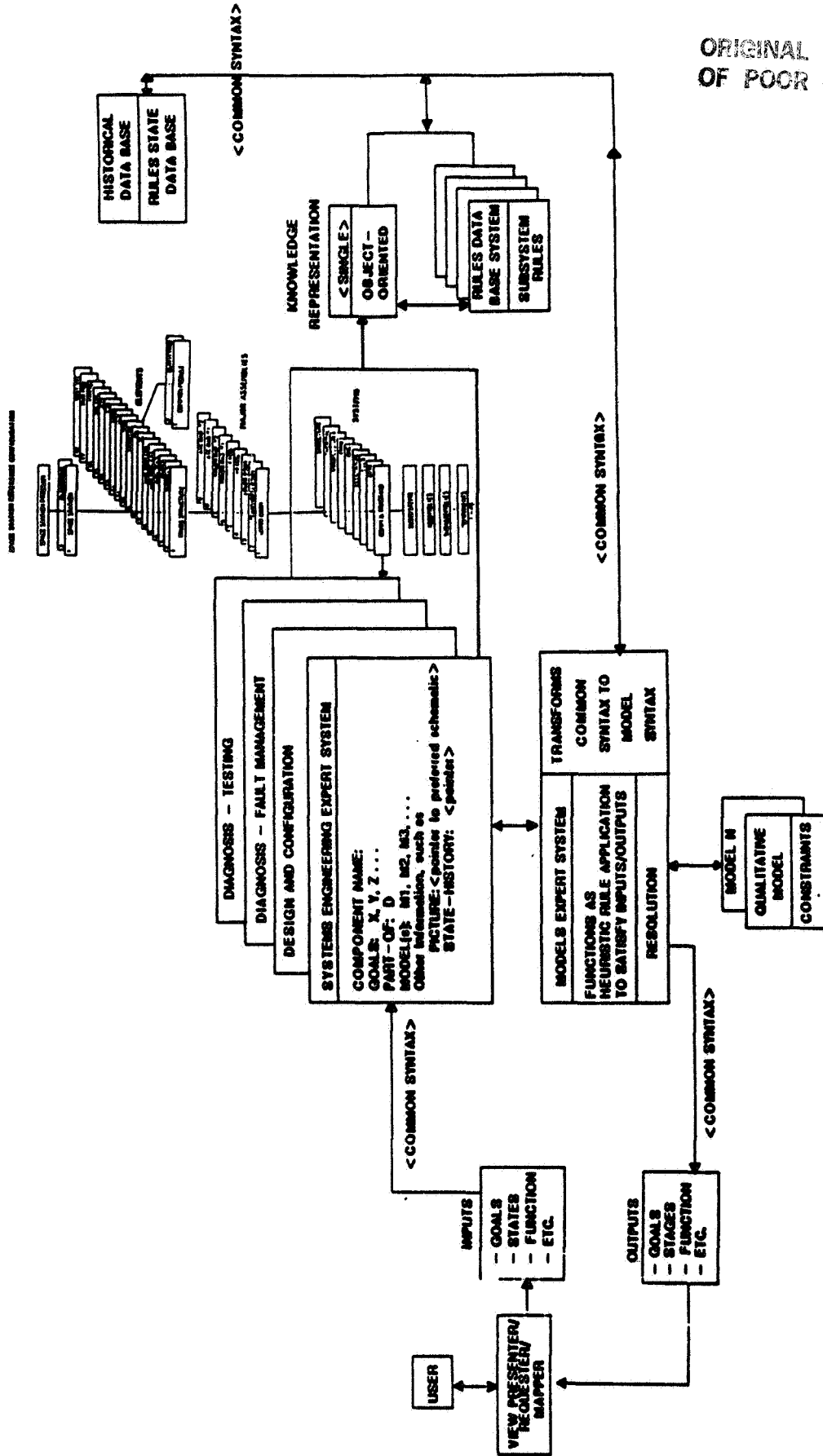- ANALOGUE REPRESENTATIONS
- FRAMES
- SEMANTIC TRIPLES, ETC.

FIGURE 1   META-LEVEL LANGUAGE TRANSFORMATION

META-LEVEL
TRANSFORMATIONS
CONTEXT-FREE

FIGURE 2   ASSOCIATED META-LEVEL TRANSFORMATION

FIGURE 4   PROTOTYPE EXPERT SYSTEMS OVERVIEW

Blackboard Architectures and Their Relationship to Autonomous Space Systems

Allison Thornbrugh

Martin Marietta Denver Aerospace
P.O. Box 179
Denver, Colorado 80201

Abstract

The blackboard architecture provides a powerful paradigm for the autonomy expected in future spaceborne systems, especially SDI and Space Station. Autonomous systems will require skill in both the classic task of information analysis and the newer tasks of decision-making, planning and system control. Successful blackboard systems have been built to deal with each of these tasks separately --- data fusion in speech understanding [Erman81] and planning of errands with OPM [Hayes-Roth79]. The blackboard paradigm achieves success in difficult domains through its ability to integrate several uncertain sources of knowledge.

In addition to flexible behavior during autonomous operation, the system must also be capable of incrementally growing from semi- autonomy to full autonomy. The blackboard structure allows this development. This paper will discuss the blackboard's ability to handle error, its flexible execution, and variants of this paradigm as they apply to specific problems of the space environment.

Automated Planning for Autonomous Systems

In engineering increasingly autonomous systems, we must concentrate on choosing an appropriate operating paradigm. Proposed spaceborne systems such as Space Station and SDI require continuous reliable operation. The environment will vary over time. Plans must be proposed internally, during operation to dynamically adapt system operations and consistently meet mission objectives throughout the system lifetime. These requirements focus our interest on automated planning.
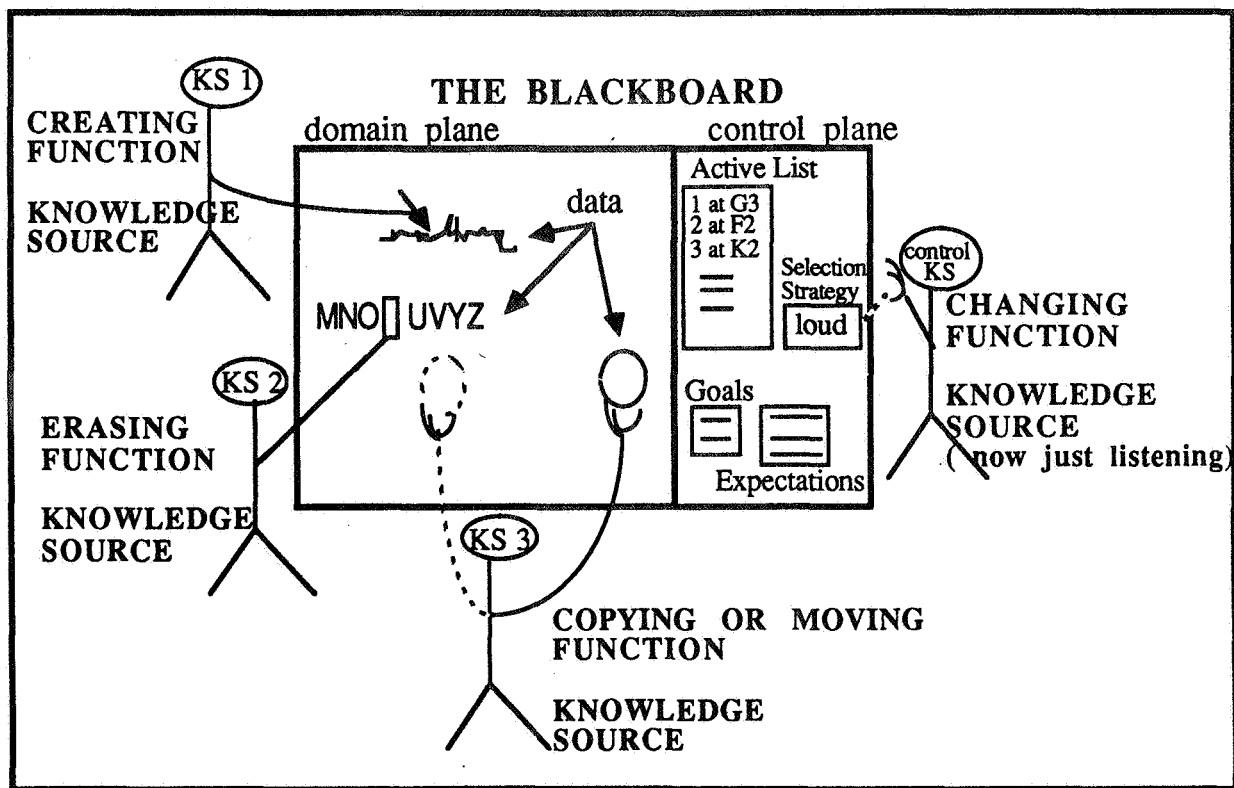
This paper extends "planners" to include time-dependent functionality such as prediction, resource allocation and context management. These factors affect the successful achievement of mission goals over the expected long lifetime of the autonomous system. Major concerns are efficient organization of multiple subsystems and adaptive management of finite resources. Planning to effectively manage these interrelated operations will be an important feature for an autonomous system.

System state determination for a large autonomous system such as Space Station or an SDI Battle Manager will be distributed among subsystems in an hierarchical manner. Effective management of such a scheme is critical to system operations.

683

At each functional level in the hierarchy, one system at the next level would be responsible for control of state determination for lower levels. Eventually, in this hierarchy, the ground becomes the next logical step to aggregate system state. If true autonomy is to be achieved, an intelligent replacement for these ground-based functions will be required. The planner paradigm is a helpful model for automating management, prediction and stabilization presently performed by man. In the following sections of this paper, a planner model will be presented in the abstract. A planning domain will be discussed separately. That domain is the top-level system executive for a complex autonomous space operation.

The Blackboard Planner

Planners based upon the blackboard architecture provide for adaptive control management and the ability to handle erroneous data or sub-optimal analysis methods to solve difficult problems [Hayes-Roth79] [Erman81] [Nii86] . Active objects within the blackboard paradigm are called *knowledge sources* . *Knowledge sources* communicate through a global knowledge base, *the blackboard.* By posting, receiving and changing messages from this *blackboard* many possibilities are available to the system of executing *knowledge sources* (KS's). See Figure 1 for a simplified and general example of a blackboard system. A blackboard planner is not static. Any planner for an autonomous mission must be able to continuously adapt to the current context. Within the blackboard paradigm, plans can be "opportunistically" derived from current data input, current control strategy, or the current highest ranked goal(s).

Blackboard Architecture with Types of Knowledge Sources
Figure 1

A basic tenet of the blackboard architecture is modularity. *Knowledge sources* can be added, removed or replaced with ease because their only interaction with each other is specified completely and exactly through *the blackboard.* Specifications for autonomous planning systems may be ill-specified. The modular nature of a blackboard development environment is thus advantageous to develop gradually increasing expertise in these difficult areas. To investigate trade-offs during development, a blackboard planner allows one to include many approaches, encoded as multiple, cooperating *knowledge sources.* If this redundancy is available in the executable system, then the planner can vary coordinations as the runtime situation changes.

It is also possible to partition *the blackboard* into different levels of abstraction, multiple compartments, contexts, or time slices. In doing this, efficiencies specific to the particular planning domain can be achieved. In addition, partitioning the triggering area often allows multiple *knowledge sources* to execute in parallel. The blackboard architecture is an interrupt system rather than a polling system and thus performs better. Modular paradigms (e.g. rule-based systems) often do not show a significant increase in performance with an increase in the amount of knowledge. The blackboard does permit increasing performance through its emphasis on knowledge-based coordination.

Handling Uncertainty

An autonomous system will need to incorporate multiple sources of uncertain knowledge. The data may contain errors; some *knowledge sources,* for the sake of speed, may not be optimal. The blackboard architecture allows the effective combination of these uncertain sources into a competent decision-making tool.

Multiple control strategies (see Table 1) can create multiple lines of inference within blackboard planner operation. Messages on *the blackboard* are often termed hypotheses, or the hypothesis. At an intermediate planning stage there might still be several competing answers to the problem. Multiple lines of inference coordinate efforts in generating and eliminating different hypotheses to achieve a better final answer. For example, a predictive algorithm could be given boundaries to work within through previous reduction of a higher level goal. Or, multiple compartments performing similar computations could pool hypotheses in order to feedback improvements to their individual results.

Control Strategy	Possible Knowledge Source
Bottom-Up	Data Fusion
Top-Down	Goal-to-Subgoal
Island-Building	Nearest Neighbor

Table 1

Blackboard planners tolerate faults well and degrade gracefully. These traits are directly related to the property of modularity. Because the blackboard architecture is

modular, depth can be added to the planning. Deep reasoners are characterized by smaller, more general reasoning steps which allow a broader range of problems to be covered, usually at the cost of more steps and thus less speed. To avoid making a brittle planning system, depth can be specifically added as back-up to more sharply focused, shallower knowledge. Back-up through depth or through multiple shallow reasoning paths is a characteristic of blackboard planners and allows them to continue gracefully as variants of base test problems arise.

Autonomous System Executive Fault Protection

A major problem encountered in autonomous operation is management of state determination. Control is often distributed through many levels and perturbed by faults. The challenge is to plan appropriate fault recovery mechanisms under changing conditions in the environment and in the system itself.

The most strategic recovery configuration is that based on current context, current assumptions. The space of available contexts varies. Indeed more than one context may be viable at the same time. Multiple contexts map to multiple hypotheses on the *blackboard.* This array of choices of "the truth" can be maintained within the blackboard architecture to provide an adaptable framework for fault isolation and recovery planning.

Multiple coordinating methods of state determination are essential because of the critical nature of fault protection in autonomous systems. The blackboard planner's ability to coordinate multiple sources of knowledge provides the environment to develop and execute these methods of state determination. Truth maintenance provides one model for state determination [Doyle80] [deKleer86]. Relationships between a context of assumptions and a hypothesis would be built into *the blackboard* and modules written to maintain these contexts. For example, truth maintenance *knowledge sources might* update dependencies between data or choose a current context. This blackboard planner restructuring would,at the very least, permit dynamic system resiliency based upon recovery mechanisms which have been assigned to different contexts.

The blackboard paradigm permits further enhancements. For example, the context-based recovery mechanisms could be aided by some sort of output data normality testing. Whether any subsystem is performing according to expectations could be, for instance, answered by pre-assigned bounds , by a system of alert statuses incorporated into the contexts, or perhaps with a "suspicion" *knowledge source* that reviews strange behavior in hopes of isolating otherwise undetectable faults.

Some Example Fault Protection KS's

1. Alert Status Change
2. Functional Output Testing
3. Normality Audit
4. Suspicious Behavior Audit

Table 2

Knowledge sources, such as those in Table 2, combined with a context or truth maintenance model, will provide system executive fault protection software that dynamically changes to meet new situations.

Evolution to Full Autonomy

Functions like determining suspicious behavior are ill-specified for automation. Many human functions will need to remain in man's hands while automation of other, more concrete modules proceeds. The blackboard planner's modularity allows autonomy to develop during a period of adjustment. The "suspicion" *knowledge source* for example can be implemented as a Man Machine Interface (MMI) in the beginning, allowing developers to experiment with and record their reactions to different situations and actions before encoding these internally. This type of human placeholder is called "man-as-a-knowledge-source" because, as an active object within the blackboard planner, the interface would be a *knowledge source* .

The evolution of placeholders to decision-making *knowledge sources* can only take place if the planner is explainable. An explanation facility is easily encompassed within the blackboard paradigm by storing a history of actions performed. Because control features such as KS choice strategy can be explictly stored on *the blackboard* [Hayes-Roth84,85], the system can not only explain what it did, but also why. With the simultaneous existence of such an explainable prototype planner and a compiled executable planning system, perhaps in space, development proceeds smoothly.

Current Research Issues

Current research issues center on the viabilitiy of moving blackboard planners from the research laboratory into the real world. There, theoretical performance must be demonstrated on actual hardware and embedded within systems software. Increase in speed through parallel implementation is one key issue. Changes to existing blackboard systems' foundations in order to achieve this parallelism may be a second issue. The blackboard paradigm is inherently parallel. This is clearer if one sees that the software model is based upon the physical blackboard and its usual uses. Most implementations to date have implemented serial execution; control has thus been an attempt to find the one best KS to execute next. Although exceptions exist, [Erickson85] [Stentz85], there is still a lack of evaluations of multiple parallel architectures and their accompanying systems software trying to find the best fit to the blackboard architecture. This is an area Martin Marietta is currently investigating, supported in part by USAF contract #F30602-86-C-0062 through Rome Air Development Center.

Finding the appropriate building blocks for a parallel implementation is another underdeveloped issue. Commonly, blackboard systems are built on top of a frame system [Erickson85] [Nii82] [Nii81]. In doing so, blackboard hypothesis organization is greatly simplified because means for partitioning and typing data are already in existence. Research to date supports two reasonable approaches toward achieving parallel implementation:

1.   Continue with the current frame system basis and use a shared-memory multiprocessor. Both a global *blackboard* and a frame inheritance hierarchy are shared memory structures. This approach emphasizes memory contention problems.

2.   Adapt an object-oriented programming system or frame system especially for blackboards, where every object contains a mini blackboard planner executive. This would enable fine grained distribution of tasks and of memory contention resolution. *Blackboard* data objects would then be "intelligent" in the sense of intelligent terminals. The main problem presented by this approach is communications contentions.

Systems engineering of these approaches will require maximizing one of two parameters, either memory access organizational efficiency or connection protocols & topologies. In any specific application, islands of knowledge interaction and shared knowledge will develop which may emphasize one of these approaches above the other.

Summary

In conclusion, the blackboard planner paradigm seems suited to the planning problems encountered in developing autonomous spaceborne systems and should be developed further. The example planning problem of fault recovery and state determination within the multi-level system executive showed the complexity of planning for autonomy. A blackboard planner manages these complexities by combining many sources of uncertain knowledge and permitting especially difficult problems to be explored through human interaction with the planner. Explicitly describing the planning environment as changeable, e.g. a truth maintenance model, avoids falsely modeling assumptions as static for planning components of a complex autonomous system operating in a time-varying space environment. Planning must be dynamic. A static scheduling approach, even if optimal at design, will be too brittle for an autonomous application.

The blackboard architecture planner system described in this paper deserves futher investigation. Research to date has interesting implications in other areas of computer science which need exploration. Some possible future projects are:

1. Compilers for creating efficient executable code
2. Hardware architecture designs perhaps based upon the idea of active data objects
3. MMI designs based upon the blackboard architecture.

Details such as the integration of truth maintenance ideas with the blackboard framework and quantified control of feedback between separate areas of planning knowledge are both topics for further study.

References

deKleer, J., (1986) "An Assumption-based TMS," *Journal of Artificial Intelligence*, 28: 127-162.

Doyle, J., (1979) "A Truth Maintenance System," *Journal of Artificial Intelligence*, 12: 231-272.

Erickson, W.K., (1985) "The Blackboard Model: A Framework for Integrating Cooperating Expert Systems," *AIAA/ACM/NASA/IEEE Computers in Aerospace V Conference*, Long Beach, CA, Oct. 21-23, 1985.

Erman, D. L., F. Hayes-Roth, V.R. Lesser & D. Raj. Reddy, (1981) "The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty," *Readings in Artificial Intelligence*, eds B. Webber & N. Nilsson, Tioga Publishing, Palo Alto, CA, pp 349-389.

Hayes-Roth, B., F. Hayes-Roth, Stan Rosenschein & Stephanie Cammarata, (1979) "Modeling Planning as an Incremental, Opportunistic Process," *IJCAI-79*, vol 1, pp. 375-383.

Hayes-Roth, B., (1984) "BB1: An Architecture for Blackboard Systems That Control, Explain, and Learn About Their Own Behavior," HPP Report 84-16, Stanford University.

Hayes-Roth, B., (1985) "Blackboard Architecture for Control," *Journal of Artificial Intelligence*, 26: 251-321.

Nii, H. Penny, N. Aiello, C. Bock & W.C. White, (1981) "Joy of AGE-ing: An Introduction to the AGE-1 System", HPP Report 81-23, Stanford University.

Nii, H. Penny, E. Feigenbaum, J. Anton & A.J. Rockmore, (1982) "Signal-to-Symbol Transformation: HASP/SIAP Case Study," *The AI Magazine*, Spring 1982, pp 23-35.

Nii, H. Penny, (1986) "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures," *The AI Magazine*, Summer 1986, pp 38-53.

Stentz, A. & S. Shafer, (1985) "Module Programmer's Guide to Local Map Builder for ALVan," Technical Report Carnegie-Mellon University, Computer Science Department, August.

"Space Station Automation Study: Autonomous Systems and Assembly, Final Report," MMDA, November 1984.

DESIGN CONSIDERATION IN CONSTRUCTING HIGH PERFORMANCE EMBEDDED KNOWLEDGE-BASED SYSTEMS (KBS)

Authors: Shelly D. Dalton
Philip C. Daley

Martin Marietta Denver Aerospace
Denver, Colorado

ABSTRACT

As the hardware trends for artificial intelligence (AI) involve more and more complexity, the process of optimizing the computer system design for a particular problem will also increase in complexity. Space applications of knowledge-based systems (KBS) systems will often require an ability to perform both numerically intensive vector computations and real-time symbolic computations. Although parallel machines can theoretically achieve the speeds necessary for most of these problems, if the application itself is not highly parallel, the machine's power cannot be utilized. A scheme is presented here which will provide the computer systems engineer with a tool for analyzing machines with various configurations of array, symbolic, scaler, and multi-processors. High speed networks and interconnections make customized, distributed, intelligent systems feasible for the application of AI in space. The method presented in this paper can be used to optimize such AI system configurations and to make comparisons between existing computer systems. It is an open question whether or not, for a given mission requirement, a suitable computer system design can be constructed for any amount of money. Additionally, significant cost and performance risk can occur which will be avoided by careful adherence to guidelines presented in this paper. This work is supported, in part, by Air Force contract F30602-86-C-0062 from the Rome Air Development Center.

INTRODUCTION

Historically, computer systems which were used for space applications focused primarily on support of the flight package functions: attitude and control, guidance and navigation, thermal control and power control. Scientific processing for experimental data collection and manipulation, was mainly limited to ground located systems which need not be real-time. Space Station and Strategic Defense Initiative (SDI) are proposing systems which will not only require enormous amounts of space-based, real-time scientific processing, but will also incorporate knowledge-based systems (KBS) into the embedded system. The science package for Space Station is expected to be 3-4 times as large as the flight package; while SDI applications such as surveillance, acquisition,

tracking and kill assessment (SATKA), battle management and weapon control/fire control will require a science package 10-100 times as large as the flight package. An involved design approach must be taken if these systems are to meet both performance and KBS requirements.

FUTURE SYSTEMS FOR SPACE

Although software and hardware environments have become richer and more efficient, integration of these environments must be carefully engineered. For Space Station and SDI, a heterogeneous mix of serial, symbolic (KBS) and vector instructions will be required. For example, processing of external commands could use conventional serial instructions, the tracking and pointing of an antenna array could use vector instructions, and fault recovery could be implemented in an expert system using symbolic manipulation. For optimum processing, each of these instructions sets could be housed in a separate processor (or multi-processor) specialized for that type of instruction, linked in either a tightly or loosely coupled heterogeneous system. While, conceptually, a distributed system is easy to manage, many difficulties exist in meeting the system software and the communications requirements. Analyses of various configurations also becomes increasingly difficult as the size and complexity of the system grows.

Due to the relative simplicity of the science package in the past, the design of on-board data management system was straight-forward. One-CPU serial processors were used which were rated less than 1-MIP. For example, the ATAC-16MS computer used by NASA's Magellan and Galileo is rated at 0.5 MIPS [ATAC, 79]. Most programs consisted of a few thousand lines of fixed-point instructions running at 0.001-0.1 MIPS. Design considerations were limited to trading-off rated speeds of space-qualified processors. For ground systems, high performance computers could be evaluated by performing analysis on these homogeneous systems [Hockney, 84], [Mohan, 84] and benchmarking.

ANALYSIS OF SYSTEM REQUIREMENTS

Today, the varying types of instructions required in the system (serial, symbolic and vector) must be analyzed with respect to quantity, structure and complexity [Cook, 84]. Also, various configurations of heterogeneous processors (pipelined, symbolic, array and multi-processors) with their communications schemes must be analyzed for data flow and effective speeds. Even at the highest level of analysis, MIPS-ratings (million instructions per second) for serial instructions cannot be compared with FLOPS-ratings (floating point operations per second) for floating point operations on another machine and LIPS-ratings (logical inferences per sendon) for symbolic (Prolog) operations on a third. Any rated speed must also be scrutinized with respect to the sustained speed which the machine can deliver when operating system services are also being performed in conjunction with the application. For most machines, actual performance is approximately one-tenth rated speed [Dongarra,

692

85]. For array and multi-processor systems, actual performance will be even smaller due to time needed for overhead and synchronization of tasks. Systems design is now far from simple.

AMDAHL'S LAW

Machines expoiting parallelism are also constrained by a principle well known among computer architects--the so-called "Amdahl's Law". Amdahl's Law states that the overall speed-up which can be obtained in a parallel system is inversely proportional to the fraction of the application which must be performed serially [Kibler, 85]. For example, if 10% of the code is serial, then the maximum speed increase over a single processor machine is 10 times, regardless of the number of processors in the parallel machine. This is antagonistic to the commonly held belief that any performance level can be reached if enough parallelism is used in the machinery. Every application has an inherent limitation on speed-up, so the degree the application is susceptible to parallelism must be understood before the hardware is choosen.

IMPACT OF KBS

While previous space-based systems have had well defined requirements which fit easily into available hardware, current requirements for space systems which will include KBS are inherently "soft" and tend to tax the performance of available hardware. The "softness" of the requirements is due, in part, to the difficulty in expressing, a priority, the compute requirements for functions normally performed by humans and their interactions with large ground systems. These functions are now proposed to be achieved via KBS running in space. Also, KBS has in the past rarely been embedded, even in ground systems. The wealth of management guidelines, specification guidelines, and design considerations which are well known for conventional embedded systems have not yet been modified to incorporate KBS [Daley, 85].

Unfortunately, the modifications in systems engineering which must be made for future space systems go well beyond those necessary for KBS. Fault tolerance and trustedness cannot just be added as an applique to design specifications as was done in the past. The requirements for the applications alone may approach the limits of hardware technology. Additional systems services which are not integrated with the application could easily reduce performance to an unexceptable level. Therefore, fault tolerance, KBS, parallel processing, and trustedness must be integrated from design conception in order to develop requirements with some degree of confidence and achievability.

TYPES OF ARCHITECTURES

Today's system engineering must involve a deep understanding of the type of architecture available for high performance computers. Fault tolerance can be integrated well with array and multi-processors. Pipelined architectures may be needed for image

693

computations and stack architectures for lisp processing. The luxury of using an architecture which is tailored to the application may well be achievable for distributed systems.

Array processors, such as the GAPP built by Martin Marietta, expoit fine grain parallelism for applications such as signal and image processing, and computations involving vectors or matrices which are large compared to the number of processing elements. Each processing element of the array is fairly simple, being controlled synchronously by a central control unit and/or a front-end processor. These processors are not efficient for problems in KBS which need asynchronous operations that are not highly structured.

Multiprocessors, such as the Butterfly built by BBN, can run asynchronously and have attracted a great deal of attention from the AI community. The processing elements can vary in power from arithmetic logic units (ALU) to small VAX's. Configurations are available which specialize in areas such as semantic networks or floating point operations, using local and/or global memories. Unfortunately, software development is very difficult on most of these machines, and it is difficult to distribute tasks to effectively use the hardware. Care must be taken to find an architecture which complements the task structure of the application, not one which merely provides enormous computing power.

Originally, pipelined computers (such as CRAY's) where used to achieve high performance. While these computers can still out-perform most multi-processor configurations, maintenance and environmental supports are often too complex for space. Their pipelines divide computations such as floating point operations into stages. Elements of vectors follow each other through the pipeline. The pipeline is less efficient when it is being filled or emptied than when it achieves steady-state of a full pipeline. When at steady-state, maximum speed-up is proportional to the number of pipeline stages. Since KBS applications do not use large vectors, pipelined processors do not provide enormous speed increases, but instruction pre-fetch pipelines can be utilized by almost any type of application.

The type of architecture which has been most effective for KBS is stack-oriented architecture. Stacks are used to hold lists and results of function calls in stack buffers which do not require explicit addresses. Since addresses are not as important in AI as in conventional languages such as Fortran, associative array processors and content addressable memory are also currently being researched as environments for KBS. Since software cost currently exceeds hardware costs, the ease of programming still makes stack-oriented architectures the most viable choice for many KBS applications.

DATA MANAGEMENT

The emphasis on speed for high performance architectures tends to overshadow the need for efficient data management by the

operating system. Immense processing power is useless if it must wait on the operating system and the I/O channels.

Data management will vary for the different architectures. For KBS, type checking and garbage collection functions must be provided [Hirsh, 84]. For processors which share memory, the data integrity problem is even greater than what is involved with virtual meory, and can quickly become intractible. Not even virtual memory management is simple, since most common approaches are based on disks which cannot be easily used in space systems. Memory problems with space systems cannot be easily fixed by adding, a posteriori, more memory or operating system software. As with the application software, the complexity of the operating system software will be high, so small deviations in requirements could cause huge reductions in performance.

From a hardware point of view, a distributed system which provides parallel machines for vector and matrix operations, stack machines for KBS and conventional machines for serial processing is becoming fairly simple. From a software point of view, operating systems do not yet exist which can distribute and control a mix of instructions over a heterogeneous hardware environment. Until such an operating system is developed, the system must be loosely coupled with the developer constantly aware of the internal interface capabilities.

CONCLUSION

Hardware is becoming available to provide for enormous increases in performance in space for the future. Likewise, software techniques such as KBS are providing means to drastically increase functionality for space systems. If systems engineering does not rise to the challenge of providing means to effectively design and control these new, complex systems, then future space systems may face additional programmatic risks or degradation of performance.

REFERENCES

[ATAC 79]   ATAC-16MS Principles of Operation, Applied
                 Technology Coporation, June 1979

[Cook 83]   Cook, Stephen A., "An Overview of Computational
                 Complexity," Communications of the ACM, 26 (1983)
                 P 401-408

[Daley 85]   Daley, Phil C., "Knowledge-Based Systems: Systems
                 Engineering and Management Issues," Conference on
                 Artificial Intelligence for Space Applications,
                 Huntsville, Alabama, 1985

[Dongarra 85]   Dongarra, J. J., "Performance of Various Computers
                 Using Standard Linear Equations Software in a
                 Fortran Environment," Mathematics and Computer
                 Science Division, Argonne National Laborator, Oct.
                 11, 1985

[Hirsh 84]   Hirsh, Abraham, "Tagged Architecture Supports
                 Symbolic Processing," Computer Design, June 1,
                 1984

[Hockney 84]   Hockney, R. W., "Performance of Parallel
                 Computers, High Speed Computation, J. S. Kowalik,
                 ed., 1984

[Kibler 85]   Kibler, Dennis F. and Conery, John, "Parallelism
                 in AI Programs", IICAI, 1985

[Mohan 84]   Mohan Joseph, "Performance of Parallel Programs:
                 Model and Analysis," PhD. Dissertation,
                 Carnegie-Mellon University, 1983

Validation of Expert Systems

Rolf A Stachowitz and Jacqueline B Combs

Lockheed Missiles & Space Company, Inc.

Software Technology Center, O/96-01, B/30E

2124 E St. Elmo Rd.

Austin, Texas, 78744

Abstract[1]

The validation of expert systems (ES's) has only recently become an active AI research topic. Current approaches have concentrated mainly on the validation of rule properties (basically syntactic) of such systems. Our effort improves on current methods by also exploiting the structural and semantic information of such systems.

To increase programmer productivity, more and more companies have begun exploiting the advent of AI technology by developing applications using ES shells or other AI-based high-level program generators.

Whereas papers and books on validating traditional software abound, the validation of ES's has received considerably less attention in the AI literature. Rare exceptions are Nguyen et al. [3] and Suwa et al. [4].

This lack of attention is also reflected in the "standard" AI terminology where *Validation of ES's* is used synonymously with *Evaluation of the Quality (or Performance) of ES's* rather than with *VV&T* (see, for example, the index in Buchanan/Shortliffe [1]). Most existing, commercially available ES shells correspondingly provide, if at all, only rudimentary support for validating ES's. Thus **KEE** (Knowledge Engineering Environment, IntelliCorp) mainly supports *legal values* and *legal numeric ranges*; the sophisticated **ART** (Automated Reasoning Tool, Inference Corporation), so far, does not provide any validation tools beyond straight-forward syntax checking.[2]

A notable exception is **LES**, (Lockheed Expert System). Like **KEE**, it supports *legal values* and *legal numeric ranges*. It also provides a program, *Check*, which detects potential errors in ES rules, such as *dead-end clauses*, *unreachable clauses*, *irrelevant clauses*, *cycles* in rules and certain cases of *inconsistency* and *incompleteness* (Perkins et al [2]).

In our opinion even the LES effort suffers from the fact that only very little use is made of semantic information or metaknowledge in validating ES's. Beginning in 1986, the AI group of the Lockheed Software Technology Center (LSTC) at Austin has started work on **EVA** (Expert Systems Validation Associate) which makes use of metaknowledge to validate ES's.

In the remainder of this paper, we outline the architecture, functionality, and future goals of **EVA** and describe the features that have been implemented *for* **ART** and, partially, *in* **ART**, the ES shell used at LSTC.

References

1. B.G. Buchanan, E.H. Shortliffe (Ed.). *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley Publishing Company, Reading, MA, .

2. *LES User's Manual.* Lockheed Missiles & Space Company, Inc., 1986.

3. T.A. Nguyen, W.A. Perkins, T.J. Laffey, D. Pecora. An Expert Systems Knowledge Base for Consistency and Completeness. Proceedings of the Ninth International Joint Conference on Artificial Intelligence, 1985, pp. 375-378.

4. M. Suwa, A.C. Scott, E.H. Shortliffe. "An Approach to Verifying Completeness and Consistency in a Rule-Based Expert System". (1982), 16-21.

A NONLINEAR FILTERING PROCESS DIAGNOSTIC SYSTEM FOR THE SPACE STATION

Raymond R. Yoel

Boeing Aerospace Company
Space Station Program
Huntsville, Alabama 35807

Dr. M. Buchner
Dr. K. Loparo
Arif Cubukcu

Case Western Reserve University
Systems Engineering Department
Cleveland, Ohio 44106

ABSTRACT

This paper will present a nonlinear filtering process diagnostic system, terrestrial simulation and real-time implementation studies, and discuss possible applications to Space Station subsystem elements.

There is the need for computer based process diagnostic systems on the Space Station. Certain processes within Space Station subsytems, e.g., Thermal Control System (TCS) and Process Material Management System (PMMS), can not be continuously monitored by the astronauts. Further, there are numerous failure modes where the time necessary for an astronaut to detect and isolate a failure is greater than the time for those failures to result in serious, even disastrous conditions. Throughout the terrestrial chemical, oil, utility, and other process industries, computer based process diagnostic systems are being implemented at an increasing rate. The field of process diagnostics has provided many techniques, with each having distinct advantages and disadvantages.

The objectives for implementing a computer based process diagnostics system are dedicated monitoring and quick detection times. Further, the system can not give failure mode response under normal conditions, i.e., false alarms. Once the reliability of the system is questioned, lack of confidence can result in slow or incorrect action by astronauts or automatic control systems.

At Case Western Reserve University, a process diagnostic system using model based nonlinear filtering for systems with random structure has been shown to provide improvements in stability, robustness, and overall performance in comparison to linear filter based systems. A sub-optimal version of the nonlinear filter (zero-order approximation filter, or ZOA filter, similar to the multiple model filter introduced by Magill) was used in simulation studies, initially, with a pressurized water reactor model and then with water/steam heat exchanger models. Finally, a real-time implementation for leak detection in a water/steam heat exchanger was conducted using the ZOA filter and heat exchanger models.

This nonlinear filtering process diagnostic system can be applied to Space Station subsystem elements which have process parameters with random structure. Some possible applications are:

    TCS: heat exchangers, valves, sensors, piping
    PMMS: separators, storage tanks, valves, sensors, piping

701

DESCRIPTION

The integration and implementation of an artificial intelligence system
that controls two robots, a vision system, tactile sensors, and a
supermini process control computer in R & D robotic applications is
discussed. The system uses an Expert System developed in a Common-Lisp
environment which can solve a variety of problems using a rule based
system. This 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.
The expert system resides within a real-time process control system.

ABSTRACT

REAL TIME AI EXPERT SYSTEM FOR ROBOTIC APPLICATIONS

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 are 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 effect, the system
can learn new rules. The expert system is interfaced through touch
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 information required to direct the robots to complete the desired
task. The presentation discusses the mechanisms that the expert system
uses to solves problems (goals), the different rule data bases, 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.

John F. Follin, Staff Engineer, Robotics and Heuristic Process Control

GA TECHNOLOGIES
PO BOX 85608
SAN DIEGO, CA   92138
(619) 455-4405

COMPUTER USED :   MV/4000 with disk, tape, line printer, and operator
                   consoles

EQUIPMENT INTERFACED THROUGH COMPUTER :

                   GCA/Par overhead robot
                   Prab cylindrical robot
                   Machine vision Genesis 2000 CCD vision system
                   CAMAC data acquisition systems (digital and analog)
                   Speech synthesizer
                   Voice Recognition
                   Megatek Graphics Display Station
                   Graphic terminals fitted with touch sensors
                   Joystick controller unit for robot control
                   Tactile touch sensors for GCA/Par robot

MATHEMATICAL ALGORITHMS FOR APPROXIMATE REASONING

John H. Murphy, Seung C. Chay and Mary M. Downs
Westinghouse R&D Center
Pittsburgh, PA 15235

ABSTRACT

Most state-of-the-art expert system environments contain a single and often ad hoc strategy for approximate reasoning. Some environments provide facilities to program the approximate reasoning algorithms. However, the next generation of expert systems should have an environment which contains a choice of several mathematical algorithms for approximate reasoning. To meet the need for validatable and verifiable coding, the expert system environment must no longer depend upon ad hoc reasoning techniques but instead must include mathematically rigorous techniques for approximate reasoning. In this paper, we review popular approximate reasoning techniques including: certainty factors, belief measures, Bayesian probabilities, fuzzy logic, and Shafer-Dempster techniques for reasoning. Then we concentrate on a group of mathematically rigorous algorithms for approximate reasoning that could form the basis of a next generation expert system environment. These algorithms are based upon the axioms of set theory and probability theory. To separate these algorithms for approximate reasoning, various conditions of mutual exclusivity and independence are imposed upon the assertions. Approximate reasoning algorithms presented include: reasoning with statistically independent assertions, reasoning with mutually exclusive assertions, reasoning

with assertions that exhibit minimum overlap within the state space, reasoning with assertions that exhibit maximum overlap within the state space (i.e. fuzzy logic), pessimistic reasoning (i.e. worst case analysis), optimistic reasoning (i.e. best case analysis), and reasoning with assertions with absolutely no knowledge of the possible dependency among the assertions. To further separate these approximate reasoning algorithms, a robust environment for expert system construction should include the two modes of inference: modus ponens and modus tollens. Modus ponens inference is based upon reasoning towards the conclusion in a statement of logical implication, whereas modus tollens inference is based upon reasoning away from the conclusion. These algorithms, when consistently applied, allow one to reason accurately with uncertain data. The above environment can also replicate most state-of-the-art expert system environments which provides a continuity between the current expert systems which cannot be validated nor verified and future expert systems which should be both validated and verified.

ABSTRACT

REAL-TIME SPACE SYSTEM CONTROL
WITH EXPERT SYSTEMS

David Leinweber, Ph.D
Lowell Hawkinson
LISP Machine Inc.
Los Angeles, California   90045

John Perry, Ph.D.
OAO Corporation
Los Angeles, California   90245

Many aspects of space system operations involve continuous control of real-time processes. These processes include electrical power system monitoring, pre-launch and ongoing propulsion system health and maintenance, environmental and life support systems, space suit checkout, on-board manufacturing, and vehicle servicing including satellites, shuttles, orbital manuvering vehicles, orbital transfer vehicles and remote teleoperators. Traditionally, monitoring of these critical real-time processes has been done by trained human experts monitoring telemetry data. However, the long duration of future space missions and the high cost of crew time in space creates a powerful economic incentive for the development of highly autonomous knowledge-based expert control procedures for these space systems.

In addition to controlling the normal operations of these processes, the expert systems must also be able to quickly respond to anomalous events, determine their cause and initiate corrective actions in a safe and timely manner. This must be accomplished without excessive diversion of system resources from ongoing control activities. Any events beyond the scope of the expert control and diagnosis functions must be recognized and brought to the attention of human operators. Real-time sensor-based expert systems (as opposed to off-line, consulting or planning systems receiving data via the keyboard) pose particular problems associated with sensor failures, sensor degradation and data consistency, which must be explicitly handled in an efficient manner. A set of these systems must also be able to work together in a cooperative manner.

This paper describes the requirements for real-time expert systems in space station control, and presents prototype implementations of space system expert control procedures in PICON (process intelligent control) for real world examples. PICON is a real-time expert system shell which operates in parallel with distributed data acquisition systems. It incorporates a specialized inference engine with a specialized scheduling portion specifically designed to match the allocation of system resources with the operational

requirements of real-time control systems. Innovative knowledge engineering techniques used in PICON to facilitate the development of real-time sensor-based expert systems which use the special features of the inference engine are illustrated in the prototype examples.

The paper concludes with a discussion of new facilities being incorporated into the PICON system, including automatic rule generation and diagnostic capabilities based solely on descriptive frames.

A FLEXIBLE SEARCH STRATEGY FOR PRODUCTION SYSTEMS

Pradip Dey
S. Srinivasan
K. R. Sundararaghavan
University of Alabama at Birmingham, AL 35294

Most problems considered to be solvable by expert systems have very large search space. It is, therefore, imperative to use efficient search strategy in expert system tools. Thus, OPS5 uses a kind of hill-climbing which is very efficient. However, hill-climbing is inadequate for many problems because it is one of the least dependable search strategies. It fails in ridges, and local maximum. In order to make the search efficient and adequate one can (1) adopt best-first search instead of hill-climbing or (2) modify hill-climbing with intelligent backtracking. There are some serious problems in application of best-first in expert systems. Therefore, the second alternative is adopted. It is implemented in a production system called PRO2 embedded in C running on UNIX. We call the search hill-tracking. It has the advantage that it reduces search space by using the hill-climbing function and avoids the deficiencies of hill-climbing by recovering from ridges, and local maximum by intelligent backtracking.

PRO2 is a general purpose tool for developing expert systems. This is a rule based production system with an effective, intelligent and flexible backtracking control mechanism, which makes the system more dependable. In case the goal state is not reached by following a path the system backtracks to an earlier point and tries alternative paths. But ordinary cronological backtracking is grossly inefficient. PRO2 has two features which allow efficient backtracking: (1) In each recognize-act cycle, at most three rules are selected by conflict resolution strategies; the best one is fired and the other two are held in an ordered set as points of backtracking. This feature reduces the number of alternatives to be tried in the event of a backtracking. (2) The system backtracks only after it fails to provide a suitable solution without backtracking. Thus, if PRO2 reaches a dead-end then it will backtrack to an earlier point and try one of the untried rules saved in the conflict resolution phase. The system also backtracks if the user is not satisfied with the solution and requests for alternatives. Thus, this system provides a greater opportunity to find an acceptable solution if the user is not satisfied with the earlier solution provided. Most production systems are either inefficient or inadequate to search alternative paths. In space applications where safety is of prime importance PRO2 will have a competitive edge over other tools, because it is efficient, flexible, and adequate for building highly dependable expert systems.

Semantic Based Man-Machine Interface for Real-Time Communication

M. Ali and C.-S. Ai
Knowledge Engineering Laboratory
The University of Tennessee Space Institute
Tullahoma, Tennessee 37388

ABSTRACT

A flight expert system (FLES) has been developed to assist pilots in monitoring, diagnosing and recovering from in-flight faults. To provide a communications interface between the flight crew and FLES, a natural language interface (NALI) has been implemented. Input to NALI is processed by three processors: 1) the semantic parser. 2) the knowledge retriever, and 3) the response generator. First, the semantic parser extracts meaningful words and phrases to generate an internal representation of the query. At this point, the semantic parser has the ability to map different input forms related to the same concept into the same internal representation. Then the knowledge retriever analyzes and stores the context of the query to aid in resolving ellipses and pronoun references. At the end of this process. a sequence of retrieval functions is created as a first step in generating the proper response. Finally, the response generator generates the natural language response to the query.

FLES's knowledge-base consists of temporal as well as non-temporal knowledge. The temporal knowledge is mainly concerned with the order and timing of events. Component failures. sensor failures and abnormal situations are a few examples of events. The non-temporal knowledge is concerned with the structural and diagnostic aspects of the flight domain. The architecture of NALI has been designed to process both the temporal and non-temporal queries. Provisions have also been made to reduce the number of system modifications required for adapting NALI to other domains. This paper describes the architecture and implementation of NALI.

The Concurrent Common Lisp Development Environment

1 Summary

A discussion of the Concurrent Common Lisp Development Environment on the iNTEL Personal Super Computer (iPSC) is presented. The advent of AI based engineering design tools has lead to a need for increased performance of computational facilities which support those tools. Gold Hill has approached this problem by directing its efforts to the creation of a concurrent, distributed AI development environment. This discussion will focus on the development tools aspect of the CCLISP environment. The future direction of Gold Hill in the area of distributed AI support environments is also presented.

2 Outline of Talk

1. AI techniques are providing a basis for current generation Engineering design tools

2. As with other AI applications, these tools are being limited by current generation computational facilities

3. Gold Hill is removing those limitations by developing a concurrent development environment which allows increased performance for the standard AI tools.

4. The original vision:
   - Bring modern AI development tools to the market on generic micro-processor based systems.

   - Result: Golden Common Lisp Developer on 80286 based machines.

   - Facilitate Distributed AI support environments as an extension to the stand-alone environment

   - Result: GCLISP-Network and CCLISP

   - Provide Access to multiple, loosely coupled nodes via message passing semantics.

5. The search for the appropriate vehicle led to iNTEL Corp, and the iPSC.

6. Under a joint development agreement, iNTEL and Gold Hill are bringing the first of the new generation, concurrent AI tools environments to market.

7. Product development is well underway, with the current alpha version being demonstrated at this conference.

8. Common Lisp is the basic development tool.

9. Extended to support messages passing via streams which are used to communicate with other processors in the iPSC.

10. The familiar development tools, including compilers, steppers and computation analysis facilities are extended to allow control of sets computations within the iPSC.

11. Access to the CCLISP environment from external Lisp based workstations. (Symbolics, TI Explorer, and IBM-AT's)

12. A short example of application in CCLISP

13. The future:
    a. Providing a uniform application interface to support the distribution of computations to external machines

    b. based upon open systems/actor technologies

    c. Provide distributed AI knowledge bases and reasoning tools.

    d. Insure that environment available in future concurrent architectures is also available in networks of generic workstations.

Dale A. Prouty
Philip Klahr
Inference Corporation
Los Angeles, CA. 90045

ABSTRACT

A workstation is being developed that provides
a computational environment for all NASA engineers[1]
across application boundries, which automates
reuse of existing NASA software and designs, and
efficiently and effectively allows new programs/
designs to be developed, catalogued, and reused.
The generic workstation is made "domain specific"
by specialization of the user interface, capturing
engineering design expertise for the domain, and
by constructing/using a library of pertinent
information. The incorporation of software
reusability principles and expert system technology
into this workstation provide the obvious benefits
of increased productivity, improved software use
and design reliability, and enhanced engineering
quality by bringing engineering to higher levels
of abstraction based on a well tested and class-
ified library.
Keywords: software workstation, software reuse,
design reuse

INTRODUCTION

The workstation applies artificial intelligent (AI)
automated reasoning technology to the problems
associated with efficient construction of pro-
cedural software. The designs and procedural
software developed are catalogued or classified
for reuse in later design efforts, so that the
two main goals of the workstation, software reuse
and engineering design automation, are achieved.
At this writing, the project is underway but not
yet completed. The completion of the project is
approximately two years hence, and follows six
months of initial work. Thus this is a report
on the progress, approach, and conceptual
principles being applied in the early phases of
effort.
Software reuse is not a new theme. Much research
effort has gone into understanding the problems
and appropriate approaches for software reuse[1-3]
The feasibility of this workstation concept is
enhanced by incorporating the fruits of that
research. Two noteworthy conceptual results are
that software should be "reused" at the highest
level of abstraction possible and that addressing
narrow problem domains lends the greatest potential
for success.

This workstation incorporates these results first,
by providing a domain-specific "requirements
language" so that workstation interactions may
proceed from engineering requirements, and second
through supporting customizations of a general
purpose shell for specific problem domains.

The highest possible level of engineering software
design automation is one where requirements are
input and executable code produced with only an
essential amount of user interaction to assure
accurate mapping of 1) requirements into a formal
specification of the problem, and 2) implementation
of the specification. If the implementation is
constrained to use a very limited set of existing
code then the user interactivity may be further
minimized, but with the obvious drawback of
constraining design flexibility based on the
available software.

APPROACH TO SOFTWARE REUSE

Two general categories of software reuse have been
discussed [1], "building blocks" and "transforma-
tion". This workstation is a hybrid attempt at
initially supporting use of existing software
subroutines, but providing an increasingly more
useful transformational capability as the work-
station evolves to achieve the end result of
maximizing reuse through transformation which is
generally believed to be the more successful route
to effective software reuse.

A transformation approach, simply put, reuses
transformational capabilities as opposed to
existing modules that are pre-canned to fit needs.
A simple example being applied in the early phases
of the workstation is in computer mathematics/
algebra packages. These programs, SMP[2] and
MACSYMA[3], can take analytic mathematic express-
ions and generate procedural code for them auto-
matically in Fortran.

[2] SMP-Symbolic Manipulation Program
[3] MACSYMA of Symbolics, Inc.

The Fortran code may be regenerated based on incrementally changed specifications of the analytic expression, i.e. based on reuse of the transformational capabilities of these mathematics programs.

THE DESIGN PROBLEM

The basic engineering approach to design is one of stating problem requirements in a requirements language, then appropriately translating this language into a formal specification language. This translation process typically requires interactive user involvement to assure correct understanding of the intended requirements, both by the user and the system. Then based on this formal specification, an implementation process ensues to produce source code in traditional programming languages, such as Fortran, C, or Ada.

The entire design process of requirements to implemented code may be captured in a "design data structure" which supports problem description at multiple levels of abstraction and implementation simultaneously. It is not necessary for the entire problem description or design process to be at the same level of specification or implementation concurrently. It is possible that some parts of the design would be fully implemented, some fully specified, and others still at various levels of definition and abstraction. The reuse of previous designs then can be achieved based on stored data structures.

The general design data structures support rule-based search of various solution pathsfrom requirements to specification and specification to implemented code, or, that is, various designs. A finalized design consists of a particular path through the data structure, and the result is an executable application. Various parts of the data structure are saved by users for future design efforts so that previously encountered designs, specifications, or implementations may be recognized and linked in as new design scenarios are developed.

REQUIREMENTS LANGUAGE

Successful design is only achieveable if the user requirements are clearly understood. To make the workstation successful at understanding the stated problem, the language available to the user for expression should be limited in vocabulary and structure. Thus, the workstation requirements definition language covers a narrow domain with restricted input syntax. Users thereby avoid having to learn a general purpose language that is foreign to their problem domain. These narrow domain or limited purpose languages can themselves by structurally based on a general purpose language, but the individual user does not have to be aware of this property.

TOWARD FORMAL SPECIFICATION

A parser accepts input requirements and translates them into specification. The language is used to allow identification of both the domain objects mentioned in the requirements and the constraints imposed on them. Once the objects are identified, the specified constraints together with stored knowledge-based constraints restrict the translation to formal specification. Also, the

workstation can suggest types of requirements that might allow further specification to occur and indentify ambiguous or contradictory inputs. Eventually, the user has accurately stated the requirements for formal specification. The translation then implements requirement in a formal specification language.

SPECIFICATION LANGUAGE

The specification language allows for modelling of multiple levels of problem abstraction and their refinement in domain specific areas. The language used has a special purpose syntax to assure accurate representation. The specification language includes constructs for a "what-description" and a "how-description" of engineering problems. The "what-description" constructs allow domain objects to be represented. The "how-description" constructs allow problem solving strategies to be represented and are automated as much as possible from stored domain expertise.

IMPLEMENTATION METHODS

The method of implementation of the engineering specification can be manual, by general transformation, based on domain-specific rules, or in the simplest case by direct correspondence with library functionality. The implementation approach taken can lead to different implementations of the same specification. A simple example of this is where library subroutines and modules (sets of subroutines) are both available. The implementation may then piece together explicitly required subroutines, or use the more general but less efficient higher level modules.

Once the specification has been implemented in a language (possibly intermediate) it may be desirable to translate this code into a target language, such as Ada, C, or Fortran.

During the implementation phase, a history of effort is maintained so that high level operations may be performed when possible. For example, it should be possible to make incremental changes to the specification and determine the effect of the changes to minimize reimplementation efforts.

STORED DESIGN DATA STRUCTURES

The stored design data structures take several forms including subroutines, various levels of abstracted logical structures (flow charts or block diagrams), and pieces of specification. A library of existing (Fortran) subroutines are classified into a library DBMS for reuse and perusal. Additional data structure elements are classified into the system to support design work until a sufficient amount of engineering domain expertise had been collated to support a specific domain of design activity. The adequate classification of library information is a fundamental basis for successful reuse of existing elements. Both keyword systems and general knowledge-based structures retaining information about the design data structure elements stored are utilized.

CURRENT WORKSTATION EFFORTS

Initial efforts are a demonstration of concepts to software development automation. The efforts during this phase result in delivery of all essential parts of the workstation design just discussed - a simple requirements language, translation to formal specification, specification language, and implementation in Fortran. The breadth of capabilities is limited in scope as might be expected. The objective is to provide NASA with a clear demonstration of an approach to software reuse and engineering design automation and could result in a fully developed and deployable system. The initial workstation is being developed on the Symbolics[4] Lisp Machine using ART[5] Automated Reasoning Tool.

The requirements language supports a very restrictive syntax for a narrow application domain - NASA's Flight Design Software. The narrow language allows requirements statement such that design constraints may be propagated and domain objects identified. A knowledge engineering effort incorporates rule-based constraints for the users based on NASA designer expertise in an effort to assist in the requirements to specification translation process.

The initial specification language consists of a graphical-flow language. Given an initial limited selection of available library routines, an engineer will not always succeed in full problem specification. After all, this is a software "development" workstation. This will be the typical state of affairs during engineering design. Some specified need would be unavailable. However, if further refinement of the specification is possible, then the workstation may be able to suggest several pieces of this lower level specification supported by library or design date structure functionality.

Finally, the code implementation of the workstation is directly in Fortran. The workstation supports automatic handling of all routine interfacing and executable module linking. Also, it is possible to request certain results presentation facilities for plotting, output, etc.

For demonstration purposes the "workstation shell" is utilized by the design team of knowledge engineers, to build a shell customization for NASA Flight Design engineers. Efforts at domain specific knowledge engineering are being maximized to demonstrate technological capabilities for a rule-based approach to Flight Design Software use.

[4]Symbolics, Inc.

[5]ART-Automated Reasoning Tool of Inference Corp., L.A., CA.

SUMMARY AND CONCLUSIONS

A Workstation targeted at improving software reusability is under development for NASA. This workstation is to accomplish the goals of engineering design automation and effective software reuse. The approach taken integrates expert system technology and software reusability principles in a hybrid reuse system which encompasses both reuse of building blocks of existing subroutines, as well as transformational capabilities to generate code in traditional programming languages such as Fortran for engineering designs. The workstation is customized for narrow engineering disciplines to allow reasonable expectations of success.

REFERENCES

(1)   IEEE Software Engineering, Special
        Issue on Software Reusability,
        Vol. SE-10 No.5, September 1984.
(2)   IEEE Software Engineering, Special
        Issue on Artificial Intelligence
        and Software Engineering, Vol.
        SE-11, No.11, November 1985.
(3)   IEEE Computer, Design for Adaptibility,
        Vol. 19, No. 2, February 1986.

AI Tools in Computer Based Problem Solving

Arthur J. Beane
Digital Equipment Corporation

July 31, 1986

The use of computers to solve value-oriented, deterministic, algorithmic problems, has evolved a structured life cycle model of the software process. The symbolic processing techniques used, primarily in research, for solving non-deterministic problems, and those for which an algorithmic solution is unknown, have evolved a different model, much less structured. Traditionally, the two approaches have been used completely independently.

With the advent of low cost, high performance 32-bit workstations executing identical software with large minicomputers and mainframes, it became possible to begin to merge both models into a single extended model of computer problem solving.

This paper describes the implementation of such an extended model on the Digital Equipment Corporation VAX family of micro/mini/mainframe systems. Examples in both development and deployment of applications involving a blending of AI and traditional techniques will be given.

Networking & AI Systems: Requirements & Benefits

I. OPEN SYSTEMS require Networks

- Evolving multi-vendor systems   (IBM, DEC, Lotus)

- Inconsistent knowledge & databases   (DB2, IMS, dBASE)

- Decentralized decision making   (international)

II. DELIVERY SYSTEMS require Networks

- access to corporate databases

- integration with desktop applications

- price/performance optimization

III. Foundations for CONCURRENT ARCHITECTURES

- Resource sharing   (network services)

- Synchronization   (protocols)

- Load balancing   (flow control)

- Fault Tolerance   (error recovery)

IV. Example Applications

- Training   (CLUG RBBS)

- Development   (GHC, Beckman, Honeywell)

- Delivery   (Perkin Elmer, PMS)

The price performance benefits of network systems is well documented. The ability to share expensive resources sold timesharing for mainframes, department clusters of minicomputers, and now local area networks of workstations and servers.

In the process, other fundamental system requirements emerged. These have now been generalized Open System requirements for hardware, software, applications and tools. The ability to interconnect a variety of vendor products has led to a specification of interfaces that allow new techniques to extend existing systems for new and exciting applications.

As an example of the a message passing system, local area networks provide a testbed for many of the issues addressed by future concurrent architectures: synchronization, load balancing, fault tolerance and scalability.

Gold Hill has been working with a number of vendors on distributed architectures that range from a network of workstations to a hypercube of microporcessors with distributed memory. Results from early applications are promising both for performance and scalability.

An Expert System for Natural Language Processing

John F. Hennessy
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342

Abstract

This paper proposes a solution to the natural language processing problem that uses a rule-based system, written in OPS5, to replace the traditional parsing method.

The advantages to using a rule-based system are explored. Specifically, the extensibility of a rule-based solution is discussed as well as the value of maintaining rules that function independently. Finally, the power of using semantics to supplement the syntactic analysis of a sentence is considered.

Introduction

Traditional approaches to natural language processing are based upon standard compiler technology. That is, the language to be parsed is first defined. Then attempts are made to match data, entered in the form of sentences, to the structure that is derived from the language definition. The primary deficiency of this method is demonstrated when a sentence that does not match the predefined structure is encountered: the parse fails - the sentence cannot be "understood". This can occur when either a valid sentence is input that has not been accounted for or when a non-grammatical sentence is entered.

Overview

The system I am developing consists of three phases: a parser, an error corrector, and a semantic analyzer. The parser, written in Lisp, attempts to parse the input sentence using purely syntactic rules. The error corrector and the semantic analyzer, both written in OPS5, are designed as expert systems. The error corrector is invoked if the parser fails. After an attempt is made to correct the sentence, control is returned to the parser. The output of the parser is given to the semantic analyzer.

Implementation Strategy

During the parsing phase, morphological analysis is performed and dictionary definitions for the individual words making up the sentence are extracted from the lexicon. The grammar definitions are maintained in a separate file to ease maintenance. The definitions are deliberately kept as simple as possible. Consequently, both the error corrector and semantic analysis modules handle the sentence analysis process. The defined grammar for the parser is as follows:

    S = NP + VP
        A sentence consists of a noun
        phrase and a verb phrase

    NP = SNP + {PP}*
        A noun phrase consists of a simple
        noun phrase and any number of
        optional prepositional phrases

    SNP = {DET} + {ADJ}* + Noun
        A simple noun phrase consists of an
        optional determiner (a, an, the),

any number of optional adjectives,
                    and a noun

          PP = Prep + SNP
                    A prepositional phrase consists of
                    a preposition and a simple noun
                    phrase

          VP = Verb + {NP}
                    A verb phrase consists of a verb
                    plus an optional noun phrase

where braces indicate optional items and the asterisk
indicates zero or more occurrences.

The words in the lexicon are provided with a list of
features, both syntactic and semantic, which includes the
part of speech and other features, such as whether it is
an animate or inanimate object, whether countable, or if
color applies. No attempt is made to define an
orthogonal or complete set of features.

The parser also maintains a list of all noun phrases
encountered and passes that list to the semantic analyzer
in order to resolve pronoun references.

The task of the error corrector is to restructure the
original input into a grammatically acceptable form,
using relatively simple rules. For example, if two
consecutive nouns appear in the sentence, then treat the
first noun as an adjective (e.g., "The house boat is
docked"). Other rules handle such things as misplaced
modifiers (e.g., "In the car the man plays a piano").
Some rules just throw out duplicate words - a common typo
(e.g., "The man in in the boat"). Note that these rules
are not necessarily restricted to syntax. That is, a
semantic rule determined that "in the car" modified "man"
and not "piano" in the previous example.

Many of the rules defined in the error corrector handle

                              727

conditions that could be processed by other means. For example, an ATN (augmented transition network) is often implemented in the parser. An ATN is a process of transformations that preserves the meaning of the sentence and checks for consistent features, such as verb agreement. An ATN, for example, could provide a mechanism for the parser to convert an imperative sentence into the equivalent declarative sentence. In the system I am developing, imperative sentences are handled in the error corrector by a rule which inserts the word "you" in front of sentences beginning with a non-question verb. Although ATNs have been shown to handle many cases, they complicate the parsing. Consequently, ATNs have been avoided in my research.

The semantic analysis module relies on a frame-based representation of the dictionary. This module resolves pronoun references using the noun history maintained by the parser. This module also attempts to resolve ambiguities and determines the "correctness" of the sentence (e.g., rejecting "Colorless green ideas sleep furiously"). The semantic analysis module shares many rules with the error corrector module. For example, a syntactically valid version of the above sentence, "The man plays a piano in the car" yields the same result: it is the man, not the piano, who is in the car.

Summary

I chose OPS5 as the primary implementation language for my research because it is a rule-based, forward-chaining language. Thus, it maps directly to the processing paradigm I am developing. Although OPS5 does not directly support the frame-based structure needed for the dictionary representation, I implemented this using a

small set of rules.

OPS5 seems to be a good choice not only because of its rules, but also because of the ease with which it interacts with other languages. Rules are independent thus, new rules can easily be added to expand the analysis capability. Two major benefits result. First, the control structure is never modified, thus providing a system that is easier to maintain and to extend. Second, rules for semantic analysis can be executed in conjunction with the syntactic rules. Rules are invoked as the sentence structure dictates, not as the parsing algorithm demands. By treating the natural language processing problem in this manner, it is possible to concentrate on the analysis without getting bogged down in the implementation details.

The success of the current system, as tested by the successful interpretation of the examples cited in this paper and other test cases, indicates that my initial premise is correct. That is, natural language understanding can be modeled by an expert system, enabling syntactic and semantic analysis to be combined in a manner analogous to the way natives of the language process text.

As more rules are added to the expert system portions, it seems clear that the grammar required by the parser can be kept to a very small subset of valid English. The error corrector and semantic analysis modules may be able to be combined because they share many common rules.

Although my research is ongoing, I believe that the expert system technology offers a major step in solving the natural language processing problem.

References

Barr, Avron and Feigenbaum, Edward A., The Handbook of Artificial Intelligence, Vol. 1, William Kaufman, Los Altos, CA, 1981.

Brownston, Lee et al., Programming Expert Systems in OPS5, Addison-Wesley, Reading, MA, 1985

Feigenbaum, Edward A. and Feldman, Julian (editors), Computers and Thought, McGraw-Hill, New York, 1963

Harris, Mary Dee, Introduction to Natural Language Processing, Reston Publishing, Reston, VA, 1985.

Keenan, Edward L., Formal Semantics of Natural Language, Cambridge University Press, London, 1975

Leister, Mark, Introductory Transformational Grammar of English, Holt, Rinehart and Winston, New York, 1971

Partee, Barbara H., Montague Grammar, Academic Press, New York, 1976

Expert System Technology as a Data Processing Tool

John F. Hennessy
Digital Equipment Corporation
5775 Peachtree Dunwoody Road
Atlanta, Georgia 30342

Abstract

This paper focuses on the expert system technology as a data processing tool. Several existing applications are analyzed. Their original definition as pure expert systems is contrasted with their current status as integrated systems. The architectural requirements needed to support such a heterogeneous environment are given.

Introduction

A pure expert system consists solely of rule-based processing. The solution can be expressed entirely in terms of the AI tool used to implement the expert system. Systems of this type are often advisory systems: a user poses a questions and receives an answer. In contrast, an integrated system contains rule-based processing, but its rules are supplemented by routines that are common to traditional data processing. Often the inputs come from another process, such as a process control system. The outputs may be required by another program, which need not be an expert system.

This paper examines three problems which required an AI solution, but which had special requirements that necessitated integrated system solutions.

731

Overview of Problem 1

In 1978, when Digital Equipment Corporation began shipping the first VAX computers (11/780), they realized that they were facing a potential problem. Digital was, and still is, in the business of providing custom computer systems - computers built to the needs of the customer. No two orders are alike. Then each order required an engineer to check the configuration to insure that the machine could be built; that the specified configuration could be supported; and that the order was complete (e.g. contained sufficient cabling, power, etc.). This was a time-consuming, people-intensive task. If a method was not found to speed up the process, either DEC's ability to ship systems in a timely manner would be impeded or DEC would have to resort to offering standard configurations.

A decision was made to automate the checking task. This was attempted using traditional programming techniques. The task, though seemingly well defined, proved too difficult to implement: the data was constantly changing - more parts were becoming available, while other parts were changing or becoming obsolete. In addition, for any given order, there were multiple "correct" configurations. Understandably, the initial project failed.

The AI Solution

Carnegie-Mellon University proposed attacking the problem by building an expert system, and Digital funded the effort. In December 1978, development began at CMU. A

732

research prototype, consisting of 250 rules, was demonstrated the following April. In January 1980, Digital began using the system for all VAX orders and in January 1981, full development commenced.

The result was XCON, a system for configuring computers. It was developed using a rule-based expert system implemented in OPS5, because the problem could not be solved traditionally. Two earlier attempts failed, not because the problem was not understood, but because the problem was ill-defined: requirements were constantly changing and, for any given system, many configurations were correct. Now that the problem has been solved, it is possible to look at the solution and identify areas that do have algorithmic solutions. As a result, XCON today, although still primarily written in OPS5, has sections written in seven traditional programming languages. Hence, it is now an integrated system.

Overview of Problem 2

Digital Equipment Corporation, after developing the XCON system, which insured that orders could be manufactured and supported, needed to aid its sales force in generating correct orders. The problem was that a sales representative could write an order that appeared to meet the customer's requirements, but that could omit certain components required for a complete system. That is, the sales rep could err in configuring a complete system, but the omission might not be obvious. If an order were submitted and later found by XCON to have errors, that information would have to be transmitted back to the sales representative and then back to the customer and a new corrected order placed. This resulted in time delay and frustration for the customer and for Digital.

733

The AI Solution

The basic functionality of XCON, a batch system, was taken and given an interactive, menu-driven front end. Consequently, the sales representative can specify a generic part, such as a particular class of disk drive, and the system will automatically generate the proper part plus all the necessary additional parts (for example, the correct disk controller). As a result, orders are being placed with a higher degree of accuracy than was possible before.

Overview of Problem 3

A major bank needed a system to help transportation managers minimize float by quickly clearing "non-on-us" checks, that is, checks drawn against other financial institutions. The current method required a transportation manager to analyze reports of the previous months' business. Assuming the same mix of checks for the current month, he had to determine the quickest way to clear the checks. Furthermore, he had to decide which checks should be sent to the issuing bank, to an area Federal Reserve, or to a local Federal Reserve branch. The problem involved the volume of checks (both in number and dollar amount), various processing charges, flight schedules, and interest rates. Although the problem could have been handled by conventional programming techniques, the volume of data made any real-time value of the information unlikely. When the non-numeric factors were entered - bad weather, flight delays, airport closings - no traditional system could handle the problem. An expert system was the only solution.

734

The AI Solution

A prototype expert system was designed that accepted the check information as it became available. The system stored the various presentation charges of the other banks and the Federal Reserve, accessed data from the Official Airline Guide (OAG), and accepted data changes (e.g., a cancelled flight) from the transportation managers. This data, combined with the heuristics, the expert knowledge, of the transportation managers, resulted in a prototype system that demonstrated that the difficult task could be handled.

Summary

Each of the above problems has been solved with an expert system using a variety of AI tools. A large part of each system is implemented using standard programming tools that supplement the AI code. That is, the XCON system uses standard database query languages to extract part information from a networked database. The XSEL system makes heavy use of a forms interface to interact with the user. The banking system uses Fortran for numerical calculations.

The key to each of these systems is that, although developed as expert systems, they were implemented on a traditional, general-purpose computer architecture. Thearchitecture provides easy interface with other languages while allowing the use of conventional development aids, such as code and module management tools, debuggers, and performance analyzers.

735

The   expert system technology allows significant problems
to be solved. Futhermore, the expert systems are useful
because they easily fit into the standard data processing
environment.

References

Feigenbaum,   Edward A. and McCorduck, Pamela, The Fifth
Generation, Addison-Wesley, Reading, MA, 1985

Harmon, Paul and King, David,   Expert Systems, John
Wiley, New York, 1985

Hayes-Roth, Frederick et al., Building Expert Systems,
Addison-Wesley, Reading, MA, 1983

Scown, Susan J., The Artificial Intelligence Experience:
An Introduction, Digital Press, Maynard, MA, 1985

Waterman, Donald A., A Guide to Expert Systems,
Addison-Wesley, Reading, MA, 1986

Next-Generation Space Manipulator

P. Brunson, W. Chun, and P. Cogeos
Advanced Automation Technology (Robotics) Group
Martin Marietta Denver Aerospace

Abstract

In 1977, Martin Marietta Corporation, Denver Division, designed and built the Protoflight Manipulator Arm (PFMA) for Marshall Space Flight Center. It is one* of two space-qualified manipulators, the other being the Shuttle Remote Manipulator System (RMS). Since then, technology has advanced considerably in terms of components such as electric motors, control electronics, materials, sensors, and end effectors.

This paper will present a conceptual design for the next-generation manipulator of space applications. The next-generation manipulator and the PFMA will be described in more detail. These differences could have a major influence on the construction, testing, and performance of a space arm. Assessed in detail are these technologies and their effect on the design.

Servicing is an important goal of robotics in space. Parameters such as environment, type of task, time sequence, and dexterity will affect the arm and its ability to accomplish its mission. Requirements such as these are important considerations in the design of the next-generation space arm.

Introduction

The PFMA[1] is a very good design for a space manipulator. Unfortunately, it was never flown and is now nearly ten years old. At the time, the arm exemplified state-of-the-art technology. For its particular size, its use proved its principle, however, it lacked an adequate control system and architecture.
This problem is now being reversed by the Intelligent Robotics Systems Study (IRSS) program (see Fig. 1).

The PFMA (Fig. 2) is a general-purpose manipulator with distributed actuators. This paper outlines the kinematics, components, drive technology, arm segments and end effector, manufacturing/assembly, tests, and performance necessary to upgrade the arm to become the next-generation space manipulator.

Kinematics

Seven degrees-of-freedom (DOF) have proven useful in avoiding the singularities inherent in a 6-DOF arm. When adding that seventh DOF, there are three possibilities[2]: (1) a 4-DOF wrist, (2) an upper arm roll, or (3) a pitch/yaw elbow. The PFMA has an upper arm roll. This removes the majority of the singularities and enables the arm to work in both a horizontal plane as well as a vertical plane. Figure 3 is a schematic of the proposed kinematics, including a compact wrist.

The work envelope of the wrist could be improved on. By using distributed actuators, the three axes of the wrist cannot be constructed concurrently. This results in having the wrist being neither compact nor dexterous. As an improvement, we suggest using a wrist design of Mark Rosheim[3] (see Fig. 4). The wrist has all three axes concurrent. The design is simple and well engineered. The motors for the wrist are located in the forearm. By relocating the motors closer to the base of the manipulator, its dynamics are improved.

*The arm was built to flight specifications and one of the motor drives completed space qualifications.
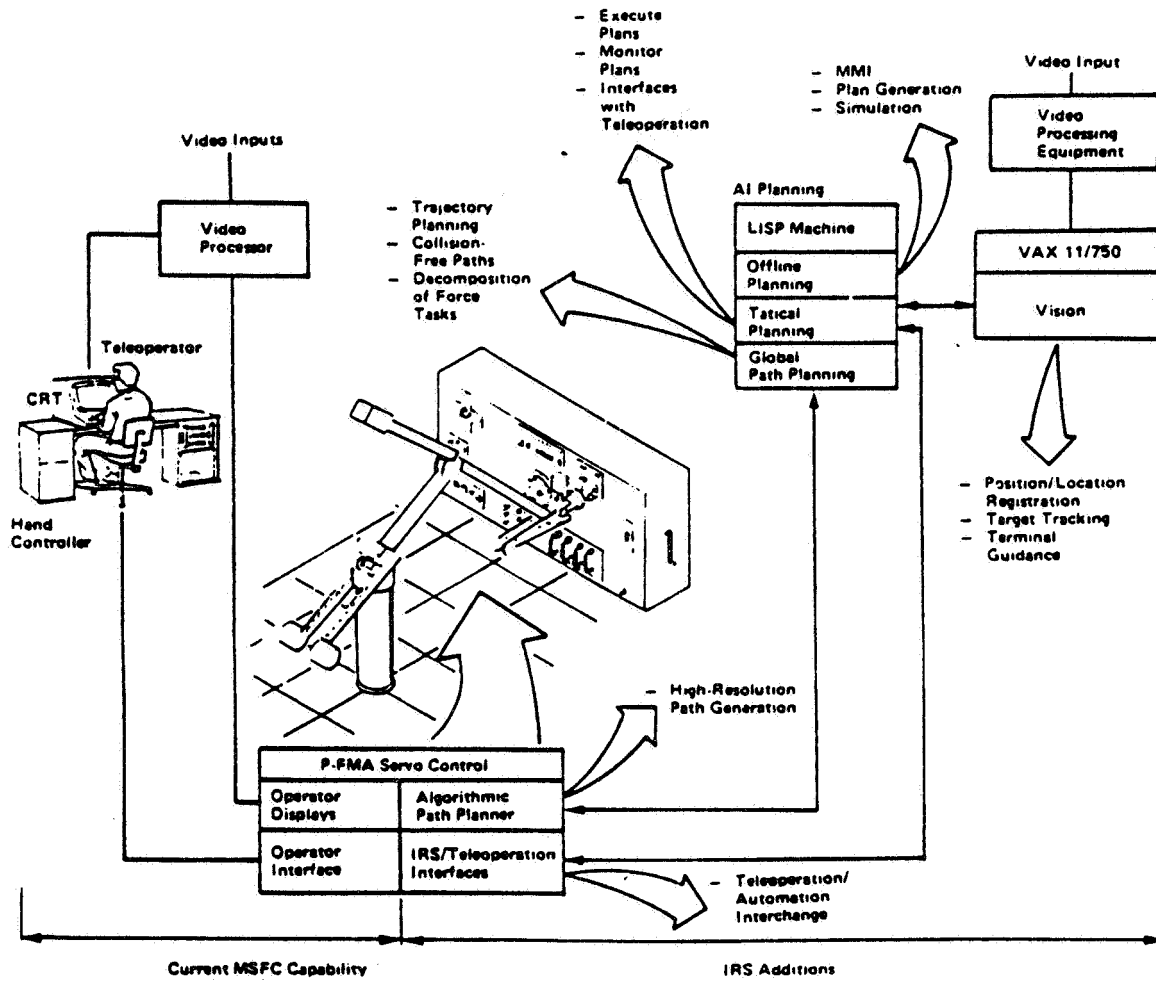
737
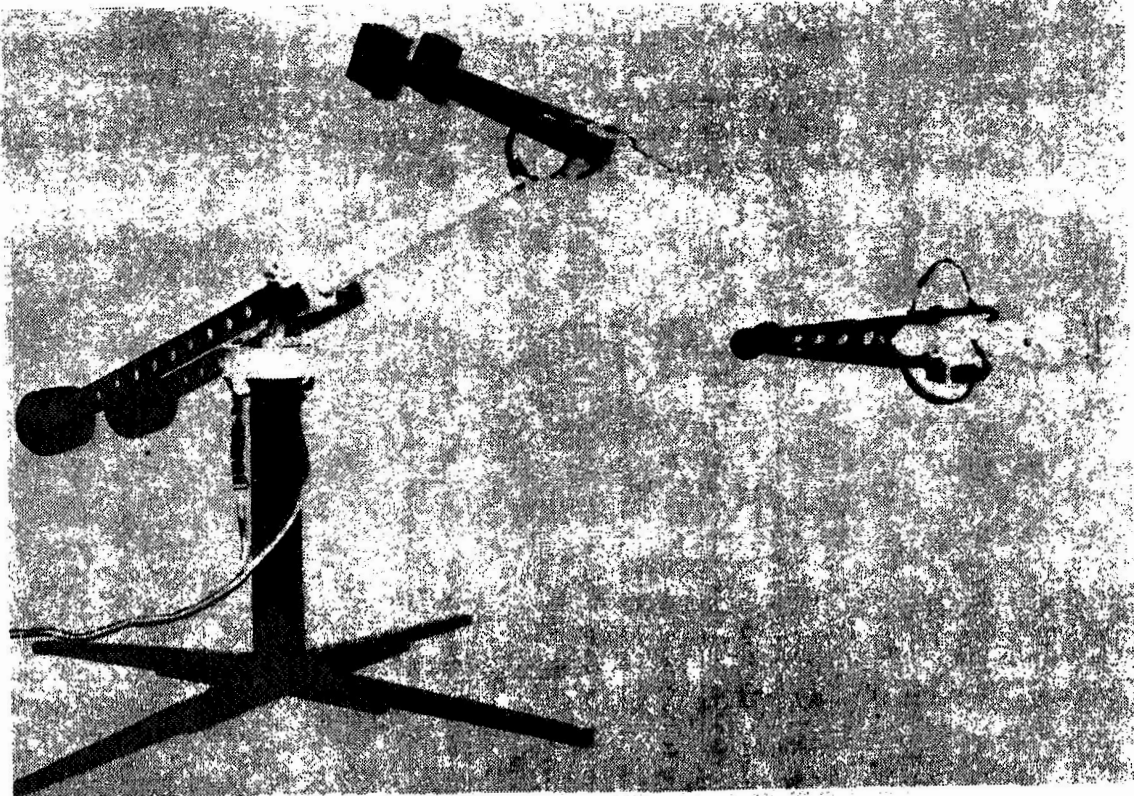
Figure 1 IRSS Program

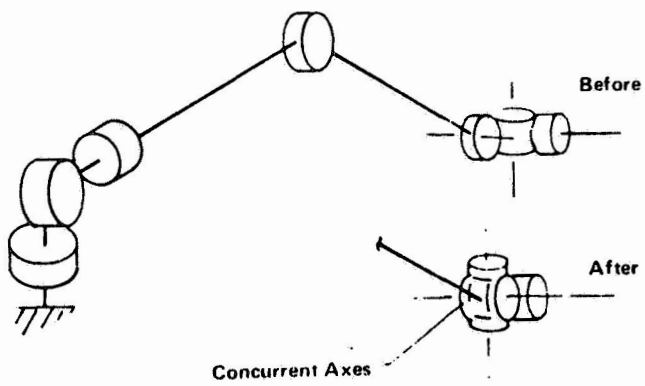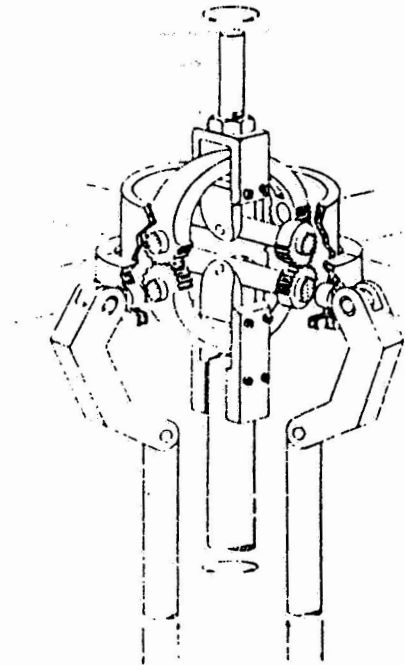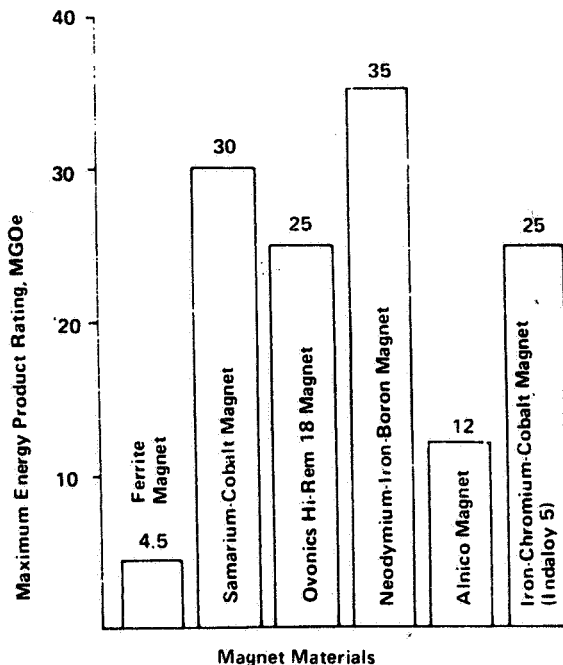*Figure 2 The PFMA*

Figure 3 Arm Kinematics

Figure 4 Rosheim Wrist

739

Components

The PFMA uses a brush-type, d-c motor using Samarium-Cobalt magnets. Another rare-earth magnet, Neodymium-Iron-Boron[4] promises more torque for the same frame size. Figure 5 compares the maximum energy product of several magnet materials. Although Neodymium-Iron-Boron has the highest rating, its output has not been consistent and will require further testing. Samarium-Cobalt is still the standard for high-torque motors.

Figure 5 Maximum Energy Product of Different Magnet Materials

The motor should be brushless. With no brushes to wear, reliability will increase, while periodic maintenance will be reduced. Brush debris will be eliminated as well as arcing, which is a common problem with brush motors.

There are several devices that can be used to provide joint angle information to the arm controller. These are:

1) Brushless resolvers,
2) Inductive couplings,
3) Optical encoders.

Each of these devices can be designed with high, inherent accuracies to enable precise end point positioning. None of them contain rubbing surfaces, such as brushes, that would introduce frictional torque, limited life resulting from wear, or become a source of electrical noise. Being analog devices, both the resolver and inductive coupling require analog-to-digital conversion electronics, which can reduce the accuracies of these devices somewhat. The optical encoder provides a digital output and therefore does not suffer this problem. All of these devices can be qualified for space use and indeed have been used in this environment before. Packaging for each is available in a variety of configurations and would not limit their use.

At this time, the next-generation space arm would not use tachometers at each of the manipulator joints; instead, rate information will be derived from the individual joint angle position sensors. This approach will reduce the complexity of each drive along with its overall size and weight. In addition, the wire count will be reduced by at least two per drive along with the reduction of power to operate tachometers as a result of not using these devices.

The low level of torque required to backdrive each of the joint actuators will necessitate a friction brake so each drive can be locked when required. These brakes must exhibit fast response, long life, and no backlash when engaged. Simple devices using spring-energized friction pads are capable of generating large torques and braking energy from a small package. These devices are fail-safe in that the spring ensures brake lockup in the event of a power failure. An electrically activated coil and armature provide the release power to allow joint rotation. Brake and drive power will be controlled individually to allow each joint to "freewheel" as desired. This characteristic has been very useful during tasks such as aligning and engaging a close fitting pin or dowel.

Drive Technology

The dual-path transmission is a mature technology. One motor drives two identical gear trains that are sprung against each other at the final output ring gear (see Fig. 6). The preloading of the gears is accomplished through the use of a split gear hub. The result is a very high-precision gear train without backlash.

A manipulator such as the next-generation space arm will use a host of electrical components and subsystems that require power to operate plus receive/send signals back to the controller or operator's console. Supporting these components requires a large number of wires that must be run, in some instances, the entire length of the arm. Routing these wires along or through the manipulator links poses no real problem. The difficulties arise when the wires must be routed across the bend and roll joints. This must be done in such a fashion that it does not restrict joint range of motion, degrade the quality of the signal passing through the conductor, or introduce excessive torque that the joint drive must overcome. Slip rings have been used, but are often a source of electrical noise, friction, and limited life. A highly reliable substitute for the slip ring in limited rotation applications is a twist capsule. These devices eliminate contact resistance variation because there is no sliding contact. Flexible tapes provide the connection between the rotating and stationary circuits. The tapes provide constant circuit resistance that is not degraded by wear, shock, or vibration. The twist capsule's life can be measured in the millions of cycles, can be fabricated from low outgassing materials, and good signal isolation can be achieved.[5] Figure 7 illustrates how a twist capsule might be incorporated into the joint drive package.

All gear design, especially precision gear designs, will be depreciated or even rendered completely useless by improper choice of material. The primary objective of the gear material in the next-generation space arm will be to provide machinability to obtain a precise profile and then to retain this precision throughout the gear's life against stress, wear, and environmental effects. Desirable gear properties include the following qualities:

741

o Machinability,
o Stability,
o Surface finish,
o Rigidity,
o Wear Resistance,
o High Strength,
o Fatigue Resistance,
o Shock Resistance,
o Corrosion Resistance,
o Temperature Stability,
o Internal Damping,
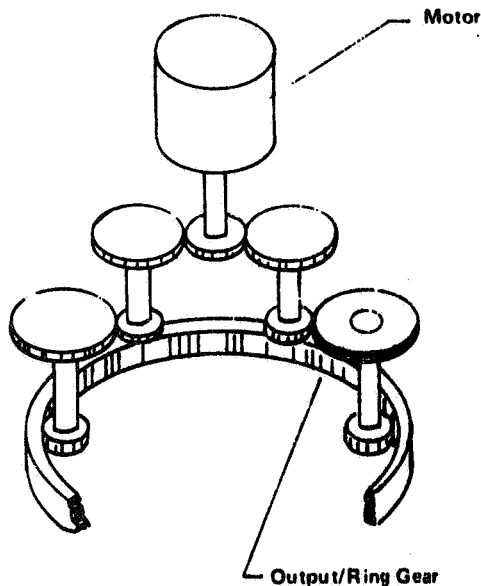o Cost,
o Availability.
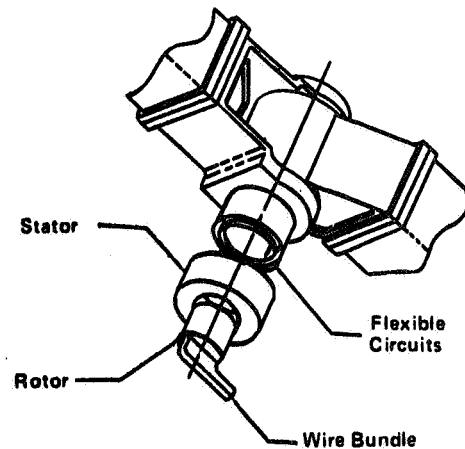

Figure 6 Dual-Path Transmission

Figure 7 Twist Capsules

In precision gear trains, the first five items are the most important, while some tradeoffs can be made with the other properties.[6]

This paper will make no attempt to define what the optional material will be for each of the joint drive gears. The above information is presented to ensure the reader understands the many variables to be considered during the gear design process. Each must be carefully evaluated to ensure a drive design with the strength, life, and precision required of a manipulator in a space environment.

Bearings are a key element in the design of any high-precision mechanism. They can influence a number of drive requirements including:

1) Internal alignments and relationships,
2) Transmission errors,
3) Drive stiffness,
4) Drive stiction and friction,
5) Load-carrying capacity,
6) Life.

Torque motors and joint angle sensors of the type used within each drive require very small clearances between rotating and nonrotating parts.

742

Excessive shaft runouts or free play would introduce rubbing contact, both increasing drive friction and damaging the devices.

The quality of the gears used (AGMA 12) in the reduction path demand shafts that run true and introduce no backlash or transmission errors. Drive stiffness must be controlled to ensure all resonances are well above the operating bandwidth of the of the system. Stiction and friction levels must be minimized to maximize drive controllability. All of these requirements are influenced by the bearing suspension system selected. Bearing characteristics that must be evaluated before selection include:

1) Tolerances,
2) Preloading,
3) Load rating,
4) Raceway geometry,
5) Lubrication,
6) Bearing fit-up,
7) Retainer design.

Each of these characteristics can affect one or more of the drive requirements and therefore must be carefully evaluated.

In addition, drive housing and shaft designs must share the same degree of precision as the bearing to fully exploit its capabilities. Accommodations in the design must also be made to ensure proper bearing operation throughout the temperature envelope without loss of precision or stiffness.

Arm Segments and End Effector

To provide a viable space-rated robotic arm, there are certain design criteria that must be met for the arm to function optimally.

The arm segments should include the following:

1) Be lightweight with high-strength and stiffness characteristics.
2) Allow for a "clean" method to route wires.
3) Neatly and compactly house the actuators and positioning sensors.
4) Contain joints that are simple and quickly engaged and disengaged; they should lock rigidly together.
5) Be designed to be modular.

The following is proposed to accomplish these goals:

1) The arm should be made of a mixture of continuous and noncontinuous magnesium graphite composite; continuous where machining is not necessary and noncontinuous where it is.
2) The arm segments should be hollow to allow for the wires to be run through their centers, keeping them hidden yet accessible.
3) There are several acceptable joint designs; two of the most promising designs feature a bayonnet mount or a bulkhead mount.

For the manipulator to be a viable tool, it must have a general-purpose gripper and yet be able to use several different servicing tools. Articulated hands are experimental and are only found in the laboratory. Their technology is not mature and requires additional considerations for housing the finger actuators thus complicating the wrist interface.

The most advanced gripper is the "Smart End Effector" developed by JPL[7]. The end effector incorporates an integral force sensor at its base and the jaw combines tactile with proximity data. The intermeshing jaw provides excellent

prehension. In addition to the gripper, the arm requires a system that incorporates different tools. Martin-Marietta has developed a power takeoff system with an extra motor built in the gripper that powers an assortment of compatible tools. As a result, the arm needs only the one integrated motor for all its tools. SRS Technologies[8] has taken a deeper investigation into the necessary tools for servicing.

Manufacturing/Assembly

Martin Marietta has been machining and assembling these types of drives since the mid 1970s. They have gained the technical expertise necessary to construct the various stages and to assemble the parts in a clean room to tight tolerances. Each step is very critical and built to exact specifications. Any inconsistencies or missed detail could degrade the performance of the drive. As a result, the drives are rugged and durable. Similarly, the arm segments are inspected. All devices are tested at each subassembly level. The final product is a flight-worthy system.

Testing

Testing of the arm is accomplished at two stages: (1) drive and (2) complete manipulator. The individual drives are characterized separately. Its performance is checked by studying torque versus speed curves and torque versus current curves. The plots are differentiated by varying the input voltage, load, or pulse-width modulation (PWM) signal. The tests are conducted through an Electromechanical Test Support Unit (ETSU), a Martin Marietta-designed test bench that is reconfigurable to perform a variety of tests.

In addition to performance testing, each drive is tested for mechanical and electrical friction at running and breakaway levels. The next test is to measure the drive compliance. Having passed performance tests, the drives are put through random vibration and thermal-vacuum testing. The prototype drive has one additional test: life cycle in a thermal chamber.

The next level of testing is at the assembled stage. The drive tests are repeated at the systems level. Moreover, a thermal balance test and an electromagnetic interference/compatibility test are performed. Complementing the characterization tests will be several robotic tests[9] to obtain the following data:

1) Geometrical values
   a) Workspace,
   b) Static behavior,
   c) Position accuracy,
   d) Path accuracy,
   e) Overshoot,
   f) Reproduction of the smallest steps,
   g) Synchronous travel accuracy,
   h) Long-term behavior;
2) Kinematic values
   a) Cycle Time,
   b) Speed,
   c) Acceleration;
3) Power and Noise Values;
4) Dynamic Values
   a) Force,
   b) Dynamic compliance,
   c) Dynamic behavior of the structure.

744

The extensive testing qualifies the design and the hardware for its mission in space.

Performance

Because of the increased strength of the motor magnets, the same diameter motor will produce greater torque. If the current torque values are sufficient, the design may favor smaller and lighter drives. Either way, the drive will produce a higher torque-to-weight ratio. Additionally, elimination of the tach generator will lighten the drive and make it more compact.

The mature, dual-path gear train exhibits zero backlash without compromising friction. The improved bearing design removes radial play and thus creates a stiffer drive. Using high-quality gears and precision assembly techniques, the gear tooth stresses can be distributed uniformly. Gear wear is reduced and thus its life extended. Brushless motors further extend life and reduce maintenance.

The approximately 100 to 1 gear ratio is very backdriveable. The higher resolution position sensor is combined with the IRSS control architecture, resulting in a high-performance arm (Fig. 8).
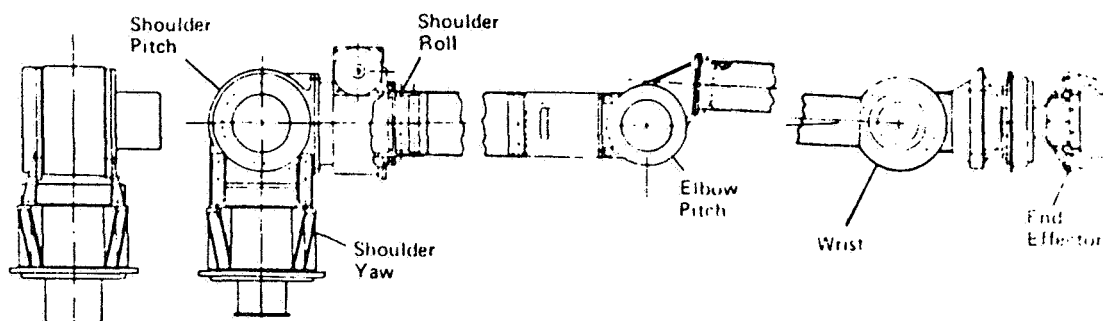

Figure 8 Next-Generation Space Manipulator

Conclusion

Upgraded components increase the performance of the individual drives. The final product is a very precise positioning arm that is characterized by a high torque-to-weight ratio. Similar upgraded drives have been built, tested, and space-qualified by Martin Marietta.

A compact wrist gives the manipulator increased dexterity to maneuver in its work envelope. Advanced materials make the arm segments stiffer without sacrificing weight or a slim profile. New concepts in attaching the various components lends the arm to a modular approach. Moreover, the use of twist-capsules eliminates noises that are inherent in existing slip rings.

The elimination of the tach generator and the placement of the drive electronics close to each drive reduces the number of power and signal wires. The smaller wire bundles can now be routed more efficiently.

Finally, the incorporation of the aforementioned upgrades on the PFMA will result in a very high-performance manipulator system for space. Combined with the control system and computer architecture of the IRSS program, the arm can be available in a very short timeframe. It has the potential to be the next space manipulator.

Acknowledgments

References

1. Protoflight Manipulator Arm (PFMA). MCR-77-201, Final Report, Martin Marietta, Denver Division, April 1977.

2. J.M. Hollerbach: "Optimum Kinematic Design for a Seven Degree-of-Freedom Manipulator." Preprints 2nd International Conference, Robotics Research, Kyoto, Aug. 19-23, pp 349-356.

3. M. Rosheim: Robot Wrist Development. NASA Conference on Robotics and Flexible Automation, Jan. 14, 1986.

4. K. Dekker: "New Materials for Improved Motor Designs." Motion, May/June 1986, pp 6-13.

5. Poly-Twist Capsules Catalogue. Poly-Scientific, Blacksbury, Virginia.

6. G. Michalec: Precision Gearing: Theory and Practice. John Wiley and Sons, 1966.

7. A. Bejczy, E. Kan, and R. Killion: "Integrated Multisensory Control of Space Robot Hand." Preprint Proceedings of the AIAA Guidance and Control Conference, Colorado, Aug. 19-21, 1985.

8. J. Cody: Interchangeable End Effector Tools Utilized on the PFMA-Task 1 Final Report. Refer to Marshall Space Flight Center, MFS-27125.

9. H. Warnecke, R. Schraft, and M. Wanner: "Performance Testing." Handbook of Industrial Robotics, Chapter 10. Simon NOF (editor), John Wiley and Sons.

TECHNICAL REPORT

1. REPORT NO. NASA CP-3007	2. GOVERNMENT ACCESSION NO.	3. RECIPIEN

4. TITLE AND SUBTITLE Second Conference on Artificial Intelligence for Space Applications	5. REPORT DATE August 1988
	6. PERFORMING ORGANIZ/

7. AUTHOR(S) Compiled by Thomas Dollman	8. PERFORMING ORGANIZATION REP

9. PERFORMING ORGANIZATION NAME AND ADDRESS George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812	10. WORK UNIT, NO. M-577
	11. CONTRACT OR GRANT NO.
	13. TYPE OF REPORT & PERIOD COVERED

12. SPONSORING AGENCY NAME AND ADDRESS National Aeronautics and Space Administration Washington, DC 20546	Conference Publication
	14. SPONSORING AGENCY CODE

15. SUPPLEMENTARY NOTES Conference Coordinator - Thomas Dollman, Information and Electronic Systems Lab., Marshall Space Flight Center, Alabama Co-sponsored by The University of Alabama in Huntsville

16. ABSTRACT

Proceedings of a conference held at Huntsville, Alabama, on November 13-14, 1986. This Second 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 Artificial Intelligence Computer Vision Design Data Capture Robotics Space Station Automation	18. DISTRIBUTION STATEMENT Unclassified/Unlimited Subject Category: 61

19. SECURITY CLASSIF. (of this report) Unclassified	20. SECURITY CLASSIF. (of this page) Unclassified	21. NO. OF PAGES 752	22. PRICE A99

National Aeronautics and
Space Administration
Code NTT-4

Washington, D.C.
20546-0001

NASA

POSTMASTER: If Undeliverable (Section 158
Postal Manual) Do Not Return