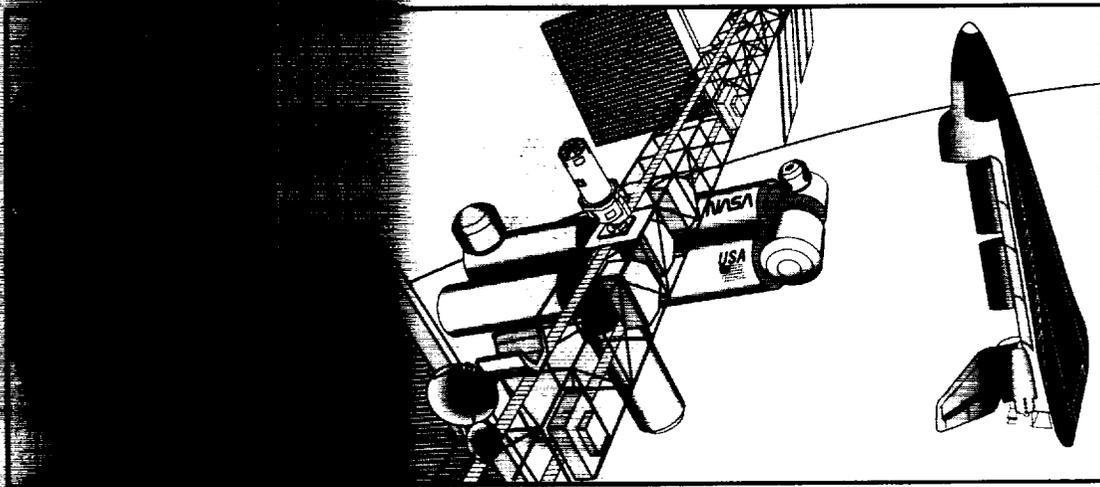


NASA Conference Publication 3073

Conference on Artificial Intelligence for Space Applications

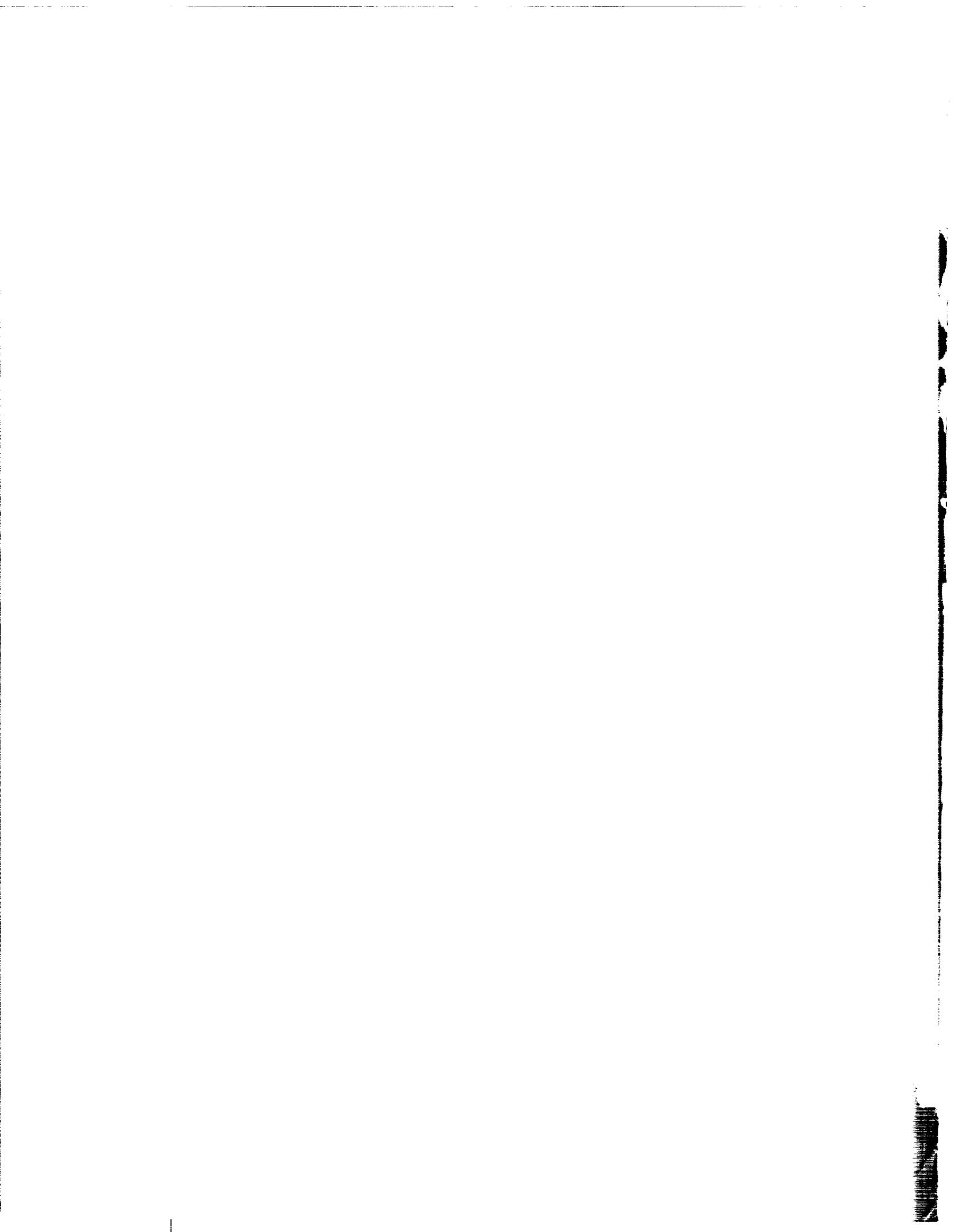


(NASA-CP-3073) PARTIAL COPY
ARTIFICIAL INTELLIGENCE FOR SPACE
APPLICATIONS (NASA) 3073

CSCL 098

CONFERENCE
PROCEEDINGS
NPO-27577
Unclass
0293560

81/51



NASA Conference Publication 3073

Fifth Conference on Artificial Intelligence for Space Applications

Compiled by

S. L. O'Dell

*George C. Marshall Space Flight Center
Marshall Space Flight Center, Alabama*

Proceedings of a conference sponsored by
the University of Alabama in Huntsville,
the IEEE Computer Society/Huntsville Chapter,
the AIAA Alabama-Mississippi Section, and
the National Aeronautics and Space Administration
and held in Huntsville, Alabama
May 22-23, 1990

NASA

National Aeronautics and
Space Administration

Office of Management

Scientific and Technical
Information Division

1990

Foreword

This document constitutes the proceedings of the Fifth Conference on Artificial Intelligence for Space Applications, which was held at the Von Braun Civic Center in Huntsville, Alabama on May 22-23, 1990. This document, in conjunction with the proceedings from the four previous conferences in this series, demonstrates that the National Aeronautics and Space Administration (NASA) has made significant progress in applying Artificial Intelligence (AI) techniques to solve some difficult problems in the space domain. The papers which have been presented and published via this conference-series represent an increasing number of implementations within the aerospace community and an even larger range of potential applications.

During the five years spanned by this conference-series, it has been demonstrated that AI technologies have provided useful and powerful tools which have been effectively used to solve certain classes of problems. These technologies will play an even more significant role in near-term and evolving space systems by further enhancing human performance and enabling humans to carry out more complex tasks in both flight and ground-support environments. It is important to continue to validate and implement applications of these technologies in order to provide a foundation for activities required for more ambitious, long-term, future space missions and the infrastructures needed to support those missions.

NASA's Marshall Space Flight Center (MSFC) and the University of Alabama in Huntsville (UAH) have jointly sponsored this conference, with co-sponsorship from the Huntsville Chapter of the Institute of Electrical and Electronics Engineers and the Alabama-Mississippi Section of the American Institute of Aeronautics and Astronautics, to establish a forum to discuss possible solutions to the complex technical requirements for execution and support of complex space missions. Additionally, the sponsors sought to provide an opportunity for those who apply AI methods to space-related problems to identify common goals, to compare the effectiveness of the various approaches being employed, and to discuss issues of general interest. Towards these ends, the program committee selected sixty-two papers for presentation in eighteen technical sessions which cover a broad range of topics related to AI software systems and to robotics and vision hardware systems.

This conference would not have been possible without the dedicated efforts of many people. First, we wish to thank the authors whose research, development, and implementation efforts are presented here. Second, we thank the members of all the committees who planned and executed the numerous activities required for a conference such as this one. We thank the exhibitors for their efforts to develop and demonstrate tools and environments for implementing many of the ideas expressed during this conference. And finally, we wish to thank the invited speakers, lecturers, and others from the aerospace community whose interest in applying artificial intelligence to space domains makes this conference both possible and meaningful.

The sponsors hope that these proceedings will be useful as a reference to some selected AI activities in progress, and will contribute to the literature of AI applications.

Thomas S. Dollman
Gary L. Workman

Fifth Conference on Artificial Intelligence for Space Applications

General Chair

Tom Dollman, NASA/MSFC *Gary Workman, UAH*

Program Chair

Caroline Wang, NASA/MSFC *James Johannes, UAH*

Program Committee

<i>Dale Thomas, NASA/MSFC</i>	<i>Bernard Schroer, UAH</i>
<i>Robert Linner, NASA/MSFC</i>	<i>Martin Hofmann, UAH</i>
<i>Bill Selig, NASA/MSFC</i>	<i>Steve Floyd, UAH</i>
<i>James Steincamp, NASA/MSFC</i>	<i>Daniel Rochowiak, UAH</i>
<i>Steve Purinton, NASA/MSFC</i>	<i>Pat Ryan, UAH</i>
<i>Linda Brewster, NASA/MSFC</i>	<i>Donnie Ford, UAH</i>
<i>Bryan Walls, NASA/MSFC</i>	<i>Sara Graves, UAH</i>
<i>Michael Whitley, NASA/MSFC</i>	<i>Mike Meehan, UAH</i>
<i>Tim Crumbley, NASA/MSFC</i>	<i>Warren Moseley, UAH</i>
<i>Dave Weeks, NASA MSFC</i>	<i>Mary Weisskopf, UAH</i>
<i>James Carnes, Boeing AI Center</i>	<i>Mike Brady, UAH</i>

Exhibits

Donnie Ford, UAH

Tutorials

Warren Moseley, UAH

Publications

Linda Brewster, NASA/MSFC *Steve O'Dell, NASA/MSFC*

Conference Management

Karen Mack, UAH *Ann Yelle, UAH* *Jill Roumeliotis, UAH*

Table of Contents

Planning and Scheduling

AGENT INDEPENDENT TASK PLANNING <i>William S. Davis</i>	1
SPIKE: ARTIFICIAL INTELLIGENCE SCHEDULING FOR HUBBLE SPACE TELESCOPE <i>Mark Johnston, Glenn Miller, Jeff Sponsler, Shon Vick, and Robert Jackson</i>	11
RESOURCE ALLOCATION USING CONSTRAINT PROPAGATION <i>John S. Rogers</i>	19

Auto Control and Testing

DEVS-BASED INTELLIGENT CONTROL OF SPACE ADAPTED FLUID MIXING <i>Sung-Do Chi and Bernard P. Ziegler</i>	25
DYNAMIC TEST INPUT GENERATION FOR MULTIPLE-FAULT ISOLATION <i>Phil Schaefer</i>	33
GENETIC ALGORITHM BASED FUZZY CONTROL OF SPACECRAFT AUTONOMOUS RENDEVOUS <i>C. L. Karr, L. M. Freeman, and D. L. Meredith</i>	43

Real-Time and Process Control

THE APPLICATION OF INTELLIGENT PROCESS CONTROL TO SPACE BASED SYSTEMS <i>G. Steve Wakefield</i>	53
ARTIFICIAL INTELLIGENCE IN THE MATERIALS PROCESSING LABORATORY <i>Gary L. Workman and William F. Kaukler</i>	59
AN ARCHITECTURE FOR INTELLIGENT TASK INTERRUPTION <i>D. D. Sharma and Srinivasa Narayan</i>	69

Knowledge-Based System and Knowledge Representation

KNOWLEDGE REPRESENTATION FOR COMMONALITY <i>Dorian P. Yeager</i>	85
A KNOWLEDGE-BASED APPROACH TO CONFIGURATION LAYOUT, JUSTIFICATION AND DOCUMENTATION <i>F. G. Craig, D. E. Cutts, T. R. Fennel, C. Case, and J. R. Palmer</i>	95
KIPSE1 - A KNOWLEDGE-BASED INTERACTIVE PROBLEM SOLVING ENVIRONMENT FOR DATA ESTIMATION AND PATTERN CLASSIFICATION <i>Chia Yung Han, Liqun Wan, and William G. Wee</i>	103
AN ARCHITECTURE FOR RULE-BASED SYSTEM EXPLANATION <i>T. R. Fennel and J. D. Johannes</i>	113

Managing AI Technology Development Tools

AN AUTOMATED TOOL FOR THE DESIGN AND ASSESSMENT OF SPACE SYSTEMS <i>Lois M. L. Delcambre and Steve P. Landry</i>	121
ATS DISPLAYS – A REASONING VISUALIZATION TOOL FOR EXPERT SYSTEMS <i>William John Selig and James D. Johannes</i>	129
ESTABLISHING A COMMUNICATIONS-INTENSIVE NETWORK TO RESOLVE ARTIFICIAL INTELLIGENCE ISSUES WITHIN NASA'S SPACE STATION FREEDOM RESEARCH CENTERS COMMUNITY <i>E. Davis Howard III</i>	139
EXPERT SYSTEM DEVELOPMENT METHODOLOGY (ESDM) <i>Charisse Sary, Lewey Gilstrap, and Larry G. Hull</i>	147

Automation for Space Station – I

A HYBRID APPROACH TO SPACE POWER CONTROL <i>E. W. Gholdston, D. F. Janik, and K. A. Newton</i>	155
A STUDY ON DIAGNOSABILITY OF SPACE STATION ECLSS <i>S. Padalkar, W. Blokland, and J. Sztipanovits</i>	165
ATTITUDE DETERMINATION AND CONTROL SYSTEM (ADCS) MAINTENANCE AND DIAGNOSTIC SYSTEM (MDS): A MAINTENANCE AND DIAGNOSTIC SYSTEM FOR SPACE STATION FREEDOM <i>David Toms, George D. Hadden, and Jim Harrington</i>	175
KNOWLEDGE-BASED SYSTEMS AND NASA'S SOFTWARE SUPPORT ENVIRONMENT <i>Tim Dugan, Cora Carmody, Kent Lennington, and Bob Nelson</i>	185

Automation for Space Station – II

SPACE STATION FREEDOM ECLSS – A STEP TOWARD AUTONOMOUS REGENERATIVE LIFE SUPPORT SYSTEMS <i>Brandon S. Dewberry</i>	193
SIMULATION-BASED INTELLIGENT ROBOTIC AGENT FOR SPACE STATION FREEDOM <i>Csaba A. Biegl, James F. Springfield, George E. Cook, and Kenneth R. Fernandez</i> ...	203
GRAPHICAL EXPLANATION IN AN EXPERT SYSTEM FOR SPACE STATION FREEDOM RACK INTEGRATION <i>F. G. Craig, D. E. Cutts, T. R. Fennel, and B. Purves</i>	211
SPACE STATION ADVANCED AUTOMATION <i>Donald Woods</i>	221

Distributed Environments

A DEVELOPMENT FRAMEWORK FOR ARTIFICIAL INTELLIGENCE BASED DISTRIBUTED OPERATIONS SUPPORT SYSTEMS	
<i>Richard M. Adler and Bruce H. Cottman</i>	231
A KNOWLEDGE-BASE ARCHITECTURE FOR DISTRIBUTED KNOWLEDGE AGENTS	
<i>Joel Riedesel and Bryan Walls</i>	241
CREATURE CO-OP: ACHIEVING ROBUST REMOTE OPERATIONS WITH A COMMUNITY OF LOW-COST ROBOTS	
<i>R. Peter Bonasso</i>	257

Design Knowledge Capture

DESIGN KNOWLEDGE CAPTURE FOR A CORPORATE MEMORY FACILITY	
<i>John H. Boose, David B. Shema, and Jeffrey M. Bradshaw</i>	271
CAPTURING DESIGN KNOWLEDGE	
<i>Brian Babin and Rasiah Loganantharaj</i>	281
AN APPLICATION OF DESIGN KNOWLEDGE CAPTURED FROM MULTIPLE SOURCES	
<i>Preston A. Cox and John H. Forbes</i>	291

Impact of AI on Software Engineering

THE INTEGRATION OF AUTOMATED KNOWLEDGE ACQUISITION WITH COMPUTER-AIDED SOFTWARE ENGINEERING FOR SPACE SHUTTLE EXPERT SYSTEMS	
<i>Kenneth L. Modesitt</i>	301
CASE-BASED REASONING IN DESIGN: AN APOLOGIA	
<i>Kirt Pulaski</i>	305
ARTIFICIAL INTELLIGENCE SOFTWARE ENGINEERING (AISE) MODEL [ABSTRACT]	
<i>Peter A. Kiss</i>	315

Vision and Robotics

DETECTING PERCEPTUAL GROUPINGS IN TEXTURES BY CONTINUITY CONSIDERATIONS	
<i>Richard J. Greene</i>	317
A VISION-BASED TELEROBOTIC CONTROL STATION	
<i>Brian Tillotson</i>	325
THE REAL-TIME LEARNING MECHANISM OF THE SCIENTIFIC RESEARCH ASSOCIATES ADVANCED ROBOTIC SYSTEM (SRAARS)	
<i>Alexander Y. Chen</i>	331

Fault Diagnosis and Maintenance

A DIAGNOSIS SYSTEM USING OBJECT-ORIENTED FAULT TREE MODELS <i>David L. Iverson and F. A. Patterson-Hine</i>	341
MODEL FOR A SPACE SHUTTLE SAFING AND FAILURE DETECTION EXPERT <i>Daphna Zeilingold and John Hoey</i>	351
OBJECT-ORIENTED FAULT DIAGNOSIS SYSTEM FOR SPACE SHUTTLE MAIN ENGINE RED- LINES <i>John Rogers and Saroj Kumar Mohapatra</i>	361

Automatic Programming

AN APPLICATION GENERATOR FOR RAPID PROTOTYPING OF ADA REAL-TIME CONTROL SOFTWARE <i>Jim Johnson, Haik Biglari, and Larry Lehman</i>	373
AN ENGINEERING APPROACH TO AUTOMATIC PROGRAMMING <i>Stuart H. Rubin</i>	383
AUTOMATED EXTRACTION OF KNOWLEDGE FOR MODEL-BASED DIAGNOSTICS <i>A. J. Gonzales, H. R. Myler, M. Towhidnejad, F.D. McKenzie, and R. R. Kladke</i>	393
DERIVATION OF SORTING PROGRAMS <i>Joseph Varghese and Rasiah Loganantharaj</i>	403

AI Issues in Simulation, Modeling, and Tutoring

A STRUCTURE FOR MATURING INTELLIGENT TUTORING SYSTEM STUDENT MODELS <i>Willard M. Holmes</i>	421
A TOOL FOR MODELING CONCURRENT REAL-TIME COMPUTATION <i>D. D. Sharma, Shie-rei Huang, Rahul Bhatt, and N. S. Sridharan</i>	429
CONSTRUCTION OF DYNAMIC STOCHASTIC SIMULATION MODELS USING KNOWLEDGE-BASED TECHNIQUES <i>Douglas Williams and Sajjan G. Shiva</i>	439
THE COMPOSITE LOAD SPECTRA PROJECT <i>J. F. Newell, H. Ho, and R. E. Kurth</i>	457

Knowledge Acquisition and Intelligent Assistants

INTELLIGENT TUTORING SYSTEMS FOR SPACE APPLICATIONS <i>Carol A. Luckhardt-Redfield</i>	467
DOCUMENTATION AND KNOWLEDGE ACQUISITION <i>Daniel Rochowiak and Warren Moseley</i>	477
INDUCTIVE KNOWLEDGE ACQUISITION EXPERIENCE WITH COMMERCIAL TOOLS FOR SPACE SHUTTLE MAIN ENGINE TESTING <i>Kenneth L. Modesitt</i>	487
VIP: A KNOWLEDGE-BASED DESIGN AID FOR THE ENGINEERING OF SPACE SYSTEMS <i>Steven M. Lewis and Kirstie L. Bellman</i>	497

Knowledge-Base / Data-Base Integration

USING DECISION-TREE CLASSIFIER SYSTEMS TO EXTRACT KNOWLEDGE FROM DATABASES <i>D. C. St. Clair, C. L. Sabharwal, Keith Hacke, and W. E. Bond</i>	507
DEB: A DIAGNOSTIC EXPERIENCE BROWSER USING SIMILARITY NETWORKS <i>Cyprian E. Casadaban</i>	517
ARTIFICIAL INTELLIGENCE TECHNIQUES FOR MODELING DATABASE USER BEHAVIOR <i>Steve Tanner and Sara J. Graves</i>	525

AI Systems in Teleoperation and Robotics

RESOLUTION OF SEVEN-AXIS MANIPULATION REDUNDANCY: A HEURISTIC ISSUE <i>I. Chen</i>	535
THE SIXTH GENERATION ROBOT IN SPACE <i>A. Butcher, A. Das, Y. V. Reddy, and H. Singh</i>	545
ROBOT DYNAMICS IN REDUCED GRAVITY ENVIRONMENT <i>Gary L. Workman, Tollie Grisham, Elaine Hinman, and Cindy Coker</i>	551

ADA as an AI Language

COMPILING KNOWLEDGE-BASED SYSTEMS FROM KEE TO ADA <i>Robert E. Fillman, Conrad Bock, and Roy Feldman</i>	557
AUTOMATED UNIT-LEVEL TESTING WITH HEURISTIC RULES <i>W. H. Carlisle, Kai-Hsiung Chang, J. H. Cross, W. Keleher, and K. Shackelford</i>	567
SDI SATELLITE AUTONOMY USING AI AND ADA <i>Harvey E. Fiala</i>	577
ADA AS AN IMPLEMENTATION LANGUAGE FOR KNOWLEDGE-BASED SYSTEMS <i>Daniel Rochowiak</i>	589
AUTHOR INDEX	599

Agent Independent Task Planning[†]

William S. Davis
Boeing AI Center
PO Box 240002, MS JA-74
Huntsville, AL 35824-6402
net: bill@huntsai.boeing.com

ABSTRACT

Agent-Independent Planning is a technique that allows the construction of activity plans without regard to the "agent" that will perform them. Once generated, a plan is then validated and translated into instructions for a particular agent, whether a robot, crewmember, or software-based control system. Because Space Station Freedom (SSF) is planned for orbital operations for approximately thirty years, it will almost certainly experience numerous enhancements and upgrades, including upgrades in robotic manipulators. Agent-Independent Planning provides the capability to construct plans for SSF operations, independent of specific robotic systems, by combining techniques of object-oriented modeling, nonlinear planning and temporal logic. Since a plan is validated using the physical and functional models of a particular agent, new robotic systems can be developed and integrated with existing operations in a robust manner. This technique also provides the capability to generate plans for crewmembers with varying skill levels, and later apply these same plans to more sophisticated robotic manipulators made available by evolutions in technology.

1. Introduction

Space Station Freedom is planned for orbital operations for approximately thirty years. Over the long life of this complex structure, it will almost certainly experience numerous enhancements and upgrades, including upgrades in robotic manipulators. Great potential for robotic automation exists in the areas of housekeeping, laboratory science, maintenance, and safety, as well as various EVA functions. Throughout this 30-year period the types of robotic manipulators available for these areas, as well as the capabilities they provide, will continuously evolve with changes in technology. On the contrary, basic procedures for intra- and extra-vehicular activity, once established, will remain relatively static. As advances in technology produce more sophisticated manipulators that are capable of performing more complicated tasks, robots may become responsible for more detailed operations. However, for these advancements in technology to be beneficial to Space Station Freedom, any robotic upgrades should be *compatible* with existing procedures.

Programming different robots for the same task is a redundant job that should be avoided. Such programming can be very labor-intensive, not to mention the job of verifying that the "new" robots are still compatible with the "old" tasks. Some tedious chores that crewmembers perform today are the duties that may be carried out by the robots of tomorrow. Thus, the robotic plans that will be developed for future on-board robots should be compatible with crew procedures that are established in the interim. Persons who are currently responsible for composing such crew

procedures may eventually be tasked to compose these procedures for robots as well. In this regard, prescribing activities for robots should be as similar as possible to prescribing activities for crewmembers.

A technique known as "Agent-Independent Planning" has been developed for addressing the above issues. Planning is determining a course of action to achieve some goal. Task planning is determining a sequence of tasks (the course of action) to be performed by an agent to achieve some desirable state (the goal) in the agent's world. "Agent-Independent Planning" is a method of automated planning that allows the generation of task plans from a set of goals, without having to be concerned with constraints imposed by the agent that will execute the plan. In the domain of Space Station Freedom (SSF), these plans can be considered a sequence of tasks for intra-vehicular and extra-vehicular operations activity. Plans, or operations procedures, are developed by considering general constraints on the planning environment and task sequences. For execution of these procedures, the plans are translated into the specific operations language of a particular agent. This methodology allows plans and their environment to be modeled in a fashion that separates different classes of constraints into independent sets.

A prototype of such a system has been developed at Boeing Aerospace and Electronics that creates agent-independent plans for SSF maintenance and repair operations. The system translates these plans into (a) code which is executed by a robot, (b) software commands which drive a graphical robotic simulator, or (c) English sentences (output through a voice synthesizer) which describe crew procedures. The actual planning mechanism is based on Chapman's TWEAK [3], but the representation incorporates Allen's time logic [1] and hierarchical abstraction [8]. Hogue integrates a temporal interval-based planner in the domain of Qualitative Physics [4], compiling plan operators by matching descriptions of an agent with descriptions of the domain. The system presented in this paper performs along similar lines to match an agent's capabilities with the needs of a plan. This combination is validated to ensure the agent can perform the given plan.

This paper begins with an explanation of the plan representation in terms of the object-oriented model of the plan environment, the temporal relationships among tasks in the plan, and how these two representations are abstracted into a task library. Once this foundation is established, a discussion ensues concerning the generation of agent-independent plans, and the steps necessary to translate these plans for a specific agent.

2. Plan Representation

In order to have agent-independent planning, one must develop a representation for the plans which is free from the agent who will perform the plan, but also which can later be transformed into a representation that includes the agent. Such a representation must model the world in segments which can be connected and disconnected to identify various aspects of the world. We accomplish this by decomposing the planning environment into three distinct entities: Tasks, Agents, and Objects. Thus, tasks are actions on objects, agents are the performers of actions, and objects are the recipients of actions. The agent-independent plan is built combining tasks and objects, and then the transformation is made to agent-dependency by incorporating knowledge about a specific agent. Obviously, "action" is the common denominator of these three entities. In a robotic realm, these actions represent physical motion. A primitive action is defined to be that which represents basic physical motion, such as locomotion, rotation, limb movement (extension, retraction, lateral/horizontal/diagonal movement), etc. Figure 1 lists the primitive actions currently employed in the planning environment. Models of the planning environment (whether agents, objects, or tasks) all relate to this set of primitives. These actions, or primitive tasks, provide the common

interface between agents and the planning environment. The agents' and objects' physical and functional constraints are represented in terms of preconditions and effects on these primitive tasks. Object-oriented models of agent and object properties allow descriptions of agents and objects to be combined through inheritance, as will be shown in the following section.

rotate clockwise	move to	move left	grasp	push	raise
rotate counter-clockwise	carry to	move right	release	pull	lower

Figure 1. Primitive Actions Employed in Planner

2.1. Object-Oriented Environment

Agents and objects are composed of properties which are ordered hierarchically, with lower levels inheriting the properties of higher levels, and communication between them is done in message-passing fashion. This hierarchical ordering allows complex, real-world descriptions of items in the environment. In terms of the Space Station Freedom maintenance domain, this means that complex items such as thermal control systems can be described by the union of the properties of their components (such as pumps, valves, pipes, etc.). Tasks, agents and objects are modeled with primitive actions as a "connecting point". In terms of this paper, task information declares which objects are affected by action (and possibly in what order the action is to occur), information about agents declares which actions they can perform and to what capacity, and object information specifies the manner in which objects can be affected by actions.

<p><u>Object: pcb / Task: PUSH(Distance?)</u></p> <p><i>subgoals:</i></p> <p>pcb . attached-p = no Slot? . occupied = no Slot? . parent-object = P-Obj? P-Obj? . power-status = off pcb . coordinates = Slot? . observation-pt pcb . depth = Distance?</p> <p><i>main-effects:</i></p> <p>pcb . attached-to = Slot?</p> <p><i>side-effects:</i></p> <p>pcb . attached-p = yes pcb . coordinates = Slot? . coordinates</p>	<p><u>Object: pcb / Task: PULL(Distance?)</u></p> <p><i>subgoals:</i></p> <p>pcb . attached-p = yes Slot? . parent-object = P-Obj? P-Obj? . power-status = off pcb . attached-to = Slot? pcb . depth = Distance?</p> <p><i>main-effects:</i></p> <p>pcb . attached-p = no</p> <p><i>side-effects:</i></p> <p>pcb . coordinates = Slot? . observation-pt pcb . attached-to = nil Slot? . occupied = no</p>
---	---

Figure 2. Functional Constraints for Printed Circuit Board

Agents and objects are represented as a mixture of characteristics describing their physical properties and functional capabilities. Such physical properties include size, mass, relative position, etc. while functional capabilities are constraints based on the primitive actions, as well as physical properties of agents and objects. These functional constraints are based on typical planning constraints, such as the preconditions/subgoals and effects found in Wilkins' SIPE [11]. This allows each agent or object to respond differently to the same primitive action according to the manner in which it is modeled. Figure 2 shows an example of the object constraints for pushing or pulling a printed circuit board. It is interesting to note how the constraints work to somewhat

specialize these generic actions into "remove" and "insert" actions. Pattern-matching is employed to allow generic constraints to be tailored for specific objects. In the figure, variables are represented by symbols appended with a question mark. These variables are instantiated in the planning process, and then the constraints are used to determine truth satisfaction for including the action in the plan. Functional constraints on an agent work in a similar fashion.

Also included in the definition of an agent are the specific instructions necessary for it to perform a primitive action. It is this set of instructions that will be used to transform the plan representation from agent-independent to agent-dependent. Agent characteristics are arranged in a hierarchy such that properties of higher level are inherited by lower levels. For example, in figure 3 all the properties associated with limbs of motion are inherited by both arms and legs. In turn, arms and legs add their own distinguishing properties and capabilities which are inherited by their lower levels (which inherit the things from the limbs of motion characteristic as well). Thus to describe a Puma-560 robot with one arm, incorporating vision, force/torque, and tactile sensors, one needs to include in the robot's description the corresponding characteristics and then add the features particular to the Puma-560 basic unit. Objects are represented in similar fashion. However, whereas agents were classified on the grounds of their capabilities to effect action, objects are classified according to how they are affected by action; the difference is one of *activity* versus *passivity*. For instance, the object classifications, or characteristics, for a printed-circuit-board (which is mounted on some rack) would include "replaceable object" and "fragile object". "Replaceable object" would be derived from "removable object" (which has constraints that inhibit certain actions on the object when "in-place"), while adding its own information concerning the location of replacement objects, the location to discard used objects, etc.

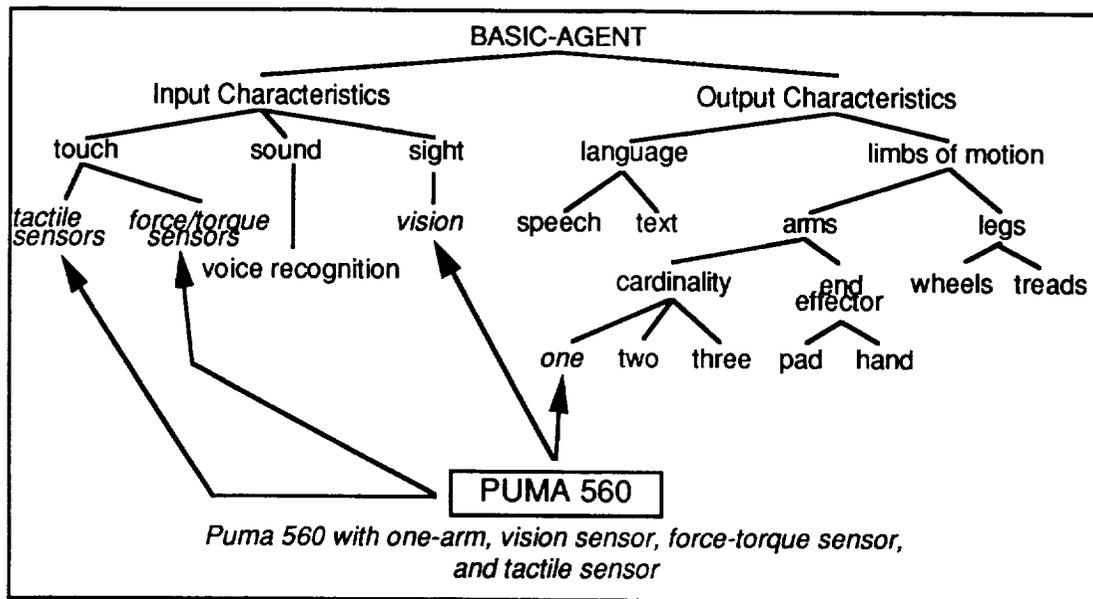


Figure 3. Example Agent Hierarchy

2.2. Task Abstraction

As was stated earlier, tasks declare which objects are affected by action. Actions are composed of sub-actions. For instance, *replace pump* may be broken into *isolate pump*, *remove pump*, *discard pump*, *get a new pump*, and *install pump*. Primitive actions (see figure 1) correspond to primitive tasks, and are used to create higher-level tasks. Higher-level tasks are

composed of previously defined higher-level and/or primitive tasks arranged in some fashion (see section 2.3 for this arrangement). Thus, a (theoretically) infinitely large task library can be developed using this means of abstraction. This capability provides for a dynamic planning environment. That is, the plan that solves some goal today may well be composed of different tasks when trying to solve the same goal tomorrow (assuming new tasks are being added to the library). This allows the system to improve incrementally. By introducing plan variables, tasks then become predications on objects as they are parameterized, such as *remove(pump-27)*. The argument to a task can also be an object characteristic (as defined above) in order to describe generic, or template, tasks for use in describing more abstract tasks. This is analogous to the manner in which Wilkins [11] uses constraints to construct partial descriptions of objects, as well as the specialization of abstraction in Tenenbergs [8] plan graphs.

When instantiating plan variables to actual objects or object characteristics, the constraints associated with that object are added to any existing constraints with the task. When abstracting a group of tasks into a higher-level task, constraints are combined into a single set representing the abstracted action. Plan variables which share the same symbol are made to be codesignating, and any preconditions which are not satisfied within the abstracted group remain as a precondition for the higher-level task; similarly, effects which reach outside the scope of the task group are established as an overall effect for the higher-level task.

As initially stated, a plan is a set of tasks to be performed in some order. Because a higher-level task fits this description, a plan is merely a higher-level task (throughout the remainder of this paper, "task" and "higher-level task" will be used interchangeably). Each plan's sub-tasks are ordered according to their connections, which represent sets of operators from temporal logic specifying their sequence within the plan.

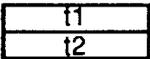
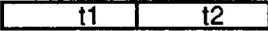
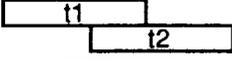
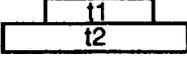
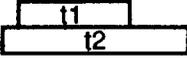
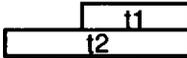
Relation	Operator	Inverse Operator	Pictorial Example
t1 <i>before</i> t2	<	>	
t1 <i>equal</i> t2	=	=	
t1 <i>meets</i> t2	m	mi	
t1 <i>overlaps</i> t2	o	oi	
t1 <i>during</i> t2	d	di	
t1 <i>starts</i> t2	s	si	
t1 <i>finishes</i> t2	f	fi	

Figure 4. Possible Temporal Relationships

2.3. Temporal Plan Network

The temporal operators that reflect time relationships between tasks are those discussed in [1], and are summarized in figure 4. The temporal relationship between two tasks is expressed as a disjoint set of temporal operators. Hence, (*task1* [*<, s, o*] *task2*) denotes that task1 either *is before*, *starts simultaneous to*, or *overlaps* task2. Using such sets as links between tasks, we construct a temporal network whose nodes are tasks and whose connections are temporal relationships between them. The network is constructed regardless of the ability of the agent who will ultimately perform

the tasks. We need not worry about parallelism during task/plan description, nor about whether the agent has one, two, or even ten arms. The only concern is what relationship each task has to other tasks. Undefined relationships are inferred based on defined relationships. Figure 5 shows an example; if we have the relationships $(task1 [s] task2)$ and $(task2 [o] task3)$ defined, we infer the relationship $(task1 [<,m,o] task3)$.

With the addition of temporal information, task abstraction becomes similar to the concept of reference intervals in [1]. Each higher-level task, then, is a temporal network of tasks from lower levels. For any such task, the temporal relations among its subtasks are validated by maintaining their transitive closure, which prevents ill-definitions such as $(task1 < task2)$, $(task2 < task3)$ and $(task3 < task1)$. This also may reduce some of the explicit ambiguity expressed in the task's definition. Any remaining ambiguity is resolved during plan translation (as seen in the next section). By means of task abstraction, expressed ambiguity, and plan variables (which can either be instantiated to actual objects, or be constrained by object characteristics), non-trivial plans can be constructed that are completely independent of the agent that will perform them.

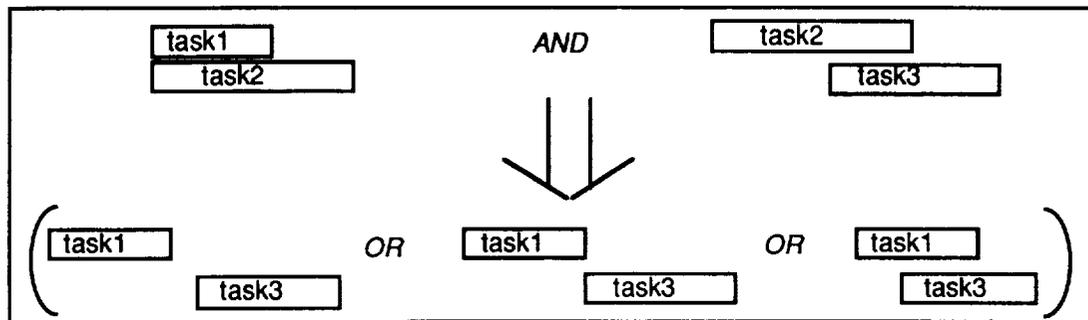


Figure 5. Implication in the Temporal Logic

3. Plan Translation

3.1. Validation Under Agent Constraints

In order to execute some plan, a declaration must be made as to what agent should perform the plan. Note, however, that this declaration may specify an agent that is incapable of such action. Therefore, the combination of agent and plan must be *validated*. This is essentially the job of plan translation. The luxuries that afford representation of the plan free from knowledge about an agent must now be considered in light of the agent capabilities. That is, any ambiguous part of the specified plan (plan variables, task abstraction, temporal ambiguity) must be resolved to a point where specific instructions understandable by the agent can be generated. While the agent-independent plan is represented in a hierarchy of temporal networks containing all possible orderings of plan performance, plan validation attempts to eliminate those temporal possibilities which are infeasible for a given agent, effectively producing an agent-dependent network. The validation process ensures that at least one possible traversal through this agent-dependent network is an acceptable plan, according to the agent's constraints, for accomplishing the desired goal. It is at this point that primitive tasks are mapped to specific agent instructions for execution. If at any point the agent is found incapable of performing the plan, it is said to be "rejected" from translation.

The first step of translation is to assign to the plan variables specific objects which the plan will manipulate, making use of object characteristics (see section 2) to aid in constraining the assignment. Once all the actual objects are determined, they can be used to check the physical

constraints of the specified agent. This is done by "matching" the physical properties of the agent with those of the objects. For instance, one constraint checks the mass of an object against the possible mass movable by the agent, while another constraint checks the size of each object to be sure the agent possesses the proper tools and end effectors to manipulate the object. Such constraints are used in validating that the agent is physically capable of manipulating the objects.

Once the agent's physical capability to perform the plan is established, the next matter of validation is to ensure the agent is functionally capable of plan execution. Functional constraints of an agent specify the temporal capability of an agent. Once a particular agent is designated for translation, its functional constraints (in the form of planning preconditions and effects) are integrated with the existing task and object constraints in the agent-independent plan. An agent is found functionally incapable when the plan requires simultaneous action which the agent cannot perform. A second reason for rejection is simply an ordering of the tasks that is incompatible with the agent's ability at a given moment. Examples of this are directing the agent to "grasp the pump" when no "hand" is free (i.e., is empty and available), or telling the agent to "release the filter" when no filter is being held. The temporal check made by functional constraints must examine transitions from one primitive task to another, and the timely ordering of the tasks can certainly be confirmed by stepping through the plan while "simulating" the action in the world.

3.2. Deriving an Agent-Dependent Network

An agent-dependent network is produced essentially by eliminating any infeasible temporal possibilities in the plan ordering. Presented here are two methods for obtaining this network. The first results in a network which contains every possible ordering in which the agent can successfully perform the primitive tasks to satisfy the goal, and excludes any problematic orderings. While this method has exponential computational complexity, it performs satisfactorily for small networks. Plans of significant abstraction, however, demand a more efficient translation process. The second method uses a simple heuristic that reduces the algorithmic complexity; and although the resulting network may produce an incomplete list of viable orderings, a straightforward assumption reducing the impact of this deficiency makes this method a preferred alternative to the first.

The essence of the first method to derive an agent-dependent network lies in computing the transitive closure over the temporal operators in the agent-independent network. Recall that such a transitive closure is always maintained for the subtasks of any high-level task while it is being defined. With this in mind, computing the transitive closure of *all* primitive tasks is a matter of relating tasks across hierarchical boundaries. This can be done using the "during" relationships (starts, during, finishes, and corresponding inverses) to relate sibling subtasks, as is done in the reference hierarchies of [1]. Such transitivity applies through all levels of the plan hierarchy, and can thus relate any primitive task to any other primitive task within a plan. A backtracking algorithm as is mentioned in [1] can be used to minimize ambiguity among the temporal operators in the final "closed" network. Once established, an agent-dependent network can be derived by a breadth-first-like traversal of the independent-network, updating virtual copies of the objects to maintain the status of the world, and eliminating operators that cause the traversal to "back up" due to an untimely ordering of events in consideration of the agent.

Intuitively, the above procedure operates in exponential time; a transitive closure algorithm for temporal intervals is discussed in [10]. The observation can be made, however, that a vast majority of these inferred links are irrelevant to the overall order of the plan. This raises the question: which links are necessary in validation and which are not? The temporal links of importance are those which impose simultaneous action on the agent. This means that validation

should be concerned not so much with the interconnections between *all* of the tasks, but rather with those which reflect the transition from one task to another. This focus on transitional tasks at the primitive level is the heart of the second method for obtaining an agent-dependent network.

This "endpoint method" makes use of a couple of existing properties of the agent-independent network to save time. First, since temporal links among the individual task's subtasks are consistent due to the maintenance of the transitive closure throughout task description, the entire network is temporally consistent before validation/translation begins (i.e., consistency among parents and consistency among children => consistency among siblings). Therefore, the transitive closure is unnecessary for this purpose. Second, by the above assumption, transitory activity is of main importance, so this method only considers relationships between primitive tasks which possibly occur at endpoints of higher-level tasks. Starting with the level-1 tasks (one level above the primitive/leaf level), a *begin-set* and *end-set* is computed for each set of subtasks. These represent the tasks that can possibly start or finish their parent task, respectively. Given higher-level tasks T1 and T2, and their respective begin-sets and end-sets, the "during" relationships are used to infer relationships among members within these sets. However, unlike the previous method which established relationships among all the primitive tasks, this method only uses the endpoint relationships to modify higher-level relationships pre-existing in the agent-independent plan. That is, the temporal relationship between two tasks T1 and T2 is pruned to eliminate any conflicts in transitioning from a primitive subtask of T1 to a primitive subtask of T2. Once the endpoint relationships are considered at this level, begin and end sets are computed for the next higher level in similar fashion, considering further relationships at the primitive level, and pruning those relationships from the network that are invalid for the agent. Pruning occurs either at the "local" network (the endpoint level), eliminating the temporal operator(s) which caused a conflicting task to be considered as an endpoint, or at the most abstract level that is appropriate, eliminating the temporal operator(s) that caused the endpoint sets to be in conflict. Finally, this method differs from the previous one, which used breadth-first traversal to find *all* viable paths, by using depth-first traversal to find *a* viable path. The viable path found becomes the sequence of the agent-dependent plan.

4. Plan Generation

Although the primary intent of this paper is to present "agent-independency," a brief description of the actual planner is provided to show how it uses the various planning and temporal constraints. The planner is nonlinear; that is, it produces a plan by deriving and further constraining sets of partial orders. It incorporates a constraint posting theme, using the objects' and agents' functional constraints as planning constraints. Certainly more thorough discussions of planning constraints and techniques exist elsewhere [3,11], but the terms are briefly defined here for clarity.

Each task contains an associated set of planning constraints (preconditions, subgoals, main effects, and side effects), which result from combining any associated agent or object constraints with other abstracted constraints. *Subgoals* are those conditions which must be satisfied before the execution of a task. *Preconditions* are those conditions which must be satisfied before the inclusion of that task into the plan. In essence, they are subgoals that are immediately satisfiable from tasks already existing in the plan. *Main effects* are conditions resulting from a task, and serve as a reason for selecting a task to achieve a particular goal. *Side effects* are conditions which are caused by a task, but which are not significant enough to warrant its inclusion in the plan. When adding tasks to a plan, a *clobberer* is a task which potentially defeats a precondition for another task, thus causing a break in plan causality. *Promotion* is the technique of constraining the clobberer to occur after the time when the precondition must be met. *Separation* is the technique of constraining the

clobberer not to codesignate with the precondition. That is, any plan variables that could be instantiated such that the clobberer's effects would defeat the precondition are prevented from doing so. Using these terms, figure 6 presents an outline of the planning approach.

Using the simple concepts of promotion and separation as defined above will force the planner to build plans that are sequential at their most abstract level. Incorporating temporal properties into the planner allows the construction of plans that are not committed to any particular order. Work along the lines of [2,9] associates temporal intervals with each task and condition/effect. Goals are achieved by "collapsing" intervals (asserting them be "equal") of conditions and effects. Hence, when a task is inserted into the plan its main effect is asserted to be temporally equal with the subgoal it is supposed to satisfy. This new relationship is propagated throughout the (top level) tasks in the plan. Upon detecting a clobberer, promotion and separation can still be employed, but rather than imposing a sequential constraint the conflicting tasks are simply constrained not to share the same interval (i.e., not be overlapping, during, starting, etc.). Such a combination between traditional nonlinear planning and temporal planning allows the generation of task plans that make no presupposition concerning which activities can be parallel and which must be sequential.

- General procedure to satisfy a goal:
1. If a main or side effect which satisfies the goal already exists in the plan, then constrain the effect's task to precede the goal. (and precede to next goal)
 2. Otherwise, select a task whose main effect matches the goal and whose preconditions can be immediately satisfied. (fail if no such task exists)
 3. Insert task *T* into the plan, binding any plan variables necessary, by constraining *T* to precede the goal and constraining tasks satisfying *T*'s preconditions to precede *T*.
 4. If any clobberers exist, try promoting them past the clobberer. (otherwise step 7)
 5. If promotion fails, try separation.
 6. If separation fails, then backtrack to step 2 and select a different task to satisfy the goal.
 7. Upon successful addition of *T* into plan, place *T*'s subgoals onto goal queue and continue until goal queue is empty.

Figure 6. Outline of Planning Approach

5. Summary and Future Directions

A system has been introduced for describing and generating plans in a representation independent of an agent, which can subsequently be translated into agent-dependent instructions suitable for execution. Agents and objects are represented in an object-oriented fashion, allowing their description of physical and functional capabilities to assist in constraining plan description/generation, and to be "matched" for plan validation purposes in translation. Tasks can either be primitive actions based on movement, or abstracted to higher-level tasks (synonymous with plans) consisting of subtasks arranged in "temporal networks" (which allow temporal ambiguity in describing task orderings). Tasks and objects are combined to form an agent-independent plan. Plan translation transforms this into an agent-dependent plan by validating the combination of the plan with the properties and constraints of a specific agent, resulting in a set of instructions executable by the agent. Plans are constructed automatically using nonlinear planning techniques which operate on functional constraints of agents and objects. Integrating temporal

planning with these techniques provides a more flexible planner which holds no bias in sequencing activity.

This system is implemented for the domain of maintenance and repair of Space Station Freedom. It currently generates VAL II instructions to a PUMA-560 with integrated force/torque and vision sensors, English instructions for a crewmember, and software procedure calls to a robotic simulator. Several extensions to this system are planned. Planning in complex domains often requires that plans be initially generated from incomplete data, or data that will evolve over time. Current methods allow the use of temporal relationships in a deductive fashion, reducing the possibilities of task ordering as more information is known about the plan. However, retracting assertions which have reduced the temporal network is computationally very expensive. Therefore, future work will concentrate on adding nonmonotonicity to the temporal logic to facilitate reasoning with changing data. This will provide a foundation to examine replanning strategies for the temporal planner. For better integration with crewmembers, techniques for explaining planner rationale will be explored. In addition, crew skill models will be developed to allow better presentation of plans and their explanations to crewmembers. These enhancements to agent modeling will also be extended to include a more sophisticated robotic agent with a dexterous three-fingered hand and advanced sensing capabilities. All of these activities will support the development of technology which allows manned spacecraft workload to be modeled and shared between crewmembers and robots, and activity plans to be automatically generated regardless of the agents who will accomplish them.

References

- [1] Allen, J.F., *Maintaining knowledge about temporal intervals*, Communications of the ACM, vol. 26, 1983, pp. 832-843.
- [2] Allen, J.F. and Koomen, J.A., *Planning using a temporal world model*, Proceedings IJCAI-83, 1983, pp. 741-747.
- [3] Chapman, D., *Planning for Conjunctive Goals*, Artificial Intelligence, vol. 32, 1987, pp. 333-377.
- [4] Hogge, J.C., *Compiling plan operators from domains expressed in Qualitative Process Theory*, Proceedings AAAI-87, 1987, pp. 229-233.
- [5] Ladkin, P., *Time representation: A Taxonomy of Interval Relations*, Proceedings AAAI-86, 1986, pp. 360-366.
- [6] Leban, B., McDonald, D., and Forster, D., *A Representation for Collections of Temporal Intervals*, Proceedings AAAI-86, 1986, pp. 367-371.
- [7] Pelavin, R.N. and Allen J.F., *A model for concurrent actions having temporal extent*, Proceedings AAAI-87, 1987, pp. 246-250.
- [8] Tenenbergs, J., *Planning with abstraction*, Proceedings AAAI-86, 1986, pp. 76-80.
- [9] Tsang, E.P.K., *TLP - A Temporal Planner*, Advances in Artificial Intelligence, eds. Hallam and Mellish, 1987, pp. 63-78.
- [10] Vilian, M. and Kautz, H., *Constraint Propagation Algorithms for Temporal Reasoning*, Proceedings AAAI-86, 1986, pp. 377-382.
- [11] Wilkins, D., Practical Planning, Morgan Kaufmann Publishers, 1988.

SPIKE: ARTIFICIAL INTELLIGENCE SCHEDULING FOR HUBBLE SPACE TELESCOPE

Mark Johnston*, Glenn Miller†, Jeff Sponsler*,
Shon Vick*, and Robert Jackson†

*Space Telescope Science Institute
3700 San Martin Drive
Baltimore MD 21218*

Abstract

The problem of optimal spacecraft scheduling is both important and difficult. Efficient utilization of spacecraft resources is essential, but the accompanying scheduling problems are often computationally intractable and are difficult to approximate because of the presence of numerous interacting constraints. We have applied artificial intelligence techniques to the scheduling of the NASA/ESA Hubble Space Telescope (HST). This presents a particularly challenging problem since a yearlong observing program can contain some tens of thousands of exposures which are subject to a large number of scientific, operational, spacecraft, and environmental constraints. We have developed new techniques for machine reasoning about scheduling constraints and goals, especially in cases where uncertainty is an important scheduling consideration and where resolving conflicts among conflicting preferences is essential. These techniques have been utilized in a set of workstation-based scheduling tools (Spike) for HST. Graphical displays of activities, constraints, and schedules are an important feature of the system. High-level scheduling strategies using both rule-based and neural network approaches have been developed. While the specific constraints we have implemented are those most relevant to HST, the framework we have developed is far more general and could easily handle other kinds of scheduling problems. This paper describes the concept and implementation of the Spike system and some experiments in adapting Spike to other spacecraft scheduling domains.

INTRODUCTION

To obtain the maximum benefit from expensive space facilities it is important to schedule spacecraft operations in an optimal manner. Since truly optimal scheduling is usually computationally intractable, it is therefore necessary to determine the best possible schedule given the resource and time constraints on the computational effort that can be invested.

The fundamental requirements of optimal spacecraft scheduling are similar in many ways to those of other scheduling problems, e.g. those encountered in commercial and industrial domains. These problems have been found to be notoriously difficult to solve in practical settings. In this paper we describe the source of some of these difficulties and how the use of advanced software technology ("artificial intelligence") can be applied to help overcome them. We describe the progress made at Space Telescope Science Institute (STScI) in developing AI tools for

* Space Telescope Science Institute (Operated by the Association of Universities for Research in Astronomy for the National Aeronautics and Space Administration)

† Astronomy Program, Computer Sciences Corporation, Staff member of the Space Telescope Science Institute

the Hubble Space Telescope (HST), and conclude with a discussion of how these tools can be adapted for other spacecraft scheduling problems.

HUBBLE SPACE TELESCOPE SCHEDULING

In this section we give a brief description of the problem of scheduling Hubble Space Telescope. Further discussion can be found in [Miller et al. 1987, Johnston 1988, and Johnston et al. 1987].

Astronomers from around the world submit proposals to obtain observations with HST. Following an annual peer review and selection process¹, accepted proposals are prepared and provided by the successful proposers to STScI. These contain the detailed specification of each observing program. In addition to such obvious aspects as which instrument to use and which astronomical targets to observe, astronomers can (and do) specify other constraints on how their observations are to be taken. These include a wide variety of relative timing constraints (precedence, minimum and maximum time separation, interruptability conditions, repetitions). Some exposures must be taken at precise times or within specified time windows. Others may require special observing conditions (e.g. they must be taken while HST is in the Earth's shadow to minimize scattered light background). Still others may be conditional on results obtained after analysis of precursor exposures taken by HST, or, in some cases, by other observatories (on the ground or in space).

Constraints specified by the astronomer must be combined with constraints due to many other sources. HST is limited in how close to bright sources of light (e.g. the Sun, Moon, and sunlit earth limb) it can be pointed. Sensitive observations require minimizing subtle sources of stray and scattered background light. The low-earth orbit of HST (~95m period) means that targets are typically occulted by the earth after no more than ~40m of observing time. Radiation belts interfere with observations to reduce even further the available viewing time per orbit. For pointing stability, a pair of suitable guide stars must be found for placement in Fine Guidance Sensors: in some parts of the sky these stars are sparse. There are thermal and power constraints that appear in the form of off-nominal roll limitations and recovery-time requirements. This list of constraints is far from exhaustive, and, in general, each exposure is subject to some tens of constraints.

A one-year observing schedule for HST will generally contain ten to thirty thousand exposures constrained as described above. It is clear that a scheduling problem of this magnitude is a very large one indeed.

SPIKE

Computer techniques for optimal scheduling have been investigated for many years by a number of researchers (see, e.g. [King and Spachis 1980] for a comprehensive review and bibliography). Much of this classical work has focussed on versions of the idealized "job-shop" scheduling problem. This problem and related ones are NP-complete, meaning that there are no efficient algorithms for finding solutions (see, e.g., [Garey and Johnson 1979]).

The basic problem with these classical results is that they require key features of the problem to be abstracted away, so that even "exact" solutions to the abstracted problem are often of little relevance to the original "real" problem. Approximate solutions to the abstracted problem suffer from the same limitations. It is clear that classical approaches can be useful for problems which

¹The first such solicitation and proposal selection was completed in mid-1989.

are sufficiently simple: in practice this often means that schedule optimization is driven by a *single* overriding criterion. For the problem of scheduling complex modern space facilities, however, this is not the case: more powerful techniques are needed that can handle the complexities of real-world problems.

There are four notable features of spacecraft scheduling that make it a difficult problem: *interacting constraints*, *uncertainty*, *optimization criteria*, and *search*. Realistic scheduling problems typically involve a large number of different types of constraints, both strict and preference. These constraints often have a very large range of timescales compared to classical scheduling domains, ranging from seconds or minutes to months or years. Trading off and balancing constraints adds greatly to the complexity of scheduling. Uncertainty can enter in a variety of ways, ranging from chaotic (i.e. completely unpredictable) scheduling factors to the smooth degradation of confidence in the results of an extrapolated model. Optimization criteria are often complex and situation-dependent: there is usually no single criterion that can be used to indicate schedule optimality. At various times, the schedule may be optimized with respect to operational efficiency, schedule robustness, or speed of recovery to an original schedule following a disruption. The process of constructing schedules by searching among alternatives is computationally expensive, with exhaustive search usually out of the question.

Spike is an activity-oriented scheduling system developed at Space Telescope Science Institute for scheduling HST. The Spike project was initiated in early 1987: the system has so far been used during ground test activities and is currently beginning work on the first flight schedule (HST launch is now scheduled for April 1990). Spike's current focus is the long-range scheduling problem, i.e. that of scheduling over a year or more to a resolution of a few days. The system is not limited to this problem and was designed to schedule at arbitrary time resolution. The basic architecture of the system is illustrated schematically in Fig. 1.

The overall approach adopted in Spike was inspired by advances in artificial intelligence research (see., e.g. [Fox and Smith 1984] and [Smith, Fox, and Ow 1986] for a discussion of these techniques as applied to factory scheduling problems). Spike incorporates several novel features, however, including an innovative constraint representation and reasoning mechanism [Johnston 1989b] and a new type of search technique motivated by research on artificial neural networks [Adorf and Johnston 1989, Johnston and Adorf 1989].

Spike's constraint representation provides not only for *strict* constraints but can also represent in a natural way *preference* constraints that indicate conditions that are desired but not required in the final schedule. Constraints are propagated in a manner that informs the scheduler (whether human or an automatic search process) what the allowed scheduling opportunities are for all tasks, as well as a measure of the degree of preference of those opportunities.

In addition to a powerful constraint representation mechanism, Spike employs several other strategies to reduce the size of the problem as much as possible:

- (1) The overall scheduling period is divided into intervals and activities to be scheduled are first committed to these intervals. Once a satisfactory set of commitments is found, the intervals are further decomposed and the process repeated. This avoids the need to make early commitments to specific times for activities when scheduling over long periods (months to years).
- (2) Implications of constraints are pre-propagated to the greatest extent possible and saved, avoiding repetitive computations during the scheduling search process. This also identifies many types of overconstrained activities that are unschedulable because of irreconcilable constraint conflicts.

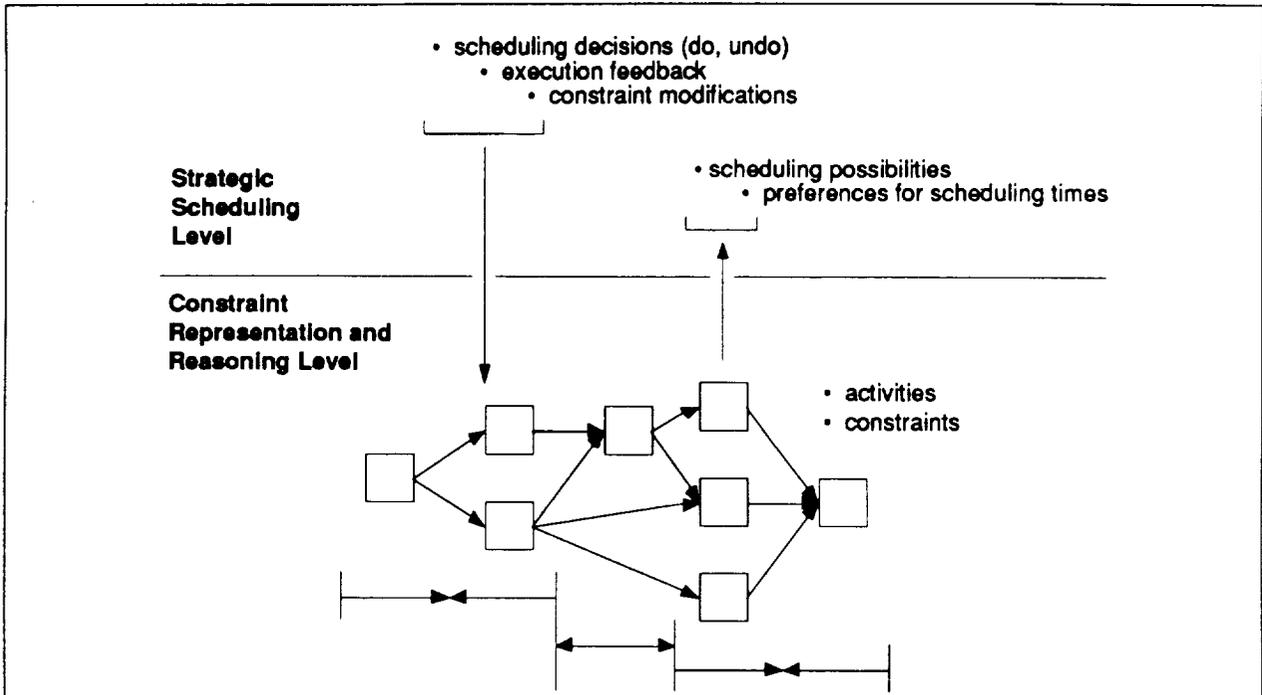


Figure 1: A diagram of the overall Spike architecture. The lower level is a constraint representation and reasoning system which contains descriptions of activities to schedule and their constraints. Temporal constraints are indicated schematically, but the system can deal with a wide variety of strict and preferences constraint types. The upper level interacts with the constraint representation level when searching for feasible and optimal schedules. A variety of modular search strategies can be utilized at this level.

- (3) Activities that can be clumped together and scheduled as single “meta-activities” are identified by the system before scheduling starts. This reduces the number of individual activities to schedule, as well as reducing the number of constraints.
- (4) The set of activities to schedule can be partitioned into disjoint sets with associated precedence. The resources consumed by scheduling one set can be cascaded to other sets, thus permitting a significant reduction in the number of activities that must be considered at one time.

Uncertainty is always a serious problem with predictive scheduling. Spike’s constraint mechanism provides several ways to deal with this problem. The most important is to define constraints that represent the *probability* of success as preference constraints, or, alternatively, constraints that represent the *maximum acceptable risk* as strict constraints. This permits some constraints to be treated in a statistical fashion. This technique is used to deal with the fact that the in-track position of HST in its orbit cannot be accurately predicted more than a few months into the future.

Although Spike was developed as an automatic scheduling system, it is fundamentally a support tool for the people who are responsible for making scheduling decisions. Thus one of the most important characteristics of the scheduler is how it interacts with users. The user must have *visibility* into all aspects of the scheduling problem and the evolving schedule. The user must also have *control*, i.e. the ability to override any decisions made by the automatic system, and the ability to create and evaluate alternative schedules. These features are provided by the system. The user interface makes extensive use of a bitmapped graphics window system and mouse to

facilitate two-way interaction with the user. An example screen from the running system is shown in Fig. 2.

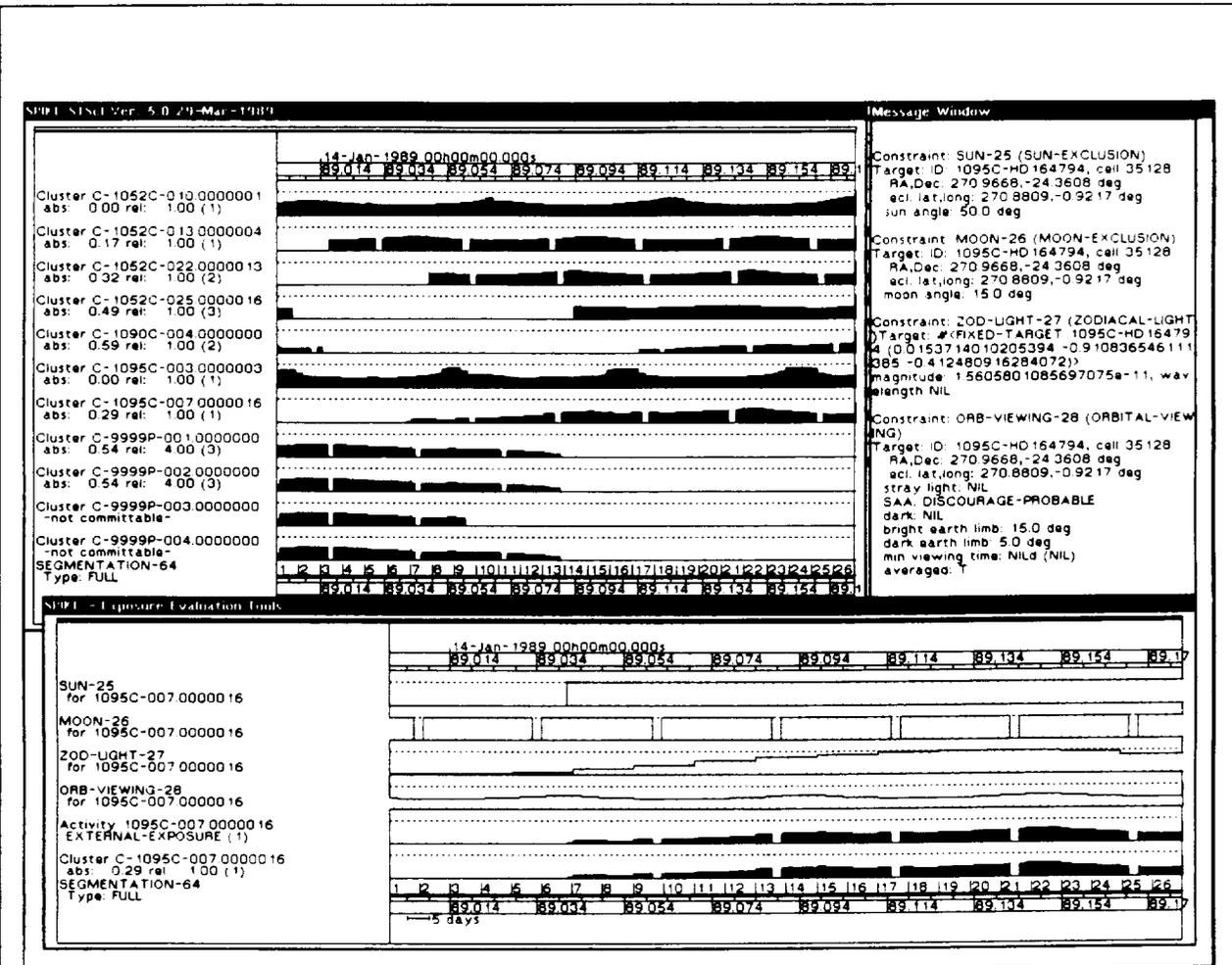
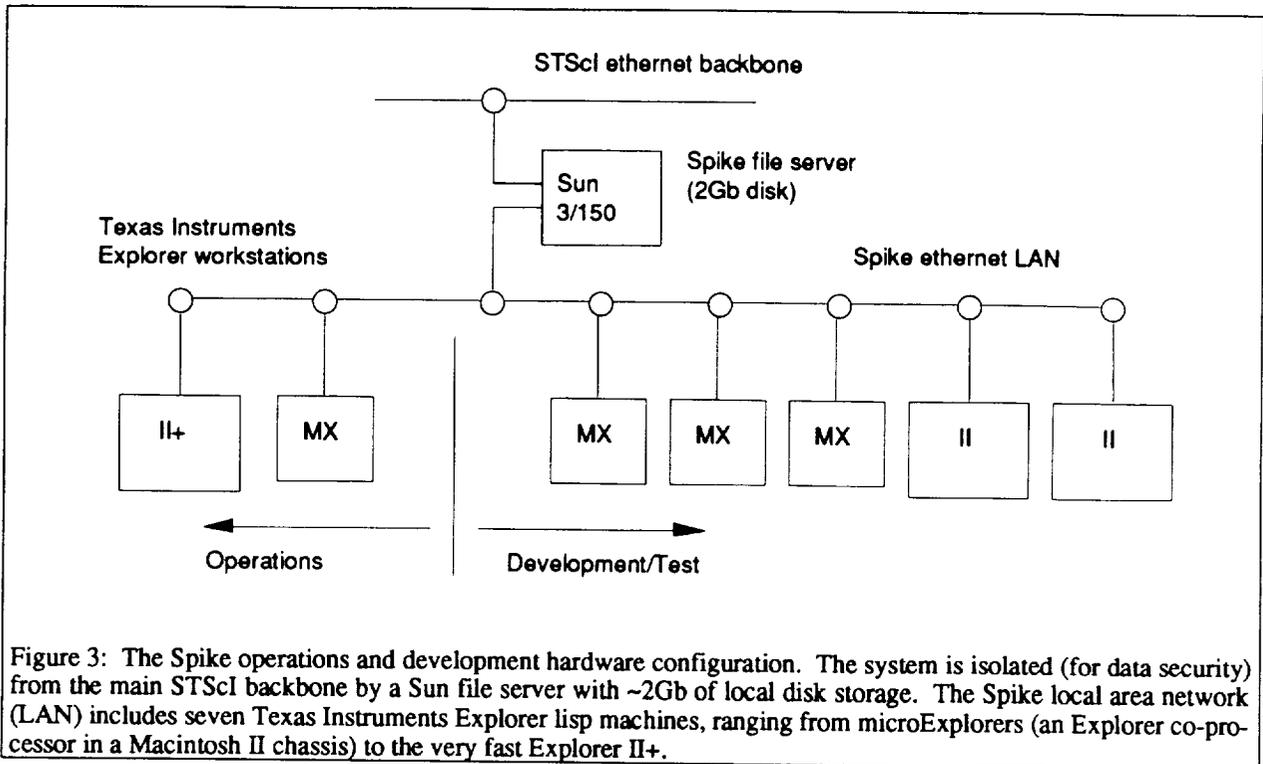


Figure 2: Example screen from Spike showing the scheduling of some HST observations. The upper left window represents a six-month scheduling interval and displays the combined degree of preference for scheduling a number of exposures (running vertically up the window). The lower window shows an expanded time view of one specific exposure and the constraints that contribute to its scheduling preference. The text window on the right displays descriptive information about the schedule, activities, and constraints. The user interacts with the system by clicking on various active regions and selecting from pop-up menus. For example, clicking on the time scale at the bottom of each window permits zooming in or out in time, or paging forwards or backwards. The user can create new windows and build new displays dynamically.

Spike was developed on Texas Instruments Explorer workstations and is implemented in Common Lisp, (old) Flavors, and the Common Lisp Object System (CLOS). Commonwindows (from Intellicorp, Inc.) is used as the windowing system for the user interface. The operations hardware configuration is shown in Fig. 3. There are two operations workstations for scheduling and for evaluating programs for scheduling feasibility. These are networked with the development cluster of five workstations. Both share the same file server which is a Sun workstation with 2Gb of local storage. This server also acts as a gateway to the STScI ethernet backbone and provides secure access to the scheduling data.



Spike has recently been ported to Allegro Common Lisp (from Franz, Inc.) running on Sun workstations. The user interface in this configuration is displayed in an X-windows environment. This permits running the computational kernel of Spike on one machine while the user interface runs on another. Although the present computational power of general-purpose workstations does not yet match that of the fastest Lisp machines, it can be expected that the demonstrated portability to Unix workstations will prove useful for HST and other applications.

Spike is currently operational at STScI and is in the first stages of constructing the initial schedule to follow launch and vehical and instrument checkout.

EXTENSIONS

Scheduling problems are rarely static: changes come about because of increased experience with the problem and because of on-orbit experience with the operating spacecraft. Spike was designed with this fact in mind, particularly with regard to changing and adding constraints.

This flexibility has been exercised by conducting several experiments in adapting Spike to schedule observations from other missions. The results of one of these experiments are shown in Fig. 4. This represents the scheduling of a one-year period of European International Ultraviolet Explorer (IUE) programs to a resolution of one week. Spike was adapted to represent the IUE scheduling constraints with minimal effort. Similar experiments have been conducted for the Extreme Ultraviolet Explorer (EUVE) as well as for a ground-based telescope in Chile [Johnston 1988a]. Spike has been chosen by the EUVE project to perform science scheduling for that mission; the Unix version of the system is presently running at the University of California, Berkeley, for that purpose.

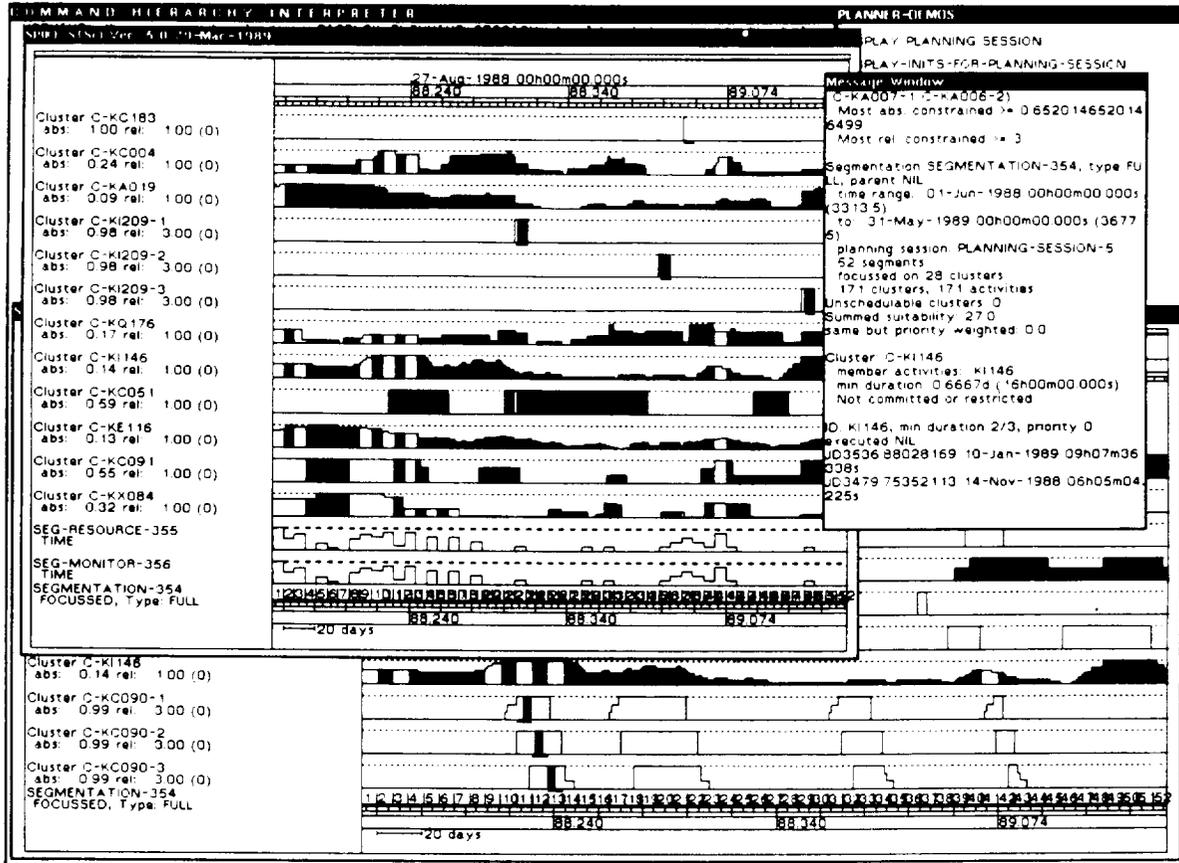


Figure 4: Display of the results of the experimental adaptation of Spike to schedule IUE programs. Shown here is a snapshot of a partial schedule. The top left window covers a time period of a year: 4-hour observing periods are being allocated to weeks. The function plotted is the scheduling degree of preference as in Fig. 2, but here the restrictions of some programs to specific times or sets of times is indicated by the outlined portions of the curves. The bottom plot in the same window shows the amount of time allocated per week (solid line) compared to that available (dashed line).

SUMMARY AND CONCLUSIONS

It is clear that the software technology and approaches to scheduling embodied in Spike have reached a sufficient level of development that intelligent spacecraft scheduling is a realistic goal. The use of AI techniques makes it possible to develop and adapt software such as Spike for a variety of spacecraft scheduling problems. Spike embodies innovative approaches in several areas, including constraint representation and reasoning and optimal scheduling search methods for large-scale problems.

Future plans for Spike include upgrading all of the system to be compatible with the new ANSI standard Common Lisp Object System (CLOS) and further research on scheduling search methods for large problems. Experience from early scheduling for HST operations (in progress now) will be valuable in further augmenting both the user interface and the computational core of the system.

REFERENCES

- Adorf, H.-M., and Johnston, M. 1989: "Stochastic Neural Networks for Constraint Satisfaction Problems", submitted.
- Fox, M, and Smith, S. 1984: "ISIS: A Knowledge-Based System for Factory Scheduling," *Expert Systems* 1, p. 25.
- Garey, M., and Johnson, D. 1979: *Computers and Intractability*, (W.H. Freeman & Co.).
- Johnston et al. 1987: "HST Planning Constraints", 1987, Space Telescope Science Institute, Spike Tech. Report 87-1.
- Johnston, M. 1988a: "Automated Observation Scheduling for the VLT", in *Proc. ESO Conference on Very Large Telescopes and their Instrumentation*, Garching, March 1988.
- Johnston, M. 1988b: "Automated Telescope Scheduling," in *Proc. Conf. on Coordination of Observational Projects*, Strasbourg, Nov. 1987.
- Johnston, M. 1989a: "Knowledge-Based Telescope Scheduling," in *Knowledge-Based Systems in Astronomy*, ed. A. Heck and F. Murtagh (Springer-Verlag: 1989)
- Johnston, M. 1989b: "Reasoning with Scheduling Constraints and Preferences," Spike Tech. Report 89-2 (Jan. 1989).
- Johnston, M., and Adorf, H.-M. 1989: "Learning in Stochastic Neural Nets for Constraint Satisfaction Problems," in *Proc. NASA Conference on Telerobotics*, Pasadena, CA (Jan. 1989).
- Johnston, M., and Miller, G. 1988: "AI Approaches to Astronomical Observation Scheduling," in *Proc 3rd International Workshop on Data Analysis in Astronomy*, Erice (June 1988) (Plenum Press, NY).
- King, J.R., and Spachis, A.S. 1980: "Scheduling: Bibliography and Review," *Int. Journal of Physical Distribution and Materials Management* 10, p. 105.
- Miller, G., Johnston, M., Vick, S., Sponsler, J., and Lindenmayer, K. 1988: "Knowledge Based Tools for Hubble Space Telescope Planning and Scheduling: Constraints and Strategies", in *Proc. 1988 Goddard Conference on Space Applications of Artificial Intelligence*.
- Miller, G., Rosenthal, D., Cohen, W., and Johnston, M. 1987: "Expert System Tools for Hubble Space Telescope Observation Scheduling," in *Proc. 1987 Goddard Conference on Space Applications of Artificial Intelligence*; reprinted in *Telematics and Infomatics* 4, p. 301 (1987).
- Smith, S., Fox, M., and Ow, P. 1986: "Constructing and Maintaining Detailed Construction Plans," *AI Magazine*, Fall 1986, p. 45

Resource Allocation Using Constraint Propagation

John S. Rogers
Artificial Intelligence Lab
Johnson Research Center
The University of Alabama in Huntsville

Introduction

Scheduling and resource allocation considerations are fast becoming one of the major areas of research today. Although these are not new problems, technological advances continue to open new and ever increasing areas where having the right resources at the right place at the right time is crucial. As we enter into the space age and as the costs involved in doing specific research and development tasks increase the fundamental equation is truly, Time is Money.

The methods and techniques applied in this area are often as much an art form as a truly mathematical form. This is not to say that the foundations of each of these methods are not mathematically sound, but rather the how, when and why used in applying the mathematical principals can produce rather unique and sometimes more acceptable results. More specifically, there are certain parameters that are common to most resource allocation problems. How a particular application manipulates and utilizes these parameters often determines the degree of performance of a resource allocation system. A technique that can be applied to one of these parameters is presented here.

In many resource allocation and scheduling problems one of the major difficulties facing the investigator is determining how to apply different constraint equations. Numerous research efforts have been done in this area in an attempt to increase the investigator's control and ability in applying constraints. There are many widely varying techniques that are available and frequently used to solve this problem. Usually specific characteristics of the particular domain involved and the types of constraint equations that must be applied are the deciding factors in the selection of the appropriate technique. This is usually adequate for the current application; however, this solution usually does result in a generalized mechanism that can be applied to differing domains with only minor modifications.

In most cases a traditional approach of evaluating the constraining parameters for each one of the activities that is being scheduled at the time it is scheduled is sufficient. This is reasonable when the constraining relations for the activities are functions which depend upon resources solely. However, when an interdependency exists between the activities or if the domain

is not constant new problems may arise. A method that addresses this in resource allocation problems is the method of constraint propagation. In the following sections of this paper the concepts involved in applying this technique are presented.

Dynamic Allocation Scenarios

One area of interest today which can pose some unique and sometimes tedious constraint evaluation problems is that of dynamic domains. In this arena the domains are either constantly changing or possibly modifiable at discrete points of time within the analysis process. This scenario creates a true illustration of reality since all too often activities may require the consumption of a variable amount of a resource. Then as the activity is performed a more refined estimate of future available quantities of the resource is determined. This causes the domain to be dynamically modified. Due to the newly refined domain, a new allocation may need to be performed over the modified range to best utilize the remaining resources.

In approaching this task one might consider some type of readily available dynamic programming technique that would iteratively reformulate the resource allocation scheme based on the current domain state. Usually this is a plausible approach; however, in many instances this can cause severe time delays in the allocation softwares ability to traverse the domains time range. This only tends to magnify the often already excessive computational time required to perform the allocation process.

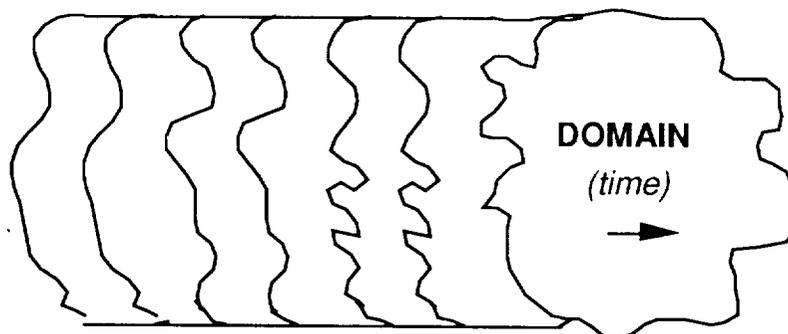


Figure 1 Dynamic Changing Over Time

What is needed is an alternative approach that can incorporate the effects domain changes can have on an existing allocation. In such an approach hopefully only those particular activities that are effected will be engaged and therefore reduce

any unnecessary constraint calculations. One such technique is *Constraint Propagation*.

What is Constraint Propagation?

Constraint Propagation is the technique of having localized mechanisms that control the behavior of individual activities as changes in the domain occur. This is more than just a constraining equation that is applied at a local level. It requires activation control to be maintained at a local level while domain changes happen globally. If only those activities that are directly affected by the domain change are activated then a direct reduction in the number of constraint equation evaluations can be achieved. Theoretically this evaluation reduction would result in the time required in the reallocation being reduced. Quite possibly each modification made by the adjustment of the allocated items across the time range could cause a subsequent alteration in the consumption of additional resources. More than likely this would initiate a series of downstream allocation modifications which could in effect initiate even more downstream allocation modifications and so on. The procession of changes effected on the downstream allocation of events can be considered analogous to a propagating wave across the domains remaining range. Thus arises the term *Constraint Propagation*.

Obviously the effectiveness of this methodology is closely tied to the amount of interdependence that exists between the different activities regarding the type and quantity of usage of the various resources. It may be logical to assume that as the commonalties regarding resource requirements between the different activities increases then the level of increased performance in this method decreases. However, logically one would assume that any reduction in the application of constraint equations should produce some amount of time savings. This implies that the overall performance increase seen by the system should always be non-negative.

How do you design a Constraint Propagation System?

Specific considerations must be made in designing a resource allocation software system that applies the technique of constraint propagation. First, there is the need for local control of processes. This tends to exhibit a requirement for code encapsulation at the activity level while maintaining global access to domain information. Although there are possibly a variety of approaches to achieve this, the best approach is through the implementation of Object Oriented Programming Techniques (OOP). With this approach the individual activities and resources can be constructed as objects. An activity object locally contains all the information describing not only resource consumption requirements and constraints, but also information which controls what activation triggers exist that would cause a

re-evaluation of this activity in the allocation process. All the information would exist as specific slot values located within the object itself. Likewise a resource object could contain the resources current state, usage history, any global limitations, and other pertinent resource specific information.

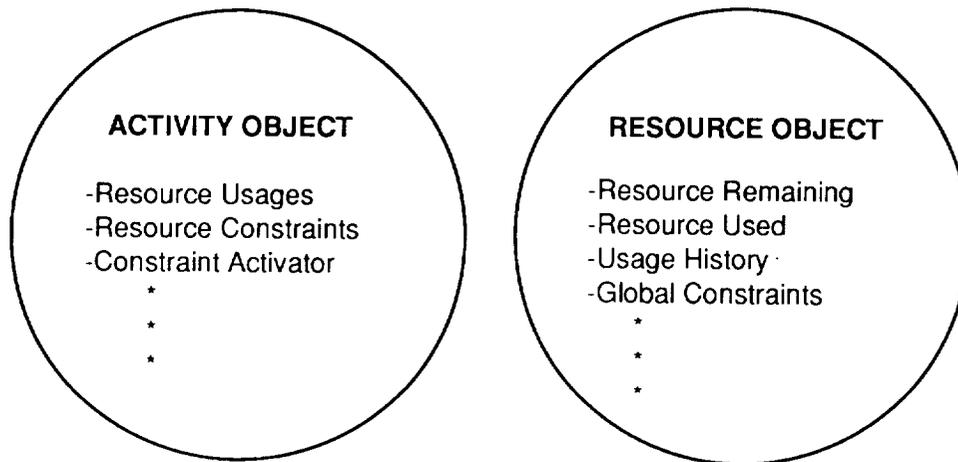


Figure 2 Object Data Structure

Many of the desired characteristics of a Constraint Propagation system are similar to those found in Frame Based Reasoning (FBR) systems. In an FBR system components can be designed that describe the control structure of the frame. These so called "daemons" are activated and controlled by the environment in which the frame resides. For example, an "if-decreased" daemon could possibly cause the initiation of the control code when a quantity of a specific resource is reduced for the time frame containing the specific performance of the activity. The range of functionality of these daemons is limited only by the developers ability to tailor the environments object manipulation characteristics.

There are several programming environments (C++, ADA, LISP, etc.) that boast an object data structure. For ease of manipulation and flexibility of control LISP is probably the most conducive of these environments in which to develop a Constraint Propagation system. This is due to the extensive functionality associated with the Lisp object as compared with other programming environments. These structures support the full range of message passing capabilities and can have external functional bodies called "Methods" which can be applied to each object. Simply put, a "Method" is a segment of computer code that acts similarly to a function that can be applied to a specific class of objects. It is through the use of these "Methods" that the behavior of each of

the daemon slots within the object is defined and controlled. By the use of this varied level of control constraint propagation techniques become feasible.

Summary

In this brief presentation the concept of constraint propagation has been discussed. Certainly, performance increases are possible with careful application of these constraint mechanisms. The degree of performance increase is related to the interdependence of the different activities resource usage. Although this method of applying constraints to activities and resources is often beneficial, it is obvious that this is no panacea cure for the computational woes that are experienced by dynamic resource allocation and scheduling problems. A combined effort for execution optimization in all areas of the system during development and the selection of the appropriate development environment is still the best method of producing an efficient system.

References

1. Howard, Geoffrey S., "Object Oriented Programming Explained," *Journal of Systems Management*, July 1986 pp. 13-18.
2. Cox, Bred J., Object Oriented Programming, Reading, Massachusettes: Addison Wesley Publishing Company, 1986.
3. Rich, Elaine, Artificial Intelligence, McGraw Hill, New York, 1983.
4. Ozden, Mufit, "A Dynamic Planning Technique for Continuous Activities Under Multiple Resource Constraints," *Management Science*, Vol. 33, No. 10, October, 1987 pp.1333 -1346.
5. Dreyfus, S. E. and A. M. Law, The Art and Theory of Dynamic Programming, Academic Press, New York, 1977.

DEVS-based Intelligent Control of Space Adapted Fluid Mixing

Sung-Do Chi and Bernard P. Zeigler
 AI-Simulation Group
 Department of Electrical and Computer Engineering
 University of Arizona, Tucson, AZ 85721

Abstract

This paper describes the development of event-based intelligent control system for a space-adapted mixing process by employing the DEVS (Discrete Event System Specification) formalism. In this control paradigm, the controller expects to receive confirming sensor responses to its control commands within definite time windows determined by its DEVS model of the system under control. We apply the DEVS-based intelligent control paradigm in a space-adapted mixing system capable of supporting the laboratory automation aboard a Space Station.

I. Introduction

"Intelligent control", the intersection of artificial intelligence, conventional automatic control, and operations research approaches, is receiving increasing attention in both theory and application(3). An intelligent control system often employs a hierarchical control structure in which a higher-level intelligent controller supervises a lower-level conventional controller. The event-based control paradigm, introduced by Zeigler(4), realizes such intelligent control by employing a discrete eventistic form of control logic represented by the DEVS formalism.

In this control paradigm, the controller expects to receive confirming sensor responses to its control commands within definite time windows determined by its DEVS model of the system under control. Since the DEVS formalism is at heart of event-based control system design, such controllers can be readily checked by computer simulation prior to implementation. Thus the DEVS formalism plays the same role to event-based control that differential and difference equation formalism play to conventional control(4). An advantage of an event-based control system using DEVS models includes its fault diagnostic capability supported by DEVS-Scheme, a LISP environment implementing of DEVS formalism(2,7,8).

This paper describes the development of DEVS-based intelligent control system for a space-adapted mixing process. The paper first reviews the concept of DEVS formalism, then uses it to formalize the dynamics of a mixing process. It shows the hierarchical architecture of the realized intelligent control system for mixing. Several simulation runs illustrate the technique.

II. DEVS Concept for Event-based Control

The Discrete Event System Specification (DEVS) formalism introduced by Zeigler(5) provides a means of specifying a mathematical object called a system. Basically, a system has a time base, inputs, states, and outputs, and functions for determining next states and outputs, given current states and inputs(6). In the DEVS formalism, one must specify 1) basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion. Detail descriptions of how a basic model, called an *atomic model* and the second form of model, called a *coupled model*, are specified are found in (1,7).

DEVS-Scheme is an implementation of the DEVS formalism in SCOOPS, an object-oriented superset of Scheme (a Lisp dialect), which enables the modeler to specify models

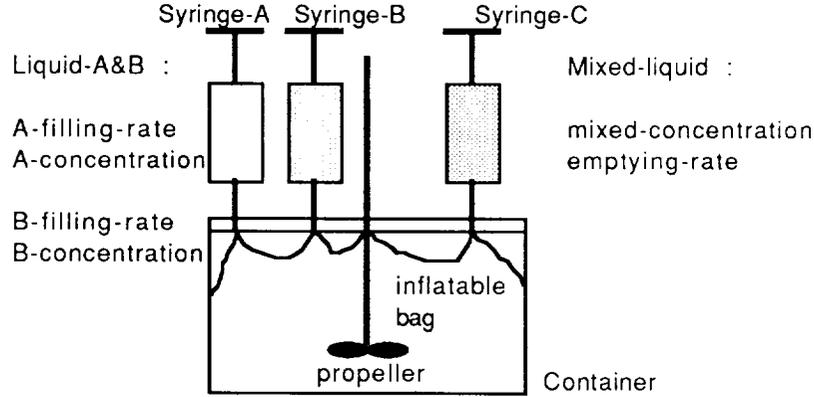


Figure 1: Space Adapted Mixing System

directly in DEVS terms. DEVS-Scheme supports building models in a hierarchical, modular manner (successively putting systems together to form larger ones), a systems-oriented approach not possible in conventional languages. Detail description of DEVS-Scheme including its class hierarchy is available in (1,7).

The DEVS formalism is more than just a means of constructing simulation models. It provides a formal representation of discrete event systems capable of mathematical manipulation, just as differential equations serve this role for continuous systems. We illustrate how mixing systems specified for the space environment may be advantageously mapped into DEVS representations. Suitably operating on the structure of such DEVS models provides a basis for the design of an event-based controller.

III. Space Adapted Mixing Process

The design of a space-adapted container would have an aluminium bottle containing an inflatable bag, which is the actual liquid container; liquid is injected/extracted by means of syringes; air pressure between the outside of the bag and the inside of the bottle wall ensures that the bag remains “full” at all times. We treat a system with space adapted container stirred by rotating propeller as shown in Figure 1. The mixing process is batch - some quantity of fluid with a given concentration is added to a container already filled with a liquid of another concentration. The container is fed with an incoming liquid-A from the syringe-A with a flow rate r_a by the control command, FILL-A. Once the level of liquid-A reaches to its pre-specified level, the flow of liquid-A might be stopped by control command, STOP. Then, another liquid-B from the syringe-B is added into the liquid-A in the container with a flow rate r_b until it receives the control command, STOP. Both feeds contain dissolved material with constant concentrations, C_a and C_b , respectively. Assume that the propeller starts either at the same time as liquid-B starts to be filled or after filling liquid-B is complete. The propeller should cease its operation by the control command, STOP. When the propeller stops, the concentration of both liquids should reach to the equilibrium value. The outgoing flow to the syringe-C has a flow rate r_c .

Figure 2 illustrates the relationship between dynamic characteristics and times. Figure 2(a) represents the various filling rates : A-filling-rate (r_a), B- filling-rate (r_b), and emptying-rate (r_c). Figure 2(b) shows the normal and fault cases of mixing effect (α) characteristics. In the fault case, broken propeller, α might be slowly decreased (propeller speed is reduced). Figure 2(c) shows the volume characteristics : liquid-A volume(V_a), liquid-B volume(V_b), and total volume(V_c). The concentration characteristics of normal and fault cases are shown in Figure 2(d). The concentrations of both liquids are exponentially changed toward the equilibrium value. Here we adopted the 2% steady value

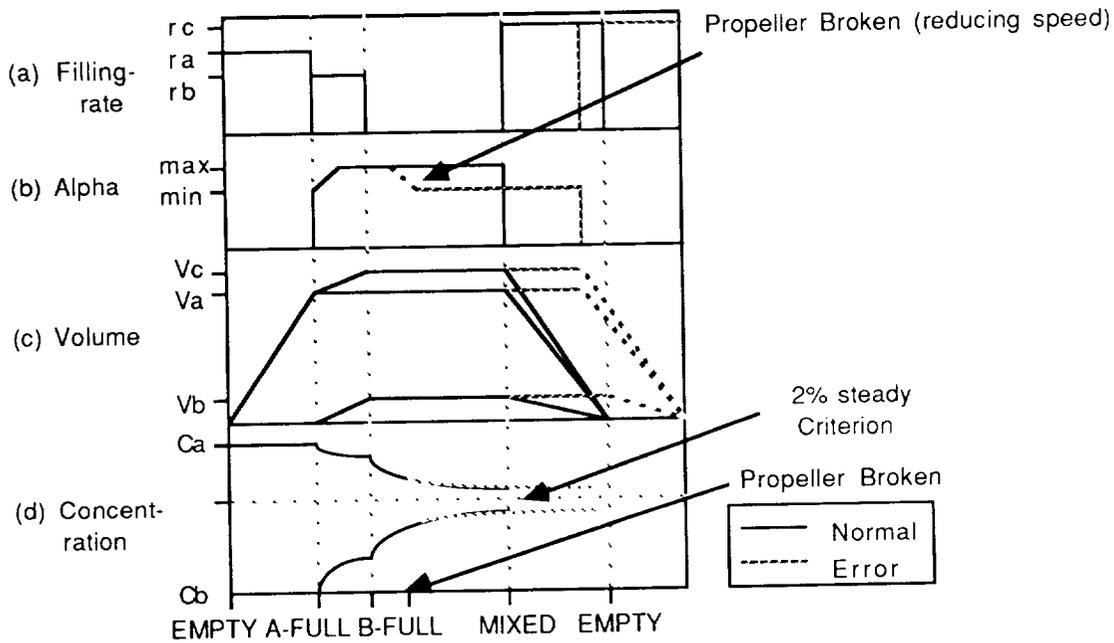


Figure 2: Dynamic Characteristics of Mixing System

criterion for reaching equilibrium level. Later, we will show how this system can be successfully controlled and how the error can be diagnosed.

IV. DEVS Representation of Mixing Process

In the DEVS representation of event-based control, a DEVS model moves through its checkstates in concert with external inputs, as long as those inputs arrive within the expected time windows. Associated with each checkstate are a minimum time and a window. In contrast to conventional sampled data systems, event-based logic does not require sensor output precision. Sensors can have threshold-like characteristics. Only two output states, for example, on/off, are needed although more may be employed. However, to generate the time windows the output states of the sensor must be accurately and reliably correlated with values of significant process variables. Figure 3(a) shows possible threshold-type sensors. A visual sensor in Figure 3(b) can provide more precise information for fault diagnosis. Figure 4 illustrates various data types of sensory inputs. The indicator used by controller keeps track of the container state estimated by level-sensors. However, the backup sensor, tube sensor and vision sensor provide more accurate container state. Therefore, by checking these various sensory sources, the diagnoser can do the fault diagnosis.

The control task is performed as the DEVS model of control system changes its state from an initial position on a given threshold sensor boundary to a succession, possibly cyclic, of boundaries. More concretely, this means we want the system to go through a predetermined sequence of states as reported by sensor readings. Our control logic will, as each boundary crossing is achieved, issue a control action, i.e., send an appropriate input to the system, in order to move toward the next desired boundary. The controller has a time window in which it expects the appropriate sensor states to change to confirm the expected boundary crossing. The time windows are derived from the DEVS external model of the system. Figure 5 presents the phase categorization in terms of the boundary conditions of the dynamic characteristics and their transition cycle with control commands. The transitions labelled by () are expected to take place within given time windows, as illustrated in the next section.

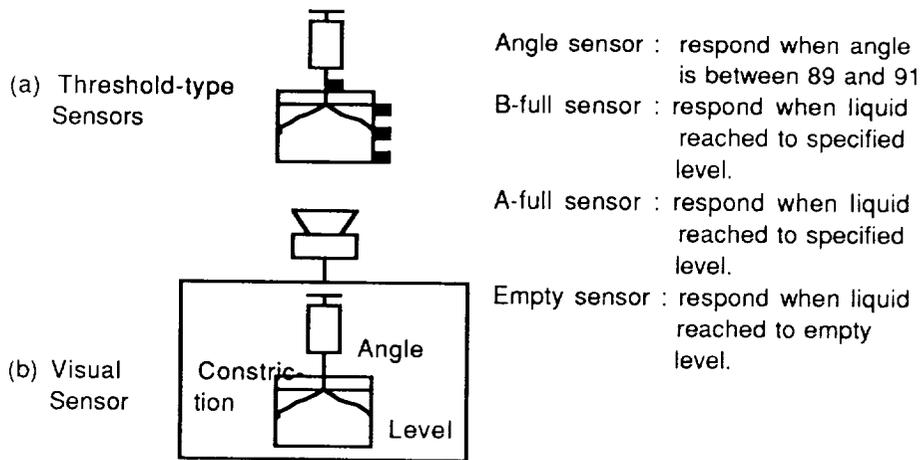


Figure 3: Sensory Inputs of Space Adapted Mixing System

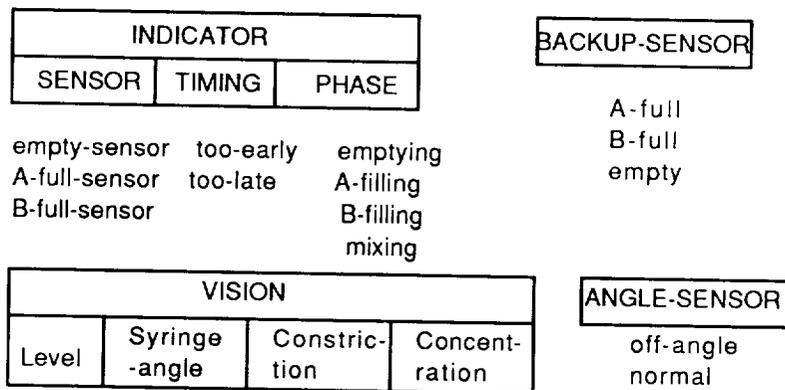


Figure 4: Data Types of Sensory Inputs

V. Simulation Approach of Mixing Process

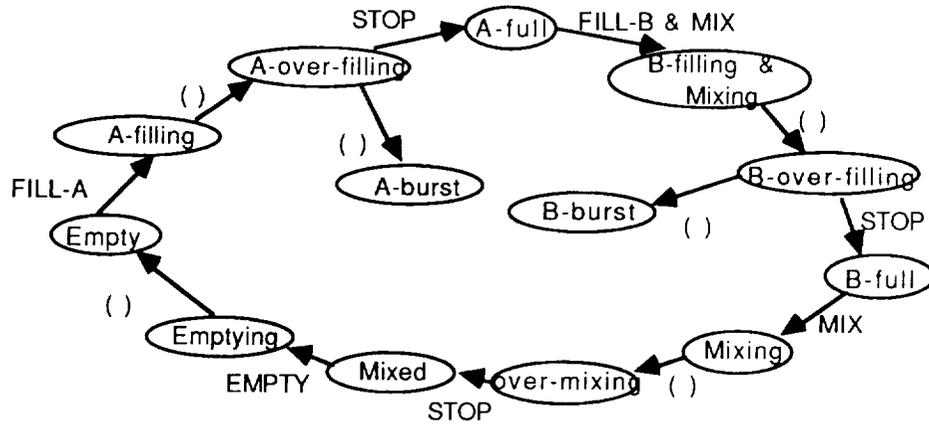
Figure 6 shows a simulation structure for the mixing process control. The simulation test is decomposed into a model representing the real bottle, MIX-E, and control-unit. The control-unit is decomposed into an operator for filling, mixing and emptying a container and a diagnoser for discovering the causes of any operational faults. There are three sub-models in the simulation :

MIX-E : Model of a space adapted mixing system. MIX-E is able to respond to both operational commands and diagnostic probes. It is external to the controller.

MIX-O : an operational model of the mixing system used by a controller, CONTRL, to generate its commands and verify the received sensor responses. Table 1 presents the operation table used in the MIX-O model. For example, the first column states that if the current state is EMPTY and the input is FILL-A then the next-state is A-FILLING; also the output in the current state is nil and its time-advance is infinity ($t_a = \text{inf}$). The time window of the next state is given by two fields : next-ta(20) gives its lower bound, and next-wind(6) gives its width (so its upper bound is 26).

MIX-D : a classification, or expert-system-like model, employed by the diagnoser inference engine, DIAGN, to determine the probable source of breakdown. As an example, an informal presentation of some of the diagnostic rules is given below :

R1 : If backup-sensor is not A-full and A-full-sensor is true, then "A-full-sensor is bad".



Empty	: empty sensor on
A-filling	: level < A-full level
A-over-filling	: A-full-sensor on. level > A-full level
A-burst	: Liquid-A overflow. level = TOP
A-full	: A-full-sensor on
B-filling & mixing	: A-full level < level < B-full-level. propeller sensor on.
B-over-filling	: B-full-sensor on. level > B-full level
B-burst	: Liquid overflow. level = TOP
B-full	: B-full-sensor on
Mixing	: Concentration > 2% criterion
Over-mixing	: Concentration < 2% criterion. Propeller on
Mixed	: Concentration < 2% criterion. Propeller off
Emptying	: empty level < level < B-full level

Figure 5: Phase Categorization and Phase Transition Diagram

R2 : If timing is too-late and phase is A-filling and level is less than 100, then “Tube is off-angle or leaky”.

R3 : If timing is too-late and phase is mixing and concentration-level is greater than 31, then “Propeller is broken”.

Figure 7 shows the hierarchical structure of the mixing control system. Level III is the lowest level, where the real system under control exists. The external model, MIX-E, receives input messages such as control commands and read-sensor commands from the next higher level, Level II. It sends the sensor readings to the next higher level. Level II corresponds to the control unit shown in Figure 6. It has an operational model, MIX-O, and a rule-based model, MIX-D, to provide necessary information for the controller and diagnoser, respectively. It also sends a result message to its next higher level, Level I, which contains the goal agent, the highest unit of the control system. The agent may represent a robotic or other autonomous decision maker.

VI. Simulation Results

To test the control logic for a mixing process, we have run several simulation experiments of a possible mixing process. The simulation experiments concern two cases : normal case and fault case. Initial values for an external model, MIX-E, under normal operation are given as follows ; $r_a = 5$, $r_b = 6.67$, $r_c = 8.3$, $V_a = 100$, $V_b = 66.6$, $C_a(0) = 50$, $C_b(0) = 0$, angle = 90, constriction effect = 5, leakage rate = 0.001, and $\alpha = 0.005$. We also assume several time delays of sensor readings, for example, 1 sec for backup-sensor

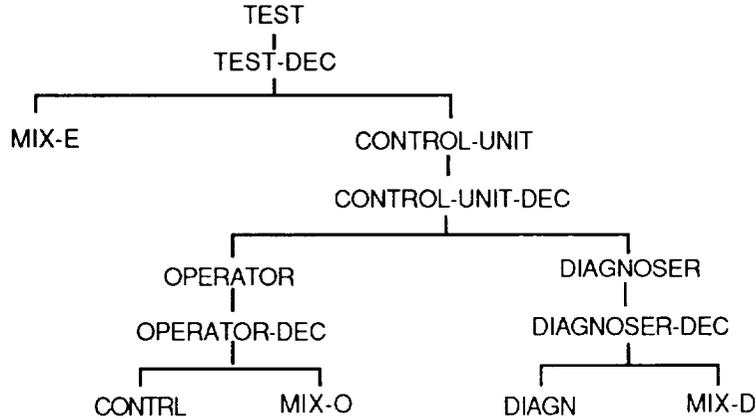


Figure 6: Hierarchical Simulation Structure for a Mixing System

state	input	next-state	output	ta	next-ta	next-wind
empty	fill-A	A-filling	()	inf	20	6
A-filling	()	A-over-filling	A-full-sensor	20	1000	20
A-over-filling	()	A-burst	()	1000	inf	()
A-over-filling	stop	A-full	()	1000	inf	()
A-full	fill-B&mix	B-filling	()	inf	10	4
B-filling	()	B-over-filling	B-full-sensor	10	1000	20
B-over-filling	stop	B-full	()	1000	inf	()
B-full	mix	mixing	()	inf	45	7
mixing	()	over-mixing	()	45	1000	20
over-mixing	()	inf-mixing	()	1000	inf	()
over-mixing	stop	mixed	()	1000	inf	()
mixed	empty	emptying	()	inf	20	6
emptying	()	empty	empty-sensor	20	inf	()

Table 1: Table Specification of an internal model, MIX-O

reading and 100 sec for visual-sensor reading. In the future, these time delays could be taken into account when deciding on sensors to interrogate.

The partial simulation results of normal and fault cases for a goal plan from B-full to MIXED are illustrated in Table 2(a) and (b), respectively. In the normal case, the controller, CONTRL, issues the control command, MIX, to the external model, MIX-E, and also to the internal model, MIX-O, and then waits for the sensory response during the scheduled time window (7 sec). If the sensory response arrives within the time window, the controller generates the next command and so on, till the MIX-E reaches to its goal state. But, in the fault case, where the propeller is broken during mixing (at clock time 35), the mixing effect, α , decreases from 0.08 to 0.05 (the base level) with decreasing rate 0.005 at each time step (see Figure 2(b)). In this case, Table 2(b) shows that there is no response from MIX-E before time step 86.3, the upper bound of the time window given by MIX-O. Therefore, the controller generates the error command, ERROR, to the diagnoser. The diagnoser, DIAGN, checks the data associated with the discrepancy, such as phase in which it occurred, and its timing. It also gets sensor data from MIX-E. The expert-system-like model, MIX-D, concludes "the propeller is broken" by using the data from DIAGN. This is because the indicator shows that its timing is TOO-LATE and current phase is MIXING but the visual sensor shows that the concentration has not reached to

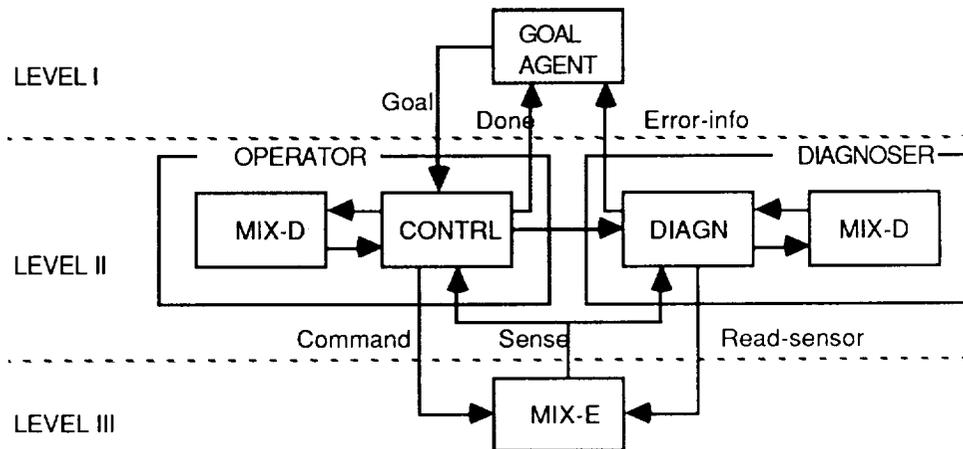


Figure 7: Block Realization of the Mixing Control System

clock	MIX-E		MIX-O		CONTRL	
	state	output	state	output	state	output
34.3	(mixing 45)	()	(mixing)	(mix 45 7)	()	(mix)
79.3	()	()	()	()	(window 7)	()
83.2	(over-mixing 200)	()	(mixed)	(stop)	(check 1)	()
84.2	(mixed)	()	()	()	()	(stop)

(a) Goal Plan : B-FULL -> MIXED (normal case)

clock	MIX-E		MIX-O		CONTRL		MIX-D		DIAGN	
	state	output	state	output	state	output	state	output	state	output
34.3	(mixing 45)	()	(mixing)	(mix 45 7)	()	(mix)	()	()	()	()
79.3	()	()	()	()	(wind 7)	()	()	()	()	()
86.3	()	()	()	()	(error)	(too-late)	()	()	(start 1)	()
87.3	(reading 1)	()	()	()	()	()	()	()	(wait-sensor)	()
88.3	(reading 10)	(B-full)	()	()	()	()	()	()	(wait-sensor)	()
98.3	(visual 100)	(normal)	()	()	()	()	()	()	(wait-sensor)	()
198.3	(finish-read)	(vision-info)	()	()	()	()	()	()	(wait-sensor)	()
199.3	(passive)	(done)	()	()	()	()	()	()	(start-diagn)	(sensor-
200.3	()	()	()	()	()	()	(passive)	(propeller	(passive)	data)
							-broken)			(propeller
										-broken)

(b) Goal Plan : B-FULL -> MIXED (error case)

Table 2: Simulation Results (partially shown)

its equilibrium value.

VII. Conclusions

This paper has shown how the space-adapted mixing control system is advantageously represented as discrete event models by employing techniques based on the DEVS formalism. Several fluid handling models have been successfully testing in the DEVS-Scheme environment. Suitably operating on the structure of such DEVS models provides a basis for design of event-based logic control. Since the DEVS formalism is at the heart of event-based control system design, such controllers can be readily checked by computer simulation prior to implementation. Thus the DEVS formalism plays the same role with respect to event-based control that differential and difference equation formalisms play to conventional control. This principle and the applicability of the DEVS-based control paradigm was illustrated here in the design of a fluid mixing system capable of supporting laboratory automation aboard a Space Station. The inclusion of event-based control units within robotic agents is discussed in (9, 10).

Acknowledgement

This research was supported by NASA-Ames cooperative agreement No. NCC 2-525, "A Simulation Environment for Laboratory Management by Robot Organizations".

References

- (1) Kim, T. G., "A Knowledge-Based Environment for Hierarchical Modelling and Simulation," Ph.D. Thesis, Dept. of ECE, University of Arizona, May, 1988.
- (2) Oren, T. I., "Taxonomy of simulation model processing," in *Encyclopedia of Systems and Control*, M. Singh, Eds. New York, NY : Pergamon Press.
- (3) Saridis, G. N., "Knowledge Implementation: Structures of Intelligent Control System," in *Proc. IEEE International Symposium on Intelligent Control*, 1987, pp. 9 - 17.
- (4) Zeigler, B. P., "DEVS Representation of Dynamical Systems : Event-based Intelligent Control," *IEEE proc.* Vol. 77, no. 1, Jan. 1989, pp. 72 - 80.
- (5) Zeigler, B. P., *Theory of Modelling and Simulation*, New York, NY : Wiley, 1976 (reissued by Krieger Pub. Co., Malabar, FL, 1985).
- (6) Zeigler, B. P., "System-theoretic representation of simulation models," *IIE Trans.*, Mar. 1984, pp. 19 - 34.
- (7) Zeigler, B. P., "Hierarchical, modular discrete event modelling in an object oriented environment," *Simulation*, Vol. 49, no. 5, 1987, pp. 219 - 230.
- (8) Zeigler, B. P., *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, London, 1984.
- (9) Zeigler, B. P., *Object-oriented Simulation with Hierarchical, Modular Models : Intelligent Agents and Endomorphic Systems*, Academic Press, 1990.
- (10) Zeigler, B. P., Cellier, F. E., and Rozenblit, J. W., "Design of a simulation environment for laboratory management by robot organizations," *J. Intelligent and Robotic Systems*, Vol. 1, 1988, pp. 299 - 309.

Dynamic Test Input Generation for Multiple-Fault Isolation

Phil Schaefer

Martin Marietta Advanced Computing Technology

P.O. Box 179, M.S. 4372

Denver, CO 80201

Abstract

Recent work in Causal Reasoning has provided practical techniques for multiple fault diagnosis. These techniques provide for a hypothesis/measurement diagnosis cycle. Using probabilistic methods, they choose the best measurements to make, then update fault hypotheses in response.

For many applications such as computers and spacecraft, few measurement points may be accessible, or values may change quickly as the system under diagnosis operates. In these cases, a hypothesis/measurement cycle is insufficient. This paper presents a technique for a hypothesis/test-input/measurement diagnosis cycle. In contrast to generating tests a priori for determining device functionality, it dynamically generates tests in response to current knowledge about fault probabilities. The paper shows how the mathematics previously used for measurement specification can be applied to the test input generation process. An example from an efficient implementation called MPC is presented.

I. Introduction

In recent years, AI techniques have proven useful for constructing fault diagnosis tools. A particularly interesting subset of these techniques is based on Causal Reasoning. Causal reasoning tools use a model of how the unit should behave, assuming no faults. This model can then be used to infer possible faults by comparing the observed behavior to the behavior predicted by the model.

Recent work has yielded techniques for multiple fault diagnosis using such techniques. The approach of (de Kleer and Williams), for example, provides an efficient framework for hypothesizing faults, given a set of measurements. At each step in diagnosis, it determines the most helpful measurement to make next. This provides a hypothesis/measurement diagnosis cycle.

A fault-isolation procedure based solely on a hypothesis/measurement cycle is often insufficient. Many complex systems such as computers and spacecraft are packaged in a way that makes measurement of most internal points time-consuming and expensive. Additionally, values within the system may not be static. For example, in a microprocessor-based system, data is sent over the databus to peripheral devices. Without sophisticated test equipment, the actual databus value cannot be measured directly, because it is present for only a fraction of a microsecond. For these reasons, it is often preferable to restrict measurement, when possible, to a few easily-accessible points. In this style of diagnosis, multiple test inputs are generated to observe the system in multiple states, rather than multiple measurements taken with the system in a single state. Recently, researchers have introduced off-line

techniques for purposes such as post-assembly checkout (Shirley). However, little has been presented about dynamically generating tests for isolating multiple faults.

The following sections present a technique for adapting the mathematics of hypothesis/measurement techniques for performing dynamic test input generation for multiple-fault isolation. Section II describes the causal models for which this technique applies. Section III presents the approach to the test generation and selection processes, and the techniques adopted for deducing the most probable faults. Finally, Section IV presents an example diagnosis using an implemented test-generation/fault-isolation system.

II. Causal Models for Multiple-Fault Diagnosis

Central to the causal-reasoning scheme is a causal model describing how the system under diagnosis properly functions. The causal model contains a description of the important points within the system, referred to as *elements*, and how the values of these elements cause and effect each other. Consider the system shown in Figure 1. It is a modified portion of an experiment control electronics subsystem concept being developed by Martin Marietta. This subsystem will be used as an example in the following sections. In the model, elements correspond to inputs and outputs of modules, as well as to a few module-internal points. Such a system can be modeled with our DEFCAUSAL syntax. For example, one operation of the Remote Interface Unit (RIU) is to write data to port EXP-CMD-3 when a :write-3

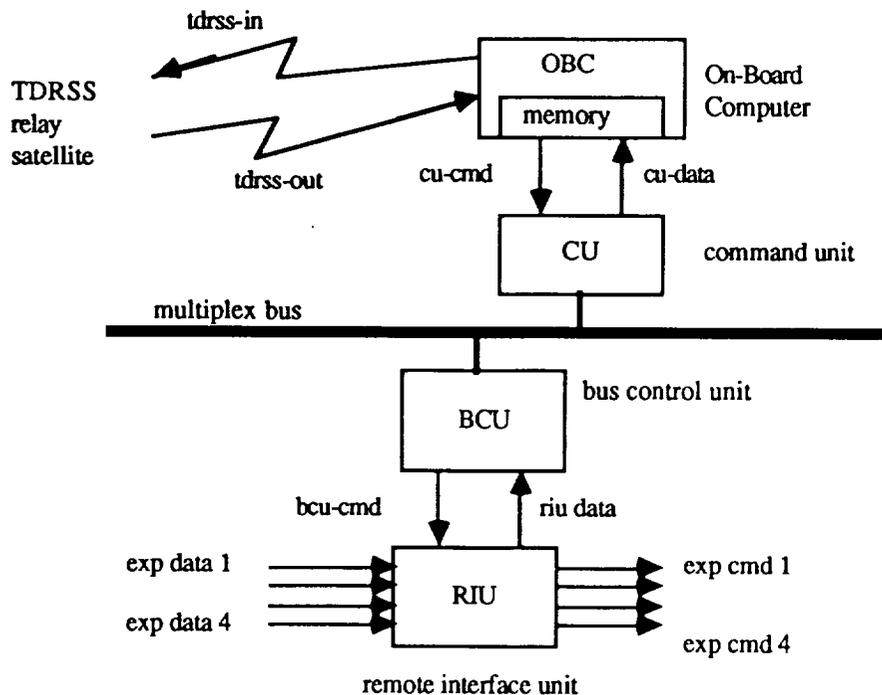


Figure 1. Example system for diagnosis Part of experiment control electronics subsystem. Commands sent and data gathered via the TDRSS relay satellite.

command is received on BCU-CMD. This relation is shown below (values starting with a "?" are variables).

```
(defcausal RIU
  (BCU-CMD (:write-3 ?data) :momentary) causes
  (EXP-CMD-3 ?data :continuing))
```

Another causal relation describes how the On-Board Computer (OBC), when in the :experiment mode, sends commands on the Command Unit (CU) line. The commands are sent from the OBC command-sequence memory:

```
(defcausal OBC
  (OBC_MEM ?memory)
  (MODE :experiment)
  (ELAPSED-TIME ?t) causes
  (CU-CMD (address ?t ?memory) :momentary)).
```

The :momentary and :continuing flags indicate the temporal relations of the causes and effects.

III. Dynamic Test Input Generation

An important part of diagnosis is to gain knowledge about the internal state of the malfunctioning system. This knowledge helps to decide which among several fault hypotheses is actually correct. In systems with many measurable internal points, this knowledge is gained by direct measurement, e.g., with voltmeters, logic probes, etc. When internal points are inaccessible, we must adopt a different means of obtaining this information. Our approach is based on a concept of *path generation*. In this approach, paths through the system are generated, which, if tested, will yield information about the internal system state. Our approach incorporates this idea with the following processes: (1) A Candidate Generator, which uses the measurements resulting from tests to produce fault hypotheses and their associated probabilities; (2) a Path Constructor, which suggests test paths through the model to gain information about the hypothesized faults; (3) a Path Selector, which chooses the path most helpful in discriminating between fault hypotheses; and (4) a Causal Planner, which produces a test input sequence to activate the selected path. Figure 2. presents a diagram of how these processes interact in a test-generation/fault-isolation system.

Candidate Generation

The Candidate Generator derives the fault hypotheses (candidates) implied by observed symptoms. It assigns probability estimates to each candidate.

Upon input of a symptom (an unexpected value at an observed element), the Causal Planner determines which causal relations imply the correct value, rather than the symptom value, was to be expected. This set is known as the *active relations set*. The Causal Planner produces this set as follows: given the correct outputs as goals and the inputs which were present when the symptom was observed

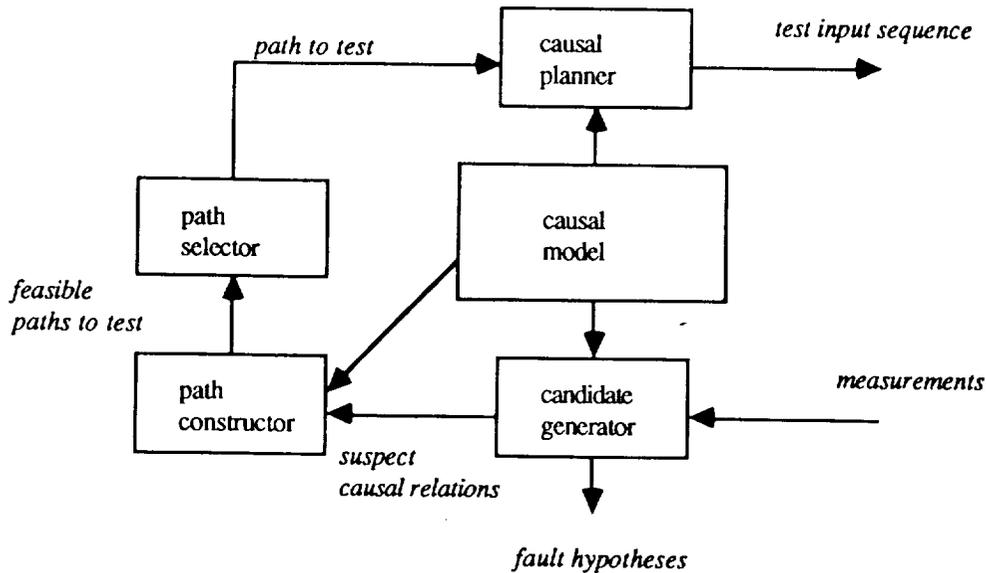


Figure 2. Test generation/ Diagnosis architecture.

as constraints on the solution, it generates a plan indicating the required inputs. During planning, the planner selects a causal relation to achieve each subgoal. When a solution is found, the set of relations selected in planning comprises the active relations set. This set can be described as a directed graph, relating input values, through intermediate values, to the desired output value. Assume, for example, that a value of 35 is desired on the TDRSS-IN line of the system of Figure 1. The resulting active relations set is shown in Figure 3. The active relations set describes the mechanisms which if functioning properly, would provide the correct behavior. When a failure symptom is observed, therefore, at least one of the active relations must not be functioning as specified. Conversely, if the correct output results, there is evidence that the active relations set is without fault.

One useful technique for generating candidates based on causal information has been described by (de Kleer and Williams). The technique maintains a set of *minimal candidates*, each of which must be able to account for all observed symptoms. Probability values are assigned to each candidate, using Bayesian probability concepts. Candidate generation, in our approach describing faults in terms of faulty active relations, is taken directly from (de Kleer and Williams), so will not be repeated here. A slight modification of the probability assignment approach is presented here.

A test *passes* if the value predicted by the model is observed at the test point. After each test is run, the probabilities of all candidates are updated, based on whether the test passes or fails. The probability that all of the relations in the candidate are faulty is assigned to each candidate. The probability of candidate C_j , with respect to its previous probability $P(C_j)$ is, by Bayes' rule (if the test fails)

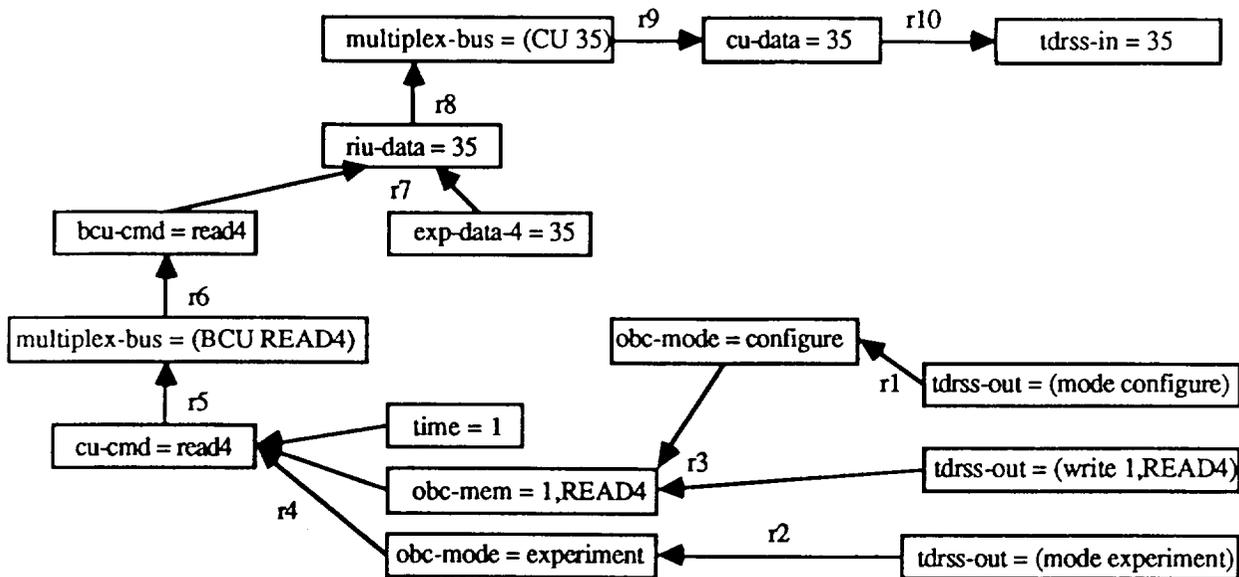


Figure 3. Active causal relations (ri) indicating why TDRSS-IN should equal 35. OBC memory is loaded with "read-4" in location 1. Then, in the "experiment" mode, when time=1, the OBC sequencer sends the command, which sends the result to TDRSS.

$$P(C_j | \text{fails}) = \frac{P(\text{fails} | C_j) P(C_j)}{P(\text{fails})}$$

where

$$P(\text{fails} | C_j) = \begin{cases} 0 & \text{if } C_j \notin \text{active-relations} \\ 1-r & \text{if } C_j \in \text{active-relations} \end{cases}$$

$$P(\text{fails}) = \sum_j^{C_j \in \text{active relations}} P(C_j) (1 - r)$$

and, if the test passes,

$$P(C_j | \text{passes}) = \frac{P(\text{passes} | C_j) P(C_j)}{P(\text{passes})}$$

where

$$P(\text{passes} | C_j) = \begin{cases} 1 & \text{if } C_j \notin \text{active-relations} \\ r & \text{if } C_j \in \text{active-relations} \end{cases}$$

$$P(\text{passes}) = \sum_j^{C_j \in \text{active relations}} P(C_j) r + \sum_j^{C_j \notin \text{active relations}} P(C_j)$$

As reflected above, even if a candidate is the fault, the correct value will in some cases result at the test point under the conditions of the test. This effect is approximated above with a constant, r , which indicates the probability that a particular relation will so behave.

Path Construction

Path construction is the basis of our test generation approach. The purpose of path construction is to generate a path using causal relations from a point internal to the system to a measurable point. By measuring the result at the output point, evidence about the internal state will be obtained.

When generating a test, we wish to obtain knowledge about which active relations have failed. The test path will therefore traverse causal relations from the active relations set. For example, given the active relations of Figure 3, each test path would pass through at least one of the relations r_1 through r_{10} . If an incorrect value results at the end of a path, there exist faults within the relations traversed by the path. Otherwise, evidence indicates faults probably do not exist in the path.

The process of path construction is depicted in Figure 4. The causal relation r_i takes the values of the elements e_i and causes the value at elements e_{i+1} . Assume that the relations r' are not within the active relations set. If e_i has the expected

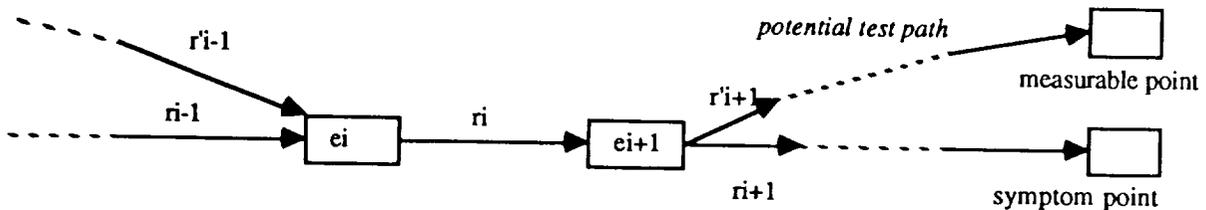


Figure 4. Path Construction for isolating possible fault of relation r_i (see text).

value, but e_{i+1} does not, r_i must be faulty. To discriminate between r_i and the relations $r_j > i$, paths not passing through r_j are needed. Paths through r_i and r'_{i+1} will therefore be constructed. Similarly, to discriminate between r_i and $r_j < i$, paths through r'_{i-1} and r_i are constructed. All of the constructed paths must terminate at a measurable element, so that measured values can be used to gain the required information.

To this end, the path constructor generates all paths through relations r_i of the active relations set, with and without alternative relations r' , terminating at measurable points. When a path is activated by a test input, the value at its measurable point is the test result.

Path Selection

The Path Selector chooses the path most useful to test, and gives it to the Causal Planner as a test goal.

The results of a particular test will give evidence about whether a fault exists within the active relations of the test. Therefore, to determine the usefulness of a path, it is necessary to know which causal relations need to be active to run the test. Figure 5 depicts this situation. The large triangle of Figure 5 represents the active relations which should have caused a correct value at the symptom-point. The small triangle represents the relations contributing to the expected values needed for the relation r_{i-1} . If the path r' were to be tested, the relations represented by the large dashed triangle of Figure 5 would be under test, because they are all needed to activate the path r' . The way to determine this set of relations is to run the Causal Planner, as described in "Candidate Generation." Given a large model and a large set of potential paths (a common occurrence), the time to evaluate all the options would be prohibitive (In the example model, the planner runs for about 5 seconds for each plan). An approximation is therefore used. The relations within the small triangle are already known, because they were determined

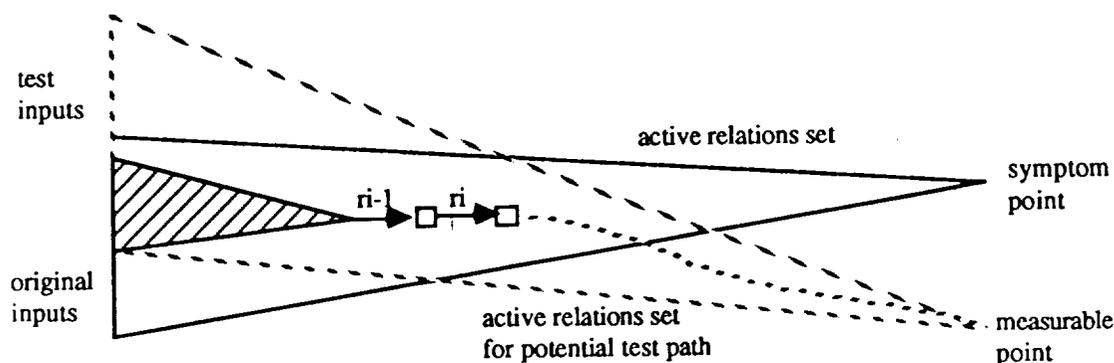


Figure 5. The active relations set for the test path is approximated by the relations comprising the small triangle union the relations forming the potential test path.

in finding the active relations leading to the original symptom. The union of that set and the set of relations used to construct the potential path can therefore approximate the active relations set. This approximation may skew path selection, by affecting the probability of the test passing, as indicated in the equations above. Fortunately, it will not affect accuracy of the test results, because the planner will later be run for the selected path to generate the test input. This will determine the exact active relations set.

At this point, the potential paths, each with the set of causal relations it tests, have been determined. The final step in path selection is to choose the set of relations most useful to test. Techniques similar to those presented in (de Kleer and Williams) apply to path selection. The essentials of the process are discussed here.

The best test is defined as that which minimizes the expected entropy of the candidate set, using the Information Theory definition of entropy

$$H = - \sum_j^{\text{candidates}} P(C_j) \log P(C_j).$$

As the probabilities move toward 0 or 1, this sum is minimized. The expected entropy resulting from a given test is

$$\hat{H} = \hat{H}(\text{passes}) p(\text{passes}) + \hat{H}(\text{fails}) p(\text{fails}).$$

In terms of the definition of entropy,

$$\hat{H}(\text{passes}) = - \sum_j^{\text{candidates}} P(C_j | \text{passes}) \log P(C_j | \text{passes})$$

$$\hat{H}(\text{fails}) = - \sum_j^{\text{candidates}} P(C_j | \text{fails}) \log P(C_j | \text{fails}).$$

The conditional probabilities of each candidate are as given in "Candidate Generation." The test which minimizes \hat{H} is selected as the best test to run next.

The final part of test generation is implemented by giving the Causal Planner the desired value of the measurable point (the output of the selected path) as a goal. The constraints on the plan are that the approximated active relations set is included in the solution. When the planner terminates, the plan produced is the test input.

IV. MPC- An Implementation of Test Input Generation

A computer program implementing the test-generation/fault-isolation architecture of Figure 2. has been implemented as part of the MPC (Multi-Purpose Causal) tool. It is implemented in Lisp on a Symbolics 3670. MPC accepts models described in the DEFCAUSAL syntax and currently has an interface requesting tests and accepting measurement results. It has been tested on several models, including an expanded version of the example presented here.

An example diagnosis session using MPC will now be described, indicating the operation of the various test-generation subsystems. Assume that the sequence of TDRSS commands (mode :configure), (write 1,(write-3 35)), and (mode :experiment) were sent to the system of Figure 1. As shown in Figure 3, a value of 35 on the EXP-CMD-3 control line would be expected. Assume that this value was not observed. MPC is therefore given EXP-CMD-3 as the initial symptom point. Ten candidates are generated, one for each of the potentially faulty relations shown in Figure 3. The candidate probabilities are as follows:

$$\{r1\} = .100, \{r2\} = .100, \{r3\} = .100, \{r4\} = .100, \{r5\} = .100, \\ \{r6\} = .100, \{r7\} = .100, \{r8\} = .100, \{r9\} = .100, \{r10\} = .100.$$

Paths from the points associated with these candidates to measurable outputs (EXP-CMD-1 - EXP-CMD-4 and TDRSS-IN) are generated. The most useful path, according to the entropy-measurement equations described in "Path Selection," is the path from the BCU-CMD element to the measurable point EXP-CMD-1. This selection corresponds to the intuitive "divide the problem in half" approach often

used by technicians. To generate a test of this path, the causal planner is invoked, and is constrained to use the causal relations r1 through r6 in the plan, as they were used to cause the point of interest BCU-CMD, as can be seen in Figure 3. The resulting plan is

```
TDRSS-OUT = (mode :configure)
TDRSS-OUT = (write 1, (write-1 35))
TDRSS-OUT = (mode :experiment).
```

MPC then prompts

```
Is the value at EXP-CMD-1 equal to 35?
```

Assume that the answer is "yes." The updated candidate probabilities are

```
{r1} = .039, {r2} = .039, {r3} = .039, {r4} = .039, {r5} = .039,
{r6} = .039, {r7} = .192, {r8} = .192, {r9} = .192, {r10} = .192,
```

indicating that the candidates describing the relations on the path from the experiment back to the TDRSS are most suspect. The paths from the active relations are once again evaluated. Based on the new candidate probabilities, however, the most useful path is from BCU-CMD to TDRSS-IN, but using a different causal relation from BCU-CMD. The path selected goes through the relation r11, using EXP-DATA-1, rather than the original EXP-DATA-4. The resulting plan is

```
TDRSS-OUT = (mode configure)
TDRSS-OUT = (write 1, read-1)
EXP-DATA-1 = 35
TDRSS-OUT = (mode experiment),
```

followed by the prompt

```
Is the value of TDRSS-IN equal to 35?
```

If the answer to the test is "yes," the probabilities indicate a strong preference for a single candidate, indicating that r7 is the faulty mechanism:

```
{r1} = .022, {r2} = .022, {r3} = .022, {r4} = .022, {r5} = .022, {r6} = .022,
{r7} = .543, {r8} = .109, {r9} = .109, {r10} = .109.
```

If this amount of convergence is sufficient to terminate testing, MPC reports its findings. Because R7 is implemented in the RIU module, RIU is reported as the suspect module. These results were obtained by using a value of .2 for r in the probability equations. With a smaller value, the convergence on the candidate (r7) would have been faster.

Conclusions

The test generation architecture implemented in the MPC system contributes a new tool to the set of causal reasoning capabilities now available. For systems in which few points are accessible, or in which transient effects are important, it provides a means to dynamically generate tests in response to observed symptoms.

The MPC approach is an extension to several other causal-reasoning efforts. Sharing some of the techniques of (Shirley), it generates tests to narrow down fault hypotheses, rather than to test specific components. It makes use of the probabilistic hypothesis generation and belief ideas of (de Kleer and Williams), but for test-generation purposes. This use of probability avoids the need for the "evidence model" required in the approach described in (Schaefer).

The current approach assumes that when a causal relation fails, the physical failure is in the device designed to implement the relation. Occasionally, however, another device may have a failure, such as a short circuit, which interferes to cause the relation to fail. Using the model to explore these possibilities, making use of "Pathways of Interactions" techniques similar to (Davis), is a topic of ongoing research. Other extensions include more sophisticated techniques for explaining the significance of test results to the user.

References

1. De Kleer, J., and B.C. Williams, "Diagnosing Multiple Faults," **Artificial Intelligence**, vol. 32, nr. 1, pp. 97-130, 1987.
2. Shirley, M. H., "Generating Tests by Exploiting Designed Behavior," **Proc. AAAI-86**, pp. 884-890, 1986.
3. Schaefer, P. R., "Higher Level Causal Reasoning for Diagnosis," **Proc. IEEE Int'l Workshop on AI for Industrial Appl.**, pp. 33-38, 1988.
4. Davis, Randall, **Diagnostic Reasoning Based on Structure and Behavior**, AI Memo 739, Massachusetts Institute of Technology, 1984.

Genetic Algorithm Based Fuzzy Control
of Spacecraft Autonomous Rendezvous

C. L. Karr¹, L. M. Freeman², and D. L. Meredith³

¹U.S. Bureau of Mines, Tuscaloosa Research Center,
Tuscaloosa, AL 35487-9777

²The University of Alabama, Department of Aerospace Engineering,
Tuscaloosa, AL 35487-0280

³The University of Alabama, Department of Computer Science,
Tuscaloosa, AL 35487-0290

ABSTRACT

The U.S. Bureau of Mines is currently investigating ways to combine the control capabilities of fuzzy logic with the learning capabilities of genetic algorithms. Fuzzy logic allows for the uncertainty inherent in most control problems to be incorporated into conventional expert systems. Although fuzzy logic based expert systems have been used successfully for controlling a number of physical systems, the selection of acceptable fuzzy membership functions has generally been a subjective decision. In this paper, high-performance fuzzy membership functions for a fuzzy logic controller that manipulates a mathematical model simulating the autonomous rendezvous of spacecraft are learned using a genetic algorithm, a search technique based on the mechanics of natural genetics. The membership functions learned by the genetic algorithm provide for a more efficient fuzzy logic controller than membership functions selected by the authors for the rendezvous problem. Thus, genetic algorithms are potentially an effective and structured approach for learning fuzzy membership functions.

INTRODUCTION

In recent years, rule-based systems have become increasingly popular as practical applications of artificial intelligence. These expert systems have performed as well as humans in several problem domains¹⁷, however their lack of flexibility in representing the subjective nature of human decision-making limits their performance in control problems. The uncertainty inherent in human decision-making can be incorporated into expert systems via fuzzy set theory¹⁸. In fuzzy set theory, abstract or subjective concepts can be represented with linguistic variables. Linguistic variables have been used in expert systems in the form of fuzzy logic controllers (FLCs)^{1,12,13}.

FLCs are rule-based systems that use fuzzy linguistic variables to model a human's "rule-of-thumb" approach to problem solving. FLCs have been used in a number of control problems¹⁶. These "fuzzy expert systems" include rules to direct the decision process, and membership functions for categorizing the precise numeric variable values as linguistic variables and vice versa. The rule set is gleaned from a human expert's experience and the membership functions are chosen by the FLC developer to represent the human expert's interpretation of the linguistic variables. A change in the membership functions alters the performance of the controller because it is the membership functions that determine when a given rule is eligible to be put into effect. Thus, the performance of the FLC is restricted by the choice of membership functions (given a set of rules).

Procyk and Mamdani¹⁵ introduced an iterative procedure for altering membership functions to improve the performance of an FLC, but in general, little has been done to develop a method for choosing membership functions that optimize the performance of an FLC. A standard method for determining the membership functions that produce maximum FLC performance is needed, yet selecting such a method poses a substantial problem due to the nonlinearity present in the search.

A search technique that is finding increasing popularity in the field of optimization is the genetic algorithm (GA)⁶. GAs are search algorithms based on the mechanics of natural genetics; they use operations found in natural genetics to guide their trek through the search space. GAs search through large spaces quickly, requiring only objective function value information to guide their search, an inviting characteristic since the majority of commonly used search techniques require derivative information, continuity of the search space, or complete knowledge of the objective function to guide their search. Occasionally these restrictions prove to be inconvenient if not insurmountable. Furthermore, because of the processing leverage associated with GAs, they take a more global view of the search space than many methods encountered in engineering optimization practice⁵. These favorable characteristics of GAs have been theoretically investigated by Holland's⁷ monograph. Empirical investigations by Hollstien⁸ and De Jong⁴ have demonstrated the technique's efficiency in function optimization. De Jong, in particular, establishes the GA as a robust search technique--one that is efficient across a broad spectrum of problems--as compared to several traditional schemes. Subsequent application of GAs to the search problems of pipeline engineering, very large scale integration (VLSI) microchip layout, structural optimization, job shop scheduling, medical image processing, and machine learning⁹ adds considerable evidence to the claim that GAs are broadly based.

The robust nature of GAs make them inviting tools for learning fuzzy membership functions. A GA has in fact been successfully used to learn high-performance fuzzy membership functions employed by a liquid level FLC¹⁰. In this application, an FLC was developed to control a mathematical model of a vessel containing liquid. The simple system is governed by a first order ordinary differential equation. All control decisions performed on the system involved altering one control variable based on the state or condition of two decision variables.

In this paper, an FLC is developed to control a mathematical that simulates the autonomous rendezvous of two spacecraft, one actively performing the rendezvous (the chaser), the other passively orbiting (the target). The rendezvous problem has received attention in the literature¹⁴ and is a challenging control problem. The FLC is capable of performing the rendezvous of the vehicles as predicted by a mathematical model of the rendezvous system. Next, a GA learns more efficient fuzzy membership functions to be used with the FLC rules. Based on the results presented, GAs appear to be effective, versatile, and straightforward enough to learn high-performance membership functions in complex control problems.

THE PHYSICAL SYSTEM

Spacecraft rendezvous operations are of importance to many forthcoming space activities. Of increasing interest are those operations which are conducted autonomously, i.e., without a human pilot operating the control systems of the chaser spacecraft. This paper is concerned with the "terminal" phase of the rendezvous maneuver prior to docking.

Since terminal navigation is accomplished using sensors onboard the chase vehicle to measure relative range and closure rates, a relative coordinate system is a logical choice. When the equations of motion are expressed in a relative coordinate frame, the coordinates are given explicitly as functions of time. In this study, the relative coordinate system (Figure 1) is fixed to the target vehicle with the y-axis directed radially from the earth, the x-axis tangent to the orbit in the direction opposite the target's motion, and the z-axis normal to the plane of the orbit and forming the customary right hand system.

The complete differential equations of relative motion are easily derived⁹. When the distance between the spacecraft is small compared to the distance from the target to the center of the earth, the three equations of relative motion reduce to the Clohessy and Wiltshire (sometimes called Hill) equations³:

$$\begin{aligned}\ddot{x} - 2\omega\dot{y} &= F_x/m, \\ \ddot{y} + 2\omega\dot{x} - 3\omega^2y &= F_y/m, \\ \ddot{z} + \omega^2z &= F_z/m,\end{aligned}$$

where ω is the angular velocity of the relative coordinate system origin rotating about the earth; F_x , F_y , and F_z are thrust forces; and m is the mass of the chaser vehicle.

In the present study the equations of motion are numerically integrated using a predictor-corrector linear multistep method, specifically the explicit Adams-Bashforth 3-step method for the predictor and implicit Adams-Moulton 3-step method for the corrector².

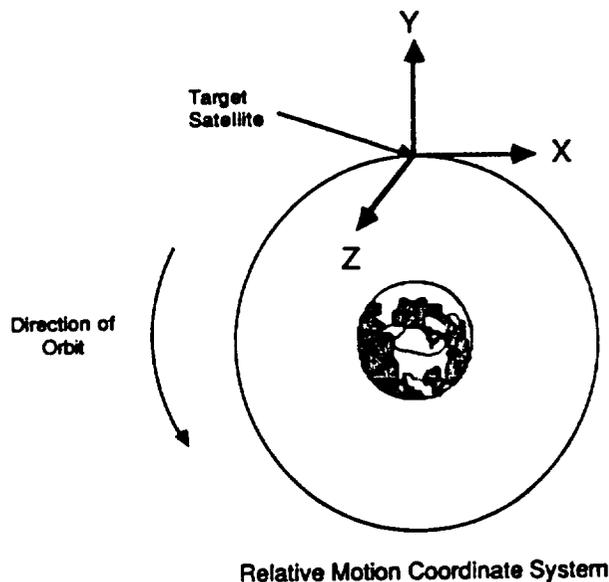


Figure 1.--The coordinate system is fixed to the target spacecraft.

A FUZZY PROCESS CONTROLLER

There are numerous approaches to developing FLCs. Generally, a compositional rule of inference¹²--a mathematical statement describing how the linguistic variables are to be manipulated--is employed to control the problem environment. In this paper, a hands-on, rational approach to the development of FLCs is taken. A step-by-step fuzzy procedure for controlling the rendezvous system is provided. This procedure is written in a generic form so that it may be easily adapted for the development of other FLCs.

The first step in developing the rendezvous FLC is to determine which variables will be important in choosing an effective control action. Six decision variables are readily identified as being important in the rendezvous control system. First, the current position of the chaser spacecraft relative to the target vehicle (identified with three variables x , y , and z) is important because it is this position that the FLC must drive to a setpoint (a close rendezvous with the satellite). Second, the time rate of change of the position (identified with three variables \dot{x} , \dot{y} , and \dot{z}) is important because it describes the relative velocity of the chaser spacecraft and becomes pertinent to the decision as the setpoint is approached.

Once the decision variables have been chosen, the control variables must be identified. In the rendezvous problem there are only three parameters that can be adjusted to alter the position and velocity of the spacecraft: the specific thrust in the three respective directions, $T_x = F_x/m$, $T_y = F_y/m$, and $T_z = F_z/m$.

Once the important decision and control variables have been identified, the linguistic variables that will be used to describe these variables must be defined (fuzzy classes). For the terminal rendezvous system, six fuzzy classes are used to characterize each of the six decision variables:

NEGATIVE-BIG (NB), NEGATIVE-SMALL (NS), NEGATIVE-CLOSE (NC), POSITIVE-CLOSE (PC), POSITIVE-SMALL (PS), and POSITIVE-BIG (PB). Five fuzzy classes are used to characterize the specific thrusts, T_x , T_y , and T_z : NEGATIVE-BIG (NB), NEGATIVE-SMALL (NS), NO CHANGE (N_C), POSITIVE-SMALL (PS), and POSITIVE-BIG (PB). These fuzzy classes were chosen because they are similar to the descriptive terms a human operator might use when attempting to rendezvous the spacecraft.

The choice of fuzzy classes described above allows for the possibility of 36 different conditions that could exist in each of the x, y, and z coordinate direction when the rules are of the form (example for x direction):

$$\text{IF } [x \text{ is } A \text{ AND } \dot{x} \text{ is } B] \Rightarrow \text{THEN } [T_x \text{ is } C]$$

where A , B , and C are fuzzy classes characterizing the respective variables. The individual coordinate directions are considered separately in this rule which would apply when x, y, or z were large since a human operator would likely consider only the two decision variables in a single coordinate direction at a time. Additionally, rules are added to take into account the coupling of x and y, but only when the chaser is near the target. The 108 rules of the above form (36 rules for each of the 3 coordinate directions) are certainly not adequate to control the spacecraft since the equations of motion are coupled (the x and y equations), requiring the FLC to consider the x and y coordinate directions collectively. This coupling effect becomes particularly important when the chaser spacecraft is near the target. Therefore, a set of "coupled rules" is needed. These rules are of the form:

$$\text{IF } [x \text{ is } A \text{ AND } y \text{ is } B \text{ AND } \dot{x} \text{ is } C \text{ AND } \dot{y} \text{ is } D] \Rightarrow \text{THEN } [T_x \text{ is } E \text{ AND } T_y \text{ is } F]$$

where A , B , C , D , E , and F are fuzzy classes characterizing the respective variables. These rules are written for all of the possible combinations of the terms when the four relevant decision variables are in the NEGATIVE-CLOSE or POSITIVE-CLOSE classes. Thus, there are 16 coupled rules which when combined with the 108 original rules provide 124 total rules.

The driving force behind an FLC is the idea that some uncertainty exists in categorizing the values of the control variables; the linguistic variables mean different things to different people. As a result, there must exist some mechanism for interpreting the fuzzy classes. This mechanism is the fuzzy membership function. The fuzzy membership functions used in the rendezvous FLC to characterize relative positions are shown in Figure 2. Similar membership functions were used to characterize relative velocities and thrusts. Fuzzy membership functions allow the precise numeric values of the decision variables to be transformed into a fuzzy class and the fuzzy control actions of the production rules to be transformed into precise, discrete control actions. Actually, fuzzy membership functions are approximations to the confidence with which a precise numeric value is described by a fuzzy class, and fuzzy membership function values (μ) are numeric representations of these confidences. For example, an x of 1300 ft might be viewed as POSITIVE-BIG ($\mu_{PB}(1300) = 0.18$), POSITIVE-SMALL ($\mu_{PS}(1300) = 0.43$), POSITIVE-CLOSE ($\mu_{PC}(1300) = 0.0$), NEGATIVE-CLOSE ($\mu_{NC}(1300) = 0.0$), NEGATIVE-SMALL ($\mu_{NS}(1300) = 0.0$) or NEGATIVE-BIG ($\mu_{NB}(1300) = 0.0$). When a fuzzy membership function has a value of $\mu = 1$, the confidence level of the precise numeric value being accurately described by the fuzzy class is high. On the other extreme, when $\mu = 0$, the confidence level of the precise numeric value being accurately described by the fuzzy class is low. It is important to realize that for each precise decision value, each fuzzy class has a membership function value, i.e., $x = 1300$ ft is POSITIVE-SMALL with a certainty of 0.43, POSITIVE-BIG with a certainty of 0.18, and described by each of the other classes with a certainty of 0.0.

Now that the precise numeric conditions existing in the rendezvous system at any given time can be categorized in a fuzzy class with some certainty, a process for determining a precise action to take on the rendezvous system must be developed. This process involves rules which must provide a fuzzy action for any condition that could possibly exist in the problem environment. Therefore, a human expert provides a fuzzy action for each condition possible in the environment (there are 124 rules in the rendezvous FLC). The formation of the rule set is comparable to the process that must exist

in the development of any expert system, except the rules incorporate linguistic variables that human operators are generally comfortable using. An example of a fuzzy rule used in the rendezvous system follows:

IF [y is NEGATIVE-BIG AND y is NEGATIVE-BIG] => THEN [T_y is POSITIVE-BIG].

This sample rule simply says that if the chaser spacecraft is well below the target vehicle and moving rapidly toward the earth, the thrust in the y direction should be increased as much as possible.

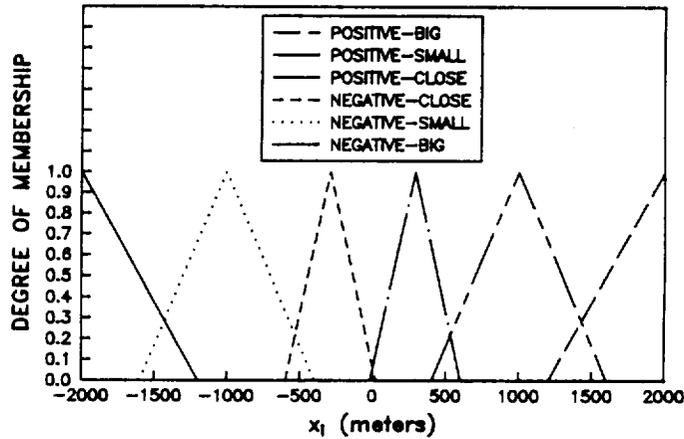


Figure 2.--The fuzzy membership functions shown provide a mechanism for converting a numeric value of position ($x_i = x, y, z$) into a fuzzy class.

At this point a means for converting a precise set of conditions existing in the rendezvous system to a set of fuzzy conditions, and a set of fuzzy rules prescribing a fuzzy action associated with a particular set of fuzzy conditions have been developed. There still remains the task of converting the 124 fuzzy actions provided by the fuzzy rules into a single, precise action to be taken on the chaser spacecraft system. Larkin¹¹ found that a procedure known as the center of area (COA) scheme is an efficient method for determining this precise action. In the COA method, a single action is defined by weighting the 124 individual actions described by the rules in proportion to the confidence that exists in each rule. The rules with high degrees of membership play proportionately higher roles in determining the action to be taken on the system.

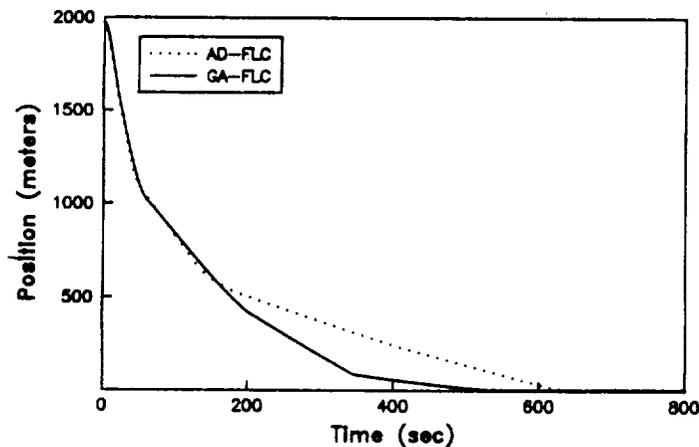


Figure 3.--The author-developed (AD) FLC is able to successfully rendezvous the spacecraft in approximately 300 seconds.

As a demonstration of the effectiveness of this fuzzy approach to control, consider a computer program that implements an FLC for manipulating a mathematical model of the rendezvous system. Figure 3 shows the position of the chaser relative to the target as a function of time for one particular initial condition for the problem environment. The FLC uses only the set of 124 fuzzy rules to govern its selection of actions. The FLC is able to rendezvous the spacecraft in approximately 600 seconds. In the remainder of this paper, a technique for improving the selection of membership functions (a GA) is introduced and applied to the rendezvous FLC.

THE MECHANICS OF A SIMPLE GENETIC ALGORITHM

GAs are powerful search algorithms based on the mechanics of natural genetics. They ensure the proliferation of quality solutions while investigating new solutions via a systematic information exchange that utilizes probabilistic decisions. It is this combination which allows GAs to exploit historical information to locate new points in the search space with expected improved performance.

GAs are unlike many conventional search algorithms in the following ways:

- 1) GAs consider many points in the search space simultaneously, not just a single point;
- 2) GAs work directly with strings of characters representing the parameter set, not the parameters themselves;
- 3) GAs use probabilistic rules to guide their search, not deterministic rules.

These differences establish inviting characteristics for a search technique. Namely, these differences preclude the requirement of derivative information and continuity of the search space. For this reason, GAs avoid convergence to local optima.

A simple GA that has given good results in a variety of engineering problems is composed of three operators: (1) reproduction, (2) crossover; and (3) mutation. These operators are implemented by performing the basic tasks of copying strings, exchanging portions of strings, and generating random numbers; tasks that are easily performed on a computer. Before looking at the operators, consider the overall processing of a GA during a single generation. The GA begins by randomly generating a population of N bit strings each of length ℓ . Each string represents one possible combination of the parameter set; one possible solution to the problem (in this case, one particular set of fuzzy membership functions). Although the coding of parameter sets to bit strings may at first seem to be a problem, many imaginative codings exist for representing large parameter sets. Each of the N strings is decoded so that the character strings yield the actual parameters. The parameters are sent to a mathematical model of the rendezvous process, evaluated with some objective function (i.e., told how good an FLC the parameters produce), and assigned a fitness value which is simply a measure of relative worth (a reward based on the quality of the solution). This fitness is then used when employing the three operators that produce a new population of strings (a new generation). Hopefully, this new generation will contain more efficient membership functions. The new strings are again decoded, evaluated, and transformed using the basic operators. The process continues until convergence is achieved or a suitable solution is found.

Reproduction is simply a process by which strings with large fitness values, good solutions to the problem at hand, receive correspondingly large numbers of copies in the new population. In this study use is made of tournament selection. In tournament selection, pairs of strings compete with each other on a head-to-head basis for the right to be reproduced in the next generation. The participants in these competitions are selected in a probabilistic fashion based on the relative fitnesses of the strings. Once the strings are reproduced, or copied for possible use in the next generation, they are placed in a mating pool where they await the action of the other two operators.

The systematic information exchange utilizing probabilistic decisions is implemented by the second operator, crossover. Crossover provides a mechanism for strings to mix and match their desirable qualities through a random process. After reproduction, simple crossover proceeds in three steps. First, two newly reproduced strings are selected from the mating pool produced by

reproduction. Second, a position along the two strings is selected uniformly at random. This is illustrated below where two binary coded strings A and B of length six are shown aligned for crossover:

$$\begin{array}{l} A = 110 \boxed{101} \\ B = 001 \boxed{011} \end{array}$$

Notice how crossing site 3 has been selected in this particular example through random choice, although any of the other four positions were just as likely to have been selected. The third step is to exchange all characters following the crossing site. The two new strings following this crossing are shown below as A' and B':

$$\begin{array}{l} A' = 110011 \\ B' = 001101 \end{array}$$

String A' is made up of the first part of string A and the tail of string B. Likewise, string B' is made up of the first part of string B and the tail of string A. Although crossover has a random element, it should not be thought of as a random walk through the search space. When combined with reproduction, it is an effective means of exchanging information and combining portions of high quality solutions.

Reproduction and crossover give GAs the majority of their search power. The third operator, mutation, enhances the ability of the GA to find near optimal solutions. Mutation is the occasional alteration of a value at a particular string position. It is an insurance policy against the permanent loss of any simple bit. A generation may be created that is void of a particular character at a given string position. For example, a generation may exist that does not have a one in the third string position when, due to the chosen coding, a one at the third position may be critical to obtaining a quality solution. Under these conditions, neither reproduction nor crossover will ever produce a one in this third position in subsequent generations. Mutation, however, causes a zero in the third position to be changed to a one occasionally. Thus, the critical piece of information can be reinstated into the population. Although mutation can serve a vital role in a GA, it should be noted that it occurs with a small probability (on the order of one mutation per thousand string positions), and is secondary to reproduction and crossover.

This has been a brief overview of a simple three-operator GA. For a more complete description of GAs including discussions of coding schemes, high-order operators, and fitness assignment, reference should be made to Goldberg⁶.

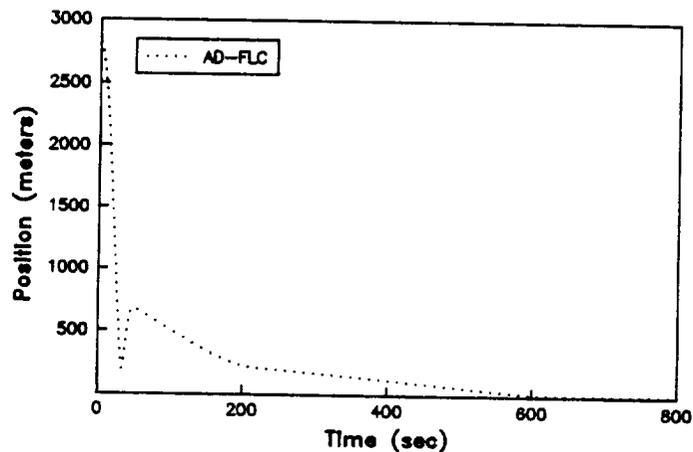
GENETIC ALGORITHM LEARNS FUZZY MEMBERSHIP FUNCTIONS

In this section, a three-operator GA learns membership functions that produce a rendezvous FLC that is more efficient than the author-developed FLC. The GA is essentially assigned the task of learning the proper definition of the linguistic variables as required for optimal performance when used with the given rule set.

An objective function was developed that tended to force the GA to locate membership functions that drove the relative position of the spacecraft to zero and held it there. The objective function tracked the vehicles for 8000 seconds. For a given set of membership functions, an error (where error is the sum of the absolute values of the relative distances between the chaser and the target taken over time) was calculated for a particular initial condition with the intent of using a GA to minimize this error term. So that a general purpose set of membership functions would be developed, four different initial conditions were considered in the evaluation of each bit string. The four initial conditions were chosen to ensure that the FLC could rendezvous the spacecraft as effectively when the chaser was approaching the target from a higher orbit as it could when it was approaching from a lower orbit.

The GA learned membership functions that provided for better control than those defined by the authors in the previous section. Figure 4 compares the GA-FLC to the author-developed FLC for one of the four initial conditions considered. In each initial condition case, the GA-FLC completed the spacecraft rendezvous more favorably than the author-developed FLC.

The desirable characteristics of the rendezvous must be conveyed to the GA through the objective functions. In this study, the objective of the GA was simply to reduce the total distance between the two vehicles summed over a period of time. The GA accomplished this goal. The FLC can be forced to exhibit other properties by altering the objective function. For example, if it is important to prevent the spacecraft from overshooting the target in the x-coordinate, the GA can be dissuaded from locating such solutions by incorporating a penalty into the objective function (see Goldberg⁶ for information on penalty methods in objective function formulation).



SUMMARY

In this paper, a GA was used to improve the performance of an FLC. Initially, an approach to FLC development was outlined. The approach was developed to be straightforward and intentionally avoided an abundance of fuzzy mathematics. Next, this approach was used to develop a fuzzy system for controlling the rendezvous of a spacecraft. The FLC was able to maintain control over the spacecraft (as simulated by a computer) by relying exclusively on fuzzy rules to determine its next action.

Altering the membership functions used in an FLC affects the performance of the controller, and the selection of appropriate membership functions can be cast in the light of a search problem. A simple three-operator GA was used to learn high-performance membership functions for the rendezvous FLC. The GA-FLC outperformed the author-developed FLC on four specific initial conditions chosen to represent a cross-section of potential initial conditions.

REFERENCES

1. Assilian, S., 1974, "Artificial Intelligence in the Control of Real Dynamic Systems," Ph.D. Thesis, Queen Mary College, London, 215 pp.
2. Burden, R. L., Faires, J. D., and Reynolds, A. C., 1978, Numerical Analysis, Prindle, Weber, & Schmidt, Boston, MA, 579 pp.

3. Clohessy, W. H., and Wiltshire, R. S., 1960, "Terminal Guidance System for Satellite Rendezvous," Journal of the Aerospace Sciences, Sept., pp. 653-674.
4. De Jong, K. A., 1975, "Analysis of the Behavior of a Class of Genetic Adaptive Systems," Ph. D. Thesis, University of Michigan, Ann Arbor, 256 pp.
5. Goldberg, D. E., 1983, "Computer-Aided Gas Pipeline Operation Using Genetic Algorithms and Rule Learning," Dissertation Abstracts International, Vol. 44, No. 10, p. 3174B.
6. Goldberg, D. E., 1989, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, Reading, MA, 412 pp.
7. Holland, J. H., 1975, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, 183 pp.
8. Hollstien, R. B., 1971, "Artificial Genetic Adaptation in Computer Control Systems," Ph. D. Thesis, University of Michigan, Ann Arbor, 297 pp.
9. Kaplan, M. H., 1976, Modern Spacecraft Dynamics & Control, John Wiley & Sons, New York, 415 pp.
10. Karr, C. L., Meredith, D. L., and Stanley, D. A., 1990, "Fuzzy Process Control with a Genetic Algorithm", Proceedings, Control '90 Mineral/Metallurgical Processing-Second International Symposium, Salt Lake City, UT, February.
11. Larkin, L. I., 1985, "A Fuzzy Logic Controller for Aircraft Flight Control," Industrial Applications of Fuzzy Control, Sugeno, M., Ed., North-Holland, Amsterdam, pp. 87-104.
12. Mamdani, E. H., 1974, "Application of Fuzzy Algorithms for Control of Simple Dynamic Plant," Proceedings, IEEE, Vol. 121, No. 12, pp. 1585-1588.
13. Mamdani, E. H., and Assilian, S., 1975, "An Experiment in Linguistic Synthesis with a Fuzzy Logic Controller," International Journal of Man-Machine Studies, Vol. 7, pp. 1-13.
14. Natenbruk, P., and Ragnitt, D., 1983, "Control Aspects as Elaborated in Space Rendezvous Simulations," Conference Proceedings on Guidance and Control Techniques for Advanced Space Vehicles, Florence, Italy.
15. Procyk, T. J., and Mamdani, E. H., 1978, "A Linguistic Self-Organising Process Controller," Automatica, Vol. 15, pp. 15-30.
16. Sugeno, M., Ed., 1985, Industrial Applications of Fuzzy Control, North-Holland, Amsterdam, 269 pp.
17. Waterman, D. A., 1970, A Guide to Expert Systems, Addison-Wesley, Reading, MA, 398 pp.
18. Zadeh, L. A., 1973, "Outline of a New Approach to the Analysis of Complex Systems and Decision Processes," IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-3, No. 1, pp. 28-44.

THE APPLICATION OF INTELLIGENT PROCESS CONTROL TO SPACE BASED SYSTEMS

Author:
G. Steve Wakefield
SRS Technologies
990 Explorer Blvd., NW
Huntsville, Alabama 35806
(205) 895-7000

Abstract

The application of Artificial Intelligence to electronic and process control can help attain the autonomy and safety requirements of manned space systems. An overview of documented applications within various industries is presented. The paper then presents a discussion of the development process and associated issues for implementing an intelligence process control system.

1.0 Introduction

Intelligent process control can be defined as the application of various artificial intelligence techniques to the control of chemical and electrical processes. Intelligent process control can provide input data validation, fault detection, diagnostics, and fault recovery implementation. Space systems to which this technology may apply include Space Station environment control systems, on-board process experiments, cooling systems for various space based systems, and future space based fuel production systems. This paper presents an overview of intelligent process control technology, and how it can be applied to various space systems.

The development of intelligent process control systems can help meet the autonomy and safety requirements of manned, space systems. For example, space station autonomy requirements specify that (6):

1. Systems be "capable of detecting and reacting to selected anomalous conditions."
2. Vital system functions must be maintained during abnormal operations.
3. "Platform systems. . . be capable of autonomously placing the platform in a safe condition."
4. Platform systems collect and transmit system information to the ground.

The first three requirements can be addressed by integrated, in-line intelligent control systems.

2.0 Applications Overview

The application of intelligent control technologies to space systems can have significant payoffs in several major areas, including:

1. Improved system reliability due to intelligent monitoring;
2. Rapid fault correction due to diagnostic and repair expert systems; and
3. Increased system efficiency due to intelligent resource management, and automated fault recovery strategies.

This section describes related commercial process control projects and how similar technologies can be applied to space applications.

The first area to which intelligent process control can be applied is intelligent monitoring of space based process systems. Embedded knowledge bases can act in a supervisory role to existing feedback controllers. The knowledge base can monitor input data to detect problems with sensory equipment or with the process itself. An expert system to validate sensor input has been developed by DuPont (8). The expert system monitors analyzer operations and notifies the operator when a fault arises. Other applications of intelligent process monitoring have been described in nuclear (7), chemical (3,9), food processing (5), and circuit board manufacturing (4) industries. Each of these applications includes the validation of sensor input. Once a thorough understanding of the system model is developed, rules concerning "feasible" input data can be developed. Factors which may indicate sensor problems include:

1. Inconsistent readings between related inputs,
2. Unreasonable input data, based on system bounds (i.e., negative fluid levels, etc.),
3. Sudden step changes in signal value, and
4. Unexpected "noise" levels in sensor values.

Some of these factors may also indicate problems with the actual process. The knowledge base must be designed to distinguish between sensor and process problems. Other factors (assuming valid sensor readings) which may indicate process problems include:

1. Unstable trends,
2. Values outside given thresholds, and
3. Performance measures determined from related process variables.

One method for detecting sensor and process problems is Statistical Process Control (SPC). SPC techniques can be integrated into the intelligent process control system's knowledge base. The use of SPC methods can be used to detect process trends before warning thresholds are exceeded, to detect shifts in the overall process, and to identify unrealistically constant input values. The application of SPC techniques to intelligent process control is described by Bailey (1) and by Blickley (2).

Once a problem is detected, the second major function of an intelligent process control system begins: diagnostics. Each of the above mentioned applications included fault diagnostics. Particularly in space applications, rapid diagnosis of system problems is critical. Diagnostic modules of intelligent process control systems are based upon system design knowledge, maintenance and repair information, and an accurate process model. Much of this knowledge is collected during ground testing of the system before it is launched. The diagnostics knowledge base should be thoroughly tested and validated with the ground system to ensure accurate diagnoses of problems once the system is deployed.

Accurate diagnostics are crucial to the implementation of system recovery or repair procedures. These procedures can be incorporated into an intelligent control system at two levels: advisory or integrated. An advisory level system suggests recovery strategies based upon the determined diagnosis. The user can then implement the suggested procedures or implement alternative procedures. An integrated in-line system would automatically begin recovery procedures where possible. Only in extremely well defined systems with time critical recovery requirements should an integrated in-line system be used. Even then, command personnel should be notified of all actions taken by the automated system.

One of the most important steps in developing an integrated intelligent process control system is the testing and validation of the knowledge base. Standard algorithm or model validation techniques are not generally applicable to knowledge-based systems. Although there have been several documented implementations of advisory level intelligent control systems, there has been

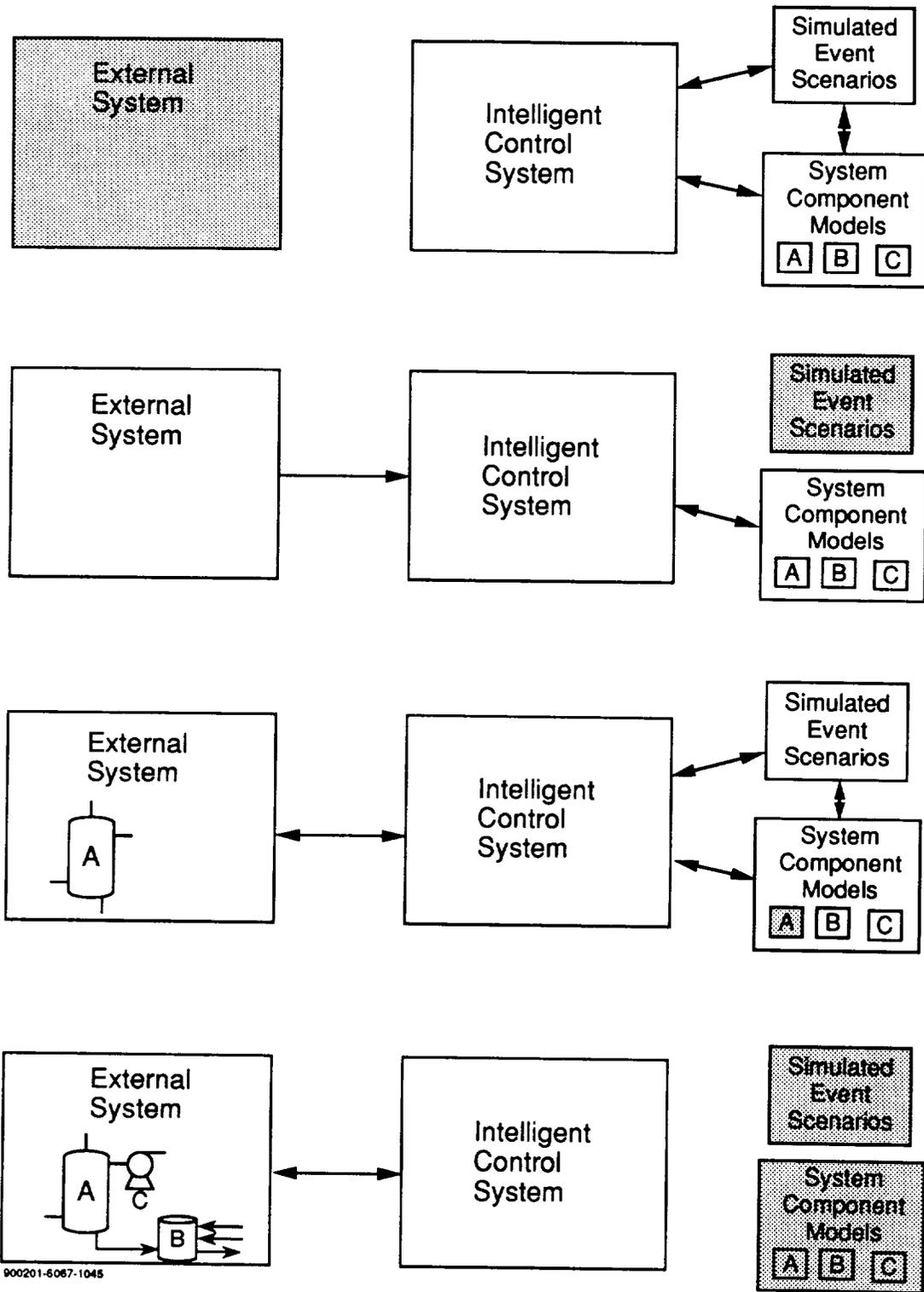


Figure 2. Step-Wise Development and Integration

1. To allow the control system developer to test various system event sequences during knowledge base construction.
2. To test different recovery strategies and their impact on the individual system components as well as the overall system.
3. To test actual in-line system components individually during system integration.

The scenario generation tool is integrated with the knowledge base as well as the system model component in order to allow interactive, dynamic determination of scenario events. For example, if the knowledge base does not diagnose the correct fault given a certain series of events, the scenario generation would stop to allow the developer to correct the problem. Otherwise, the scenario would continue. Probabilistic branching in the scenario path can be used to introduce random events into the test sequences.

Once the knowledge base has been fully validated, the process of integration must begin. This, too, can consume considerable resources. First, the hardware and/or software links must be developed for accessing the real-time data. This often involves integrating specific data acquisition routines with the inference structure. SRS has utilized the open architecture of the Nexpert OBJECT (from Neuron Data Corporation) expert system development shell to integrate data acquisition and direct memory access hardware with the intelligent control knowledge base. The ability to incorporate user defined external routines is also important for a commonly overlooked link in the overall system: the data fusion link. Often, the data obtained directly from sensors is not in a form amenable to the knowledge base. In these cases, external filters, statistical analysis, or other data fusion algorithms are used to abstract higher level information from the raw data.

The output links are even more critical to an integrated, in-line system. These routines must translate a knowledge base output to control hardware inputs. Any errors in the output routine definitions can jeopardize the integrity of the system. The simulated system models can also be used during this final stage of development since actual components can be integrated and tested individually to ensure that heuristically derived control outputs produce the desired system response.

4.0 Conclusions

This paper has presented an overview of some general issues involved in developing control systems utilizing embedded knowledge base processing. Although significant research has been conducted in the application of heuristic processing to system monitoring and fault diagnostics, further research is required in the areas of knowledge base testing methodologies and integrated, in-line systems. The use of scenario generation and system modeling tools for knowledge base testing was presented as a currently used approach. Also, the issues of integrating the knowledge base with the external systems were presented.

References

1. Bailey, S.J., "Autonomous Decision Sources for Today's Automated Plants," Control Engineering, July 1988, pp. 75-78.
2. Blickley, George J., "SPL May Really Become Process Control," Control Engineering, July 1988, pp. 83-84.
3. Calandranis, J., et.al., "DiAD-Kit/BOILER: On-Line Performance Monitoring and Diagnosis," Chemical Engineering Progress, Vol 86, No. 1, January 1990, pp. 60-68.
4. Chapman, Bruce and Rick Yeager, "Artificial Intelligence Applications, in PWIB Manufacturing," TI Engineering Journal, January-February 1986, pp. 34-46.
5. Herrod, Richard and Michael Smith, "The Campbell Soup Story: An Application of AI Technology in the Food Industry," TI Engineering Journal, January-February 1986, pp. 16-19.
6. NASA, Space Station Program Definitions and Requirement Document (PDRD), SSP 30000, Sec. 3, Rev. H, p. 3-20.
7. Nelson, William R. and James P. Jenkins, "Expert System for Operator Problem Solving in Process Control," Chemical Engineering Progress, December 1985, pp. 25-29.
8. Rowan, Duncan A., "On-line Expert Systems in Process Industries," AI-Expert, August 1989, pp. 30-38.
9. Tayler, Clive, "Moving Toward More Expert Process Management," Process Engineering, April 1988, pp. 63-66.

ARTIFICIAL INTELLIGENCE IN THE MATERIALS PROCESSING LABORATORY

Gary L. Workman and William F. Kaukler
University of Alabama in Huntsville
Huntsville, AL 35899

INTRODUCTION

Materials science and engineering provides a vast arena for applications of artificial intelligence. Advanced materials research is an area in which challenging requirements confront the researcher, from the drawing board through production and into service. Advanced techniques results in the development of new materials for specialized applications. Hand-in-hand with these new materials are also requirements for state-of-the-art inspection methods to determine the integrity or fitness for service of structures fabricated from these materials.

The many facets of materials science and engineering form a complex mixture of interests in compositional variances, processing parameters and service environments. All need to be considered in designing and implementing an advanced material for a particular function. It is anticipated that artificial intelligence can provide many benefits in sorting out some of the complex relationships which help to produce ideal materials for extreme conditions. Sorting out compositions from process parameters is complicated and normally requires expertise from many sources. These sources can often take the form of humans who have vast experience in a specific area and thereby are domain experts in a very focussed area. This knowledge may be accessible from either the persons themselves or from the published literature. On the other hand, with the rapidly evolving materials technology in today's society, one often must include current results from actual or simulated experiments to obtain the knowledge required to attain specific goals.

Selection of the expert system implementation or platform most suited to meeting required goals also presents interesting challenges in the materials world, because the knowledge or awareness of artificial intelligence is quite new and only within the last few years has any real effort been made to incorporate new AI technologies into the realm of new materials technologies.^{1,4}

Two problems of current interest to the Materials Processing Laboratory at UAH are an expert system to assist in eddy current inspection of graphite epoxy components for aerospace and an expert system to assist in the design of superalloys for high temperature applications. Each project requires a different approach to reach the defined goals. The first project has been in existence several months now, while the second will be undertaken during Spring of 1990. Hence this paper will describe results to date for the eddy current analysis, but only the original concepts and approaches considered will be given for the expert system to design superalloys.

EDDY CURRENT INSPECTION OF GRAPHITE EPOXY FILAMENT COMPONENTS

Composite materials have many beneficial characteristics which enhance their role in today's aerospace systems. Lightweight, but still very strong, even at high temperatures, graphite epoxy filaments have been fabricated to replace structures traditionally constructed of metals.

Most activity has occurred in the airframe and missile industries, with some interest currently being developed in Space structures. A major component of the Space Station is the truss structure, which is still being proposed to be made of graphite epoxy composites. Several problems still exist in selecting graphite epoxy for an aerospace structure. No accurate structural models based on traditional fracture mechanics can predict reliabilities for these structural members in the presence of flaws. Determining flaws also presents a major difficulty. Flaws can arise from fabrication errors, impact damage during assembly, and obviously in Space, impact damage and erosion from atomic oxygen. A major need exists in being able to measure the integrity of the graphite-epoxy structures during fabrication and in Space. Many research groups are involved in developing NDE techniques to detect flaws in graphite epoxy structures. Primary interest is to not only to determine that there is a flaw, but more importantly, the size and nature of the flaw. This project seeks to develop the capability to utilize simple heuristics types of logic to determine the flaws in real time, as compared to the more time consuming methods based upon statistical and computationally intensive methods.⁵⁻⁶

Eddy current inspection is a nondestructive testing technique which utilizes a high frequency electromagnetic field to induce eddy currents in the material under test. The magnitude of the eddy currents induced in the materials provides a response of the material which should indicate a measure of the integrity of the material. Since graphite fibers have a measurable conductivity, eddy current methods can be applied for inspection purposes. Unfortunately, due to the very low conductivity, the signals are weak. Several methods have been developed to increase the signal-to-noise ratio, including signal processing schemes and design of special eddy current transducers. A horseshoe or E-probe design has advantages in that it can concentrate more magnetic flux within the fiber, in addition to providing directional selectivity. Finite element models of the two types are shown in Figure 1, where a.) is the normal pancake probe and b.) is a horseshoe or E-probe.⁷⁻⁹

A number of defects exist for composite materials, as shown in Table 1. The goal of the expert system is to assist in the determination of the nature, size and location of the flaws using knowledge which relates changes in resistance and reactance of the component being tested to identify a flaw type and depth. Knowledge is incorporated into the expert system through heuristics gained by measurements on known flaws and through finite element calculations on simulated defects. The location and size of the flaw will be determined to a large extent by the scanning parameters.

The eddy current inspection facility is a robot-scanned facility using an Indellidex 550 Robot. The scanning manipulator has 5 degrees of freedom and uses a programmable computer controller for trajectory and task programming. At this time no off-line programming tools exist. The system supervisor and the platform for the expert system is a MacIntosh II using a MacIvory engine. The robotic cell layout is shown in Figure 2. A major part of the problem, in addition to the expert system development, is the integration of the components making up the cell into a flexible and productive inspection facility. Developing off-line programming tools for the facility will become a major goal in later work.

The overall expert system architecture is shown in Figure 3. As mentioned earlier, the knowledge base consists of two types, computed and measured. This type becomes the principal

form of knowledge in domains where few experts reside, but the availability of knowledge applicable to a specific domain required for real-time processing of signals will most often take the form of a measured quantity.

The MacIntosh II platform was chosen for its enhanced user interface and graphical environment. The MacIvory board set represents a state-of-the-art symbolics processor residing inside the MacIntosh computer. The two environments are accessible through program calls, but do not run on top of one another. The rationale for choosing this platform for the expert system was to establish the capability of the system to support the various tasks required in taking eddy current data, processing computationally and symbolically to determine flaw sizes and locations, and then to generate three-dimensional graphic representations of the structure and the flaws. The over-all task description is very robust and represents a good test of the capabilities of the MacIvory environment. In any case, the computer controller for the robot can perform as the robot cell controller, allowing for the MacIntosh to function solely as a data acquisition and analysis module. Both cases will be tested.

Expert System for the Design of Superalloys

This section presents some of the concepts which evolved in developing a proposal to create such an expert system. The primary rationale behind the choice of an expert systems to be applied to the problem is that there is a lot of experimental data existing in the literature which presents a very complicated, and often ambiguous description of the essential elements or processing required to prepare a superalloy with specific attributes. For example, in the proposed study, in order to basically weed out what are the most important parameters for stability to hydrogen environment, the literature repeatedly contradicts itself, i.e. there is a conflict between the experts.

Figure 4 shows the original concept for an expert system to assist in the design of superalloy systems. To implement such a robust system, will obviously take several man-years of effort. However, by developing the overall concept in modules which can provide a useful benefits for functions such as stability in hydrogen environments, the capabilities are appreciated. Rapid prototyping only one of the modules at a time will allow the system to evolve in conjunction with the ability to generate experimental data also.

Expertise for designing superalloys exists in a few researchers and within organizations desiring proprietary handling of data. Consequently the knowledge required for this effort is not only scattered, but also, in some cases, of such proprietary nature that one company will not share their knowledge with another organization. Computations on phase formations and kinetics of metallurgical reactions for particular compositions also are required for microstructural predictions. Hence the problem solving approach consists of heuristics, data from established databases, and computations as required. Choosing an expert system shell to prototype the system was difficult.

Several expert systems shells were considered for this project. Among all the choices available, Nexpert Object was chosen because of its capabilities. Both forward and backward chaining are allowed, program calls to computational or other types of procedures are allowed

and a very easy to use human interface allows for simple rule construction and inferencing. Dynamic inferencing through its non-monotonic reasoning capability are essential in order to deal with the complexities of superalloys.

The project will begin in January, 1990 with the goal to develop a prototype system to design a superalloy system with optimal stability against hydrogen embrittlement within six months. The results will be interesting. Successful completion of the preliminary goals will obviously provide a foundation upon which to add other modules in order to build up the capabilities of the system. Also we anticipate that other expert systems for advanced materials design will be attempted in the near future.

ACKNOWLEDGEMENTS

This work was performed under contract from Marshall Space Flight Center on NAS8-36955. The eddy current inspection activity has been performed by graduate students, Mr. Morgan Wang and Mr. Sung Lee at UAH and Craig Bryson in the NDE Laboratory at Marshall Space Flight Center.

Bibliography

1. D.E. Marinaro and J.W. Morris, Jr., "Research Towards an Expert System for Materials Design", Artificial Intelligence Applications in Materials Science, ASM, Metal Park, OH, 1989, pp 49-78
2. I. Hulthage, M. Przystupa, M.L. Farinacci, and M.D. Rychener, "The Metallurgical Database of ALADIN - An Alloy Design System", Artificial Intelligence Applications in Materials Science, ASM, Metal Park, OH, 1989, pp105-122
3. W.A. Boag Jr, D.B. Reiser, D.O.Sprows, and M.D. Rychener, "Cordial - A Knowledge-Based System for th Diagnosis of Stress Corroison Behavior in High Strength Alloys", Artificial Intelligence Applications in Materials Science, ASM, Metal Park, OH, 1989, pp123-145
4. K.C. Maddux and S.K. Jain, "Manufacturing Process Simulation: A Building Block in an Artificial Intelligence System for the Manufacturing Engineer", Artificial Intelligence Applications in Materials Science, ASM, Metal Park, OH, 1989, pp13 - 34.
5. P.G.Doctor, T.P. Harrington, T.J. Davis, C.J. Morris, and D.W. Fraley, " Pattern-Recognition Methods for Classifying and sizing flaws using eddy Current Data", pp464-483
6. L. Upda and W. Lord, "An AI Approach to the Eddy Current Defect Characaterization Problem", QNDE 6, 1986, pp 900 - 906.
7. S.N. Vernon, "Probe Properties Affecting the Edy Current NDI of Grpahite Epoxy", QNDE, vol.5, pp 1113 - 1123.
8. S.N. Vernon and P.M. Gammell,"Eddy Current Inspection of Broken Fiber Flaws in Non-

metallic Fiber Composites", QNDE 4, 1984, pp. 1229 - 1237.

9. M.J. Donachie, Jr., "Introduction to Superalloys", Superalloys Source Book, published by ASM, Metals Park, OH, 1984, pp 3-15.

TABLE 1
DEFECTS IN COMPOSITE MATERIALS

FABRICATION DEFECTS

VOID CONTENT
STATE OF RESIN CURE
FIBER/MATRIX INTERFACE
FOREIGN INCLUSIONS
TRANSLAMINAR CRACKS
DELAMINATIONS
FIBER ALIGNMENT

SERVICE DEFECTS

MOISTURE INGRESS
ULTRAVIOLET DEGRADATION
TEMPERATURE EXTREMES
MATRIX CRACKING
DELAMINATION
FIBER/MATRIX DEBONDING
FIBER BREAKAGE
IMPACT DAMAGE

FIGURE 1. Finite element models for a.) pancake probe and b.) E-probe for 5 megahertz electromagnetic coupling to graphite fiber. The coil is wrapped around the center bobbin in both cases. Note that the pancake probe produces a field 360° around the centerline, while the E-probe produces a flux across the direction of the fiber.

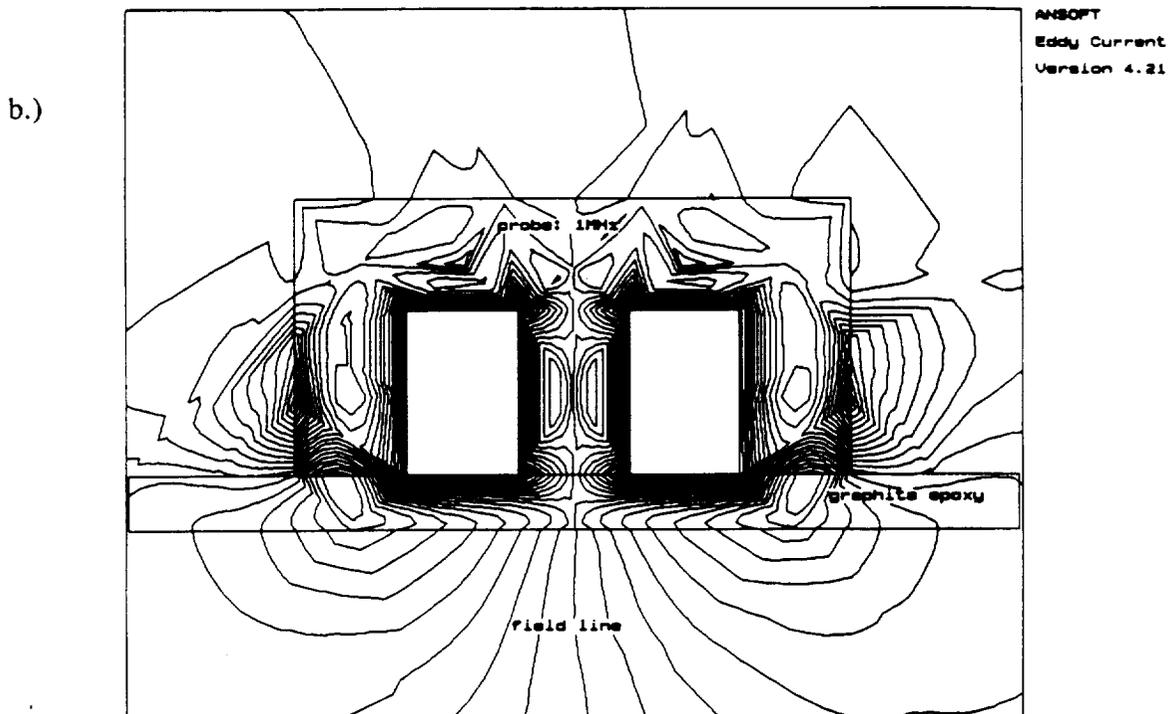
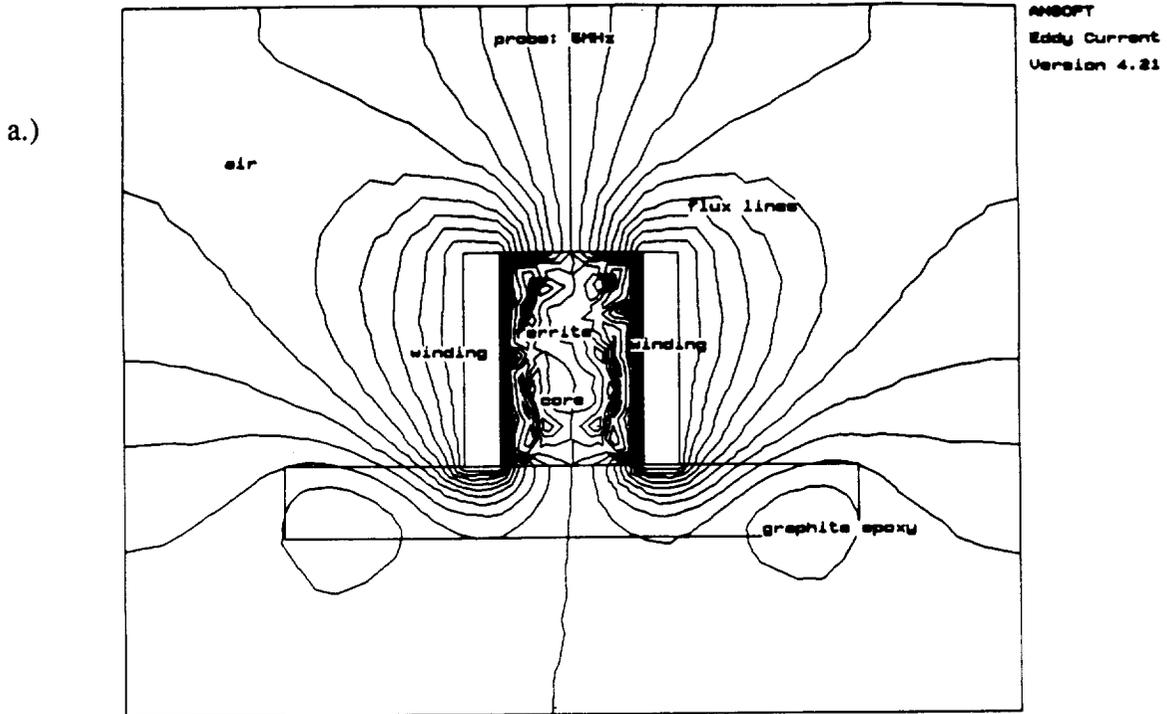


FIGURE 2. EXPERT SYSTEM FOR AUTOMATED
EDDY CURRENT ANALYSIS

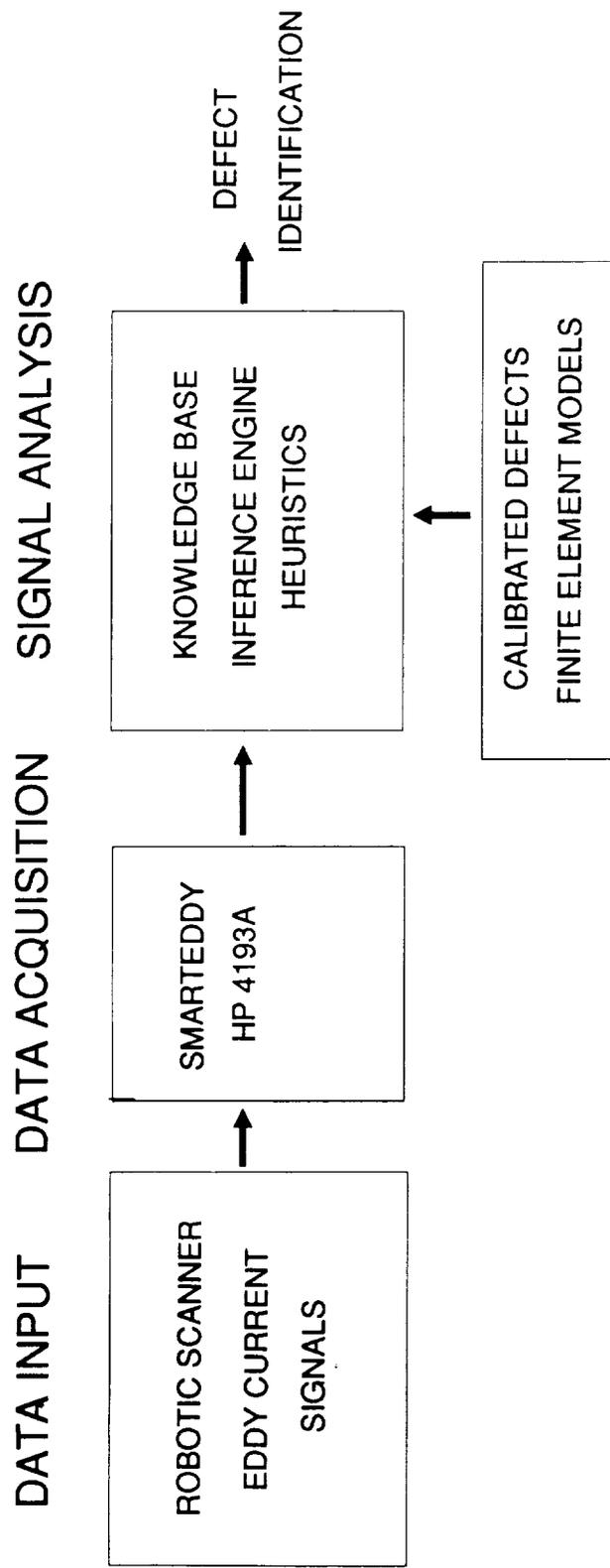


FIGURE 3. EXPERT SYSTEM ARCHITECTURE FOR EDDY CURRENT ANALYSIS

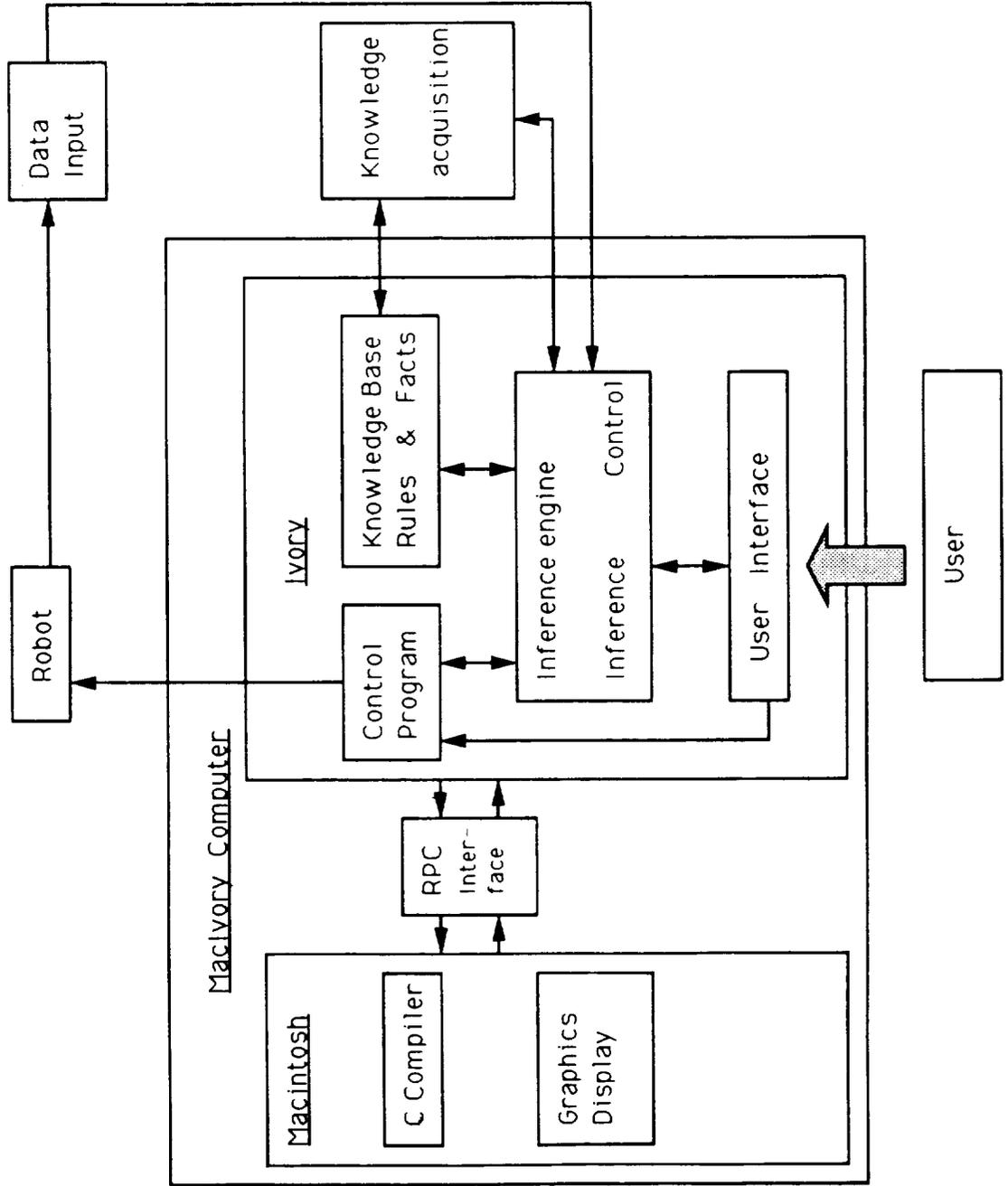
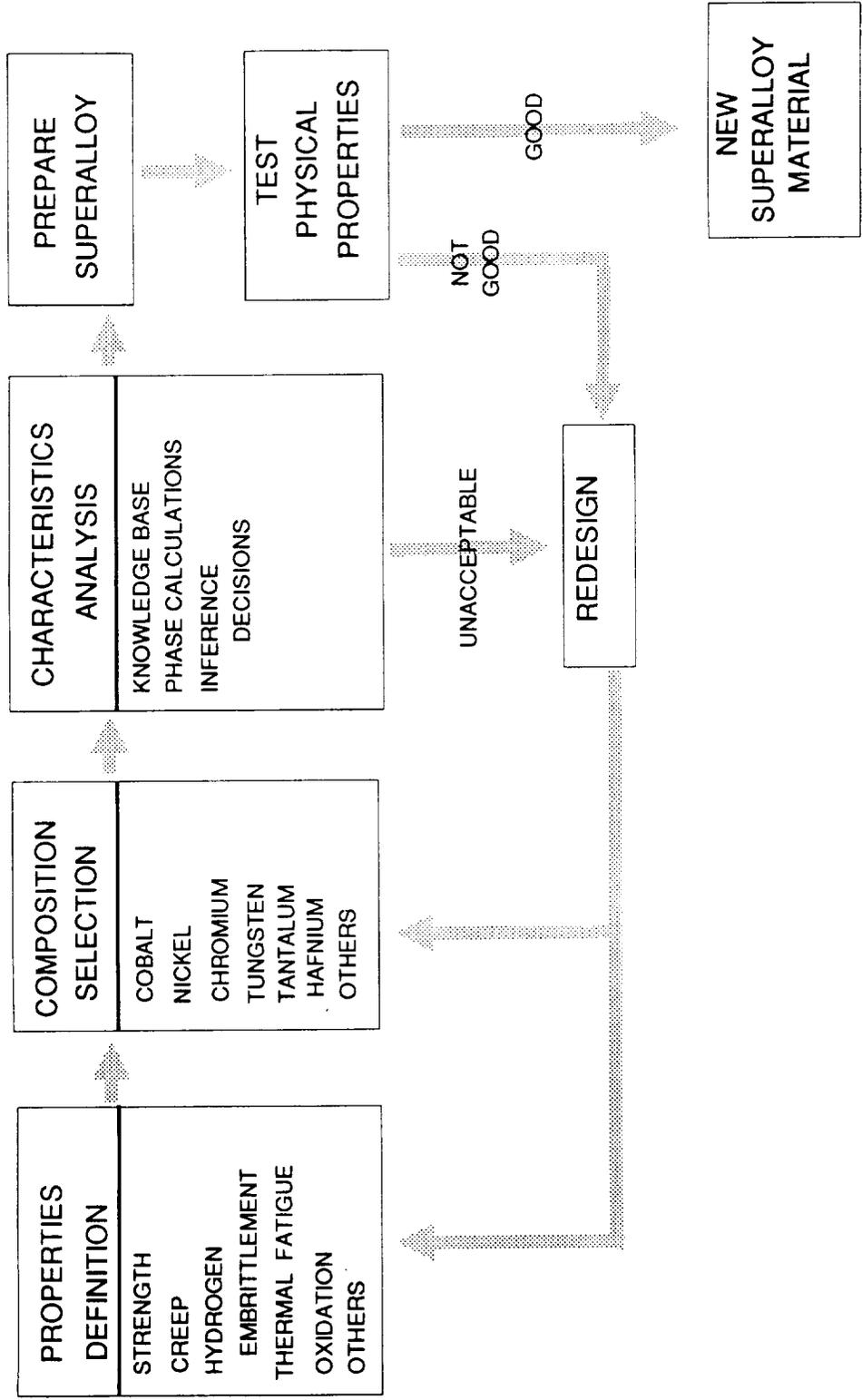


FIGURE 4 SUPERALLOY INTERACTIVE DESIGN APPROACH





An Architecture for Intelligent Task Interruption

D.D. Sharma and Srinu Narayan
Corporate Technology Center
FMC Corporation
1205 Coleman Ave
Santa Clara, CA 95052

Abstract

In the design of real-time systems the capability for task interruption is often considered essential. In this paper we examine the problem of task interruption in knowledge-based domains. We propose that task interruption can be often avoided by using appropriate functional architectures and knowledge engineering principles. Situations for which task interruption is indispensable we describe a preliminary architecture based on priority hierarchies.

1 Motivation

Real-time systems require that decisions be made dynamically in response to incoming sensor data with desired responsiveness within the specified time deadlines. Often real-time solutions should degrade gracefully. Knowledge-based real-time systems perform several functions such as sensing information, processing information to determine the situation status, evaluating the situation, plan for needed actions, and executing the planned actions while monitoring the status of the plans, the system, and the situation. In such systems time stress arises from sources external (in terms of high data rate, unanticipated demands on problem solving, and unexpected deadlines) and internal (knowledge based processing is computationally intensive and the computation time is often unpredictable) to the knowledge based system. All practical systems have limited resources and given the unexpected demands requires intelligent resource management schemes.

Resource management involves making resources available to tasks that need them and reclaiming resources from tasks that do not need them. To accomplish resource management, amongst other things, the capability to interrupt tasks, suspend execution, and later resume execution may be needed. Given the tasks to be interrupted, real-time operating systems provide efficient low level mechanisms to implement task interruption. In knowledge-based systems the determination of the task to be

interrupted has to be inferred in runtime and is based on the specific context. Runtime reasoning to determine which task to interrupt and is computationally expensive and alternative solutions are desired.

2 Issues of Task Interruption in Real-Time Knowledge-Intensive Architectures

An hospital is something all of us can relate to, therefore, following [Hayes-Roth 1987] we use a hospital situation as an example to describe issues of task interruption. Hospitals are well designed functional structures, they have finite resources (heterogeneous), perform various tasks, and respond to time critical situations. The function of a hospital is to provide health maintenance services to the community it serves while controlling the cost of providing such services. A hospital situation has the following characteristics relevant to task interruption.

1. Finite Resources: Resources of an hospital are its physicians, surgeons, nurses, pharmacy, operating theaters, operating budget, etc. The resources in each category are finite and often cannot be interchanged.
2. Environment Demands: The hospital receives a continuous stream of patients. Different patients have different treatment requirements. During certain seasons the demand for one type of treatment may increase significantly. During certain calamities the demand on certain facilities may suddenly increase. The hospital are required to adapt to such external changes.
3. Internal Demands: Apart from the constraints of finite resources the internal systems are subject to failure (e.g., strike by nurses). Thus the available internal resources are dynamically changing.
4. Emergency Situation: Not all patients are attended to within uniform time interval. Some patients make appointments. However, there is often a stream of critical patients who need immediate care. The way a hospital solves this problem is by providing an architectural solution; i.e., an emergency ward. An emergency ward is staffed and operated under a different set of resource allocation policies.
5. Categorized External Demands: Given the inflow of patients to a hospital a single receptionist desk will quickly overflow. The problem is solved by recognizing that incoming patient have different needs. Often patients themselves know the general nature of problem. Again a hospital provides an architectural solution: it has several speciality wards and screening nurses. An incoming patient can directly go to a speciality ward or go to the screening nurse from where he is directed to an appropriate speciality ward.
6. Resource Contention: At any given point in time there are only a limited number of doctors, nurses, operating theaters, and anesthesiologists available. Since hospitals attempt to minimize operating costs the resources are scheduled based on an expected resource demand profile. Whenever external demands exceed the average demand the available resources fall short of the need. This leads to contention of resources. Also internal failures can cause less than allocated resources to be available thus causing resource contention even during a normal demand situation.
7. Resource Facilitators: Certain resources such as Nurses are like resource

facilitators - they make sure that resources needed to attend to a patient are available to a doctor when needed. Also in situations of resource contention nurses or interns can relieve a surgeon for a limited period of time. Thus having resource facilitators helps in achieving graceful interruption.

8. Anticipatory Scheduling of Resources: Apart from providing architectures for efficiently applying resources, hospitals deliberately schedule resources to reduce the effects of unanticipated upsets in resource demands. Thus instead of being reactive to resource demands, the demands and the readiness of resources are anticipated based on past models and activities are scheduled accordingly. For example, major surgeries are often scheduled in morning hours. Blood transfusion and anesthesiology is not scheduled during weekends [Hester, 1989].
9. Taxonomy of Tasks from Interruption viewpoint: A patient (i.e. a task) needs a wide range of heterogeneous services. In a hospital situation these services are provided by various specialists. To look into the problem of interruptibility instead of thinking in terms of surgeons (i.e. resources) being interruptible or non-interruptible. Typically, most of the activities of a physician might be interruptible. In other cases the activity need not be interrupted while in other cases the activity cannot be interrupted.
10. Context Dependency: The above example gives a default classification of activities. In reality the interruptibility determination is made with respect to the context. For example, contrary to expectations a surgeon's activity is often interruptible. In fact the interruption is facilitated by the presence of numerous interns and highly skilled nurses. Depending upon the stage of the surgery a surgeon can be pulled out to assist on a relatively more complex and urgent case.
11. Task Interactions: A certain patient is undergoing a course of treatment. The patient develops a new abnormality requiring administration of new drugs. Do we stop the ongoing treatment completely? Can the new drugs unfavorably interact with the new treatment? Such issues need careful examination.
12. Task Interdependency: A patient is undergoing an intricate surgical procedure. Can the anesthesiologist be pulled away for another case? This will depend upon the stage of the surgery and the future needs of this procedure.

In summary:

1. Responsiveness and resource contention problems do not necessarily require interruption. Architectural and appropriate resource scheduling solutions are preferred over task interruption.
2. In situations of emergency (time critical situations) where the demand considerably exceeds available resources task interruption may be needed.
3. Determining interruptibility of a task is highly context dependent.

3 Intelligent Task Interruption

The Task Interruption problem can be defined as follows: Given n concurrently active tasks T1, T2, ..., Tn (where each task is characterized by its resource needs and completion deadlines) and a new task T waiting for execution the following question need to be answered: Should we interrupt any of the existing tasks and which one?

Task Interruption in realistic situations requires use of situation and domain specific knowledge to determine whether and who to interrupt. To perform efficient interruption we need two things:

1. A Problem Solving model to enable deciding whether, who, and where to interrupt. The model should take into account logical and temporal dependencies between the on going activities.
2. An architecture to support efficient interruption thus addressing how to interrupt.

In this paper the architecture support will be not be discussed in detail. Various architectural designs will be briefly mentioned where appropriate.

4 Why Interrupt? or The Goals for Interruption

The problem of deciding whether to interrupt a task should be put in context of the objectives of the situation. Some of the objectives and possible solutions are discussed below.

1. **Responsiveness.** In real-time systems improving responsiveness for a certain a category of tasks is important. In conventional software systems responsiveness is achieved by sending an interrupt signal to the appropriate hardware. In AI architectures responsiveness can be improved without interruption. Following designs lead to improved responsiveness;

- **Task Grain Size Control:** In agenda based architectures (Figure 1a) tasks are posted on a agenda and scheduled for execution. A task is executed only after the previous task was completed. The waiting time of a task on the agenda is equal to the product of number-of-tasks-ahead in the task queus and the average-task-execution-time. The agenda gets a chance to deallocate a task every average-task-execution-time seconds. Thus responsiveness can be increased by decreasing the task granularity. Large tasks can be decomposed and expressed as a sequence of such micro-tasks. For example, hospitals have time-based agendas for nurses. By appropriately decomposing the size of the tasks they can do (such as administor medicine, take temperature) their responsiveness is increased.
- **Priority Channel Architectures.** In priority channes architectures different channles are provided for events of different priority (Dodhiawala et al., 1989). A channel defines computation through all the stages of problem solving (very much like a thread) (Figure 1b). Tasks on a higher priority channel are given precedence over tasks on lower priority channel. Thus the responsiveness for high priority tasks is much better. For example, an

accident victim will probably be put on an emergency channel.

- **Multiple Priority Task Queues:** Instead of having a single task queue one can define multiple task queues based on the priority of tasks. In such a scheme a cpu time slice will always be first given to the highest priority task queue. In multiprocessor architectures instead of a cpu time slice we will allocate adequate number of processors to appropriate task queue. The allocation of processors can be either static or dynamic. In dynamic allocation the responsiveness requirement can be changed to meet the external demand and processors be allocated appropriately. The QP-Net architecture provides the dynamic allocation feature [Sharma and Sridharan, 1988]. As an example certain wards in a hospital, such as intensive care, get a high priority.

2. Redundant Tasks. AI tasks often have unpredictable execution times. In some designs multiple solution tracks are simultaneously investigated and the best solutions obtained within the deadline is accepted. For example, in some emergencies both the hospital and the local fire department can get a call. This causes the problem of deleting no more desired solution tracks. One way is to keep track of all of the redundant tasks and then delete them later. Another possibility is to make these tasks timed tasks with an allocated time budget. If the tasks do not complete within the time budget then they self terminate.

3. Lack of Resources. In some situations it is possible that resources necessary for a task may no longer be available (Dynamic reallocation of resources will cause this situation). Thus it will be desirable to remove the task from the list of active tasks. This problem is of importance in control systems tasks where certain actions may continue ignorant of the status of resources. A solution is to check for the resources periodically and self suspend/terminate when such resources will no longer be available.

4. Remove Undesired/Harmful Tasks. In some contexts certain tasks if allowed to continue will cause unacceptable consequences. In such cases these tasks have to be identified and interrupted. Task interruption will also be needed in case of conflicts between the tasks.

5. Free-Up Resources. If the resources needed for a task cannot be made available by rescheduling and the tasks cannot wait then certain number of tasks will have to be interrupted.

In summary:

1. For most of the traditional reasons for interruption it is possible to avoid task interruption by utilizing appropriate architectural and task model designs. Given the cost of interruption such alternatives are preferable.
2. Situations where task interruption is needed are: task conflicts (logical), removal of potentially harmful tasks, and for freeing-up resources for more critical tasks.

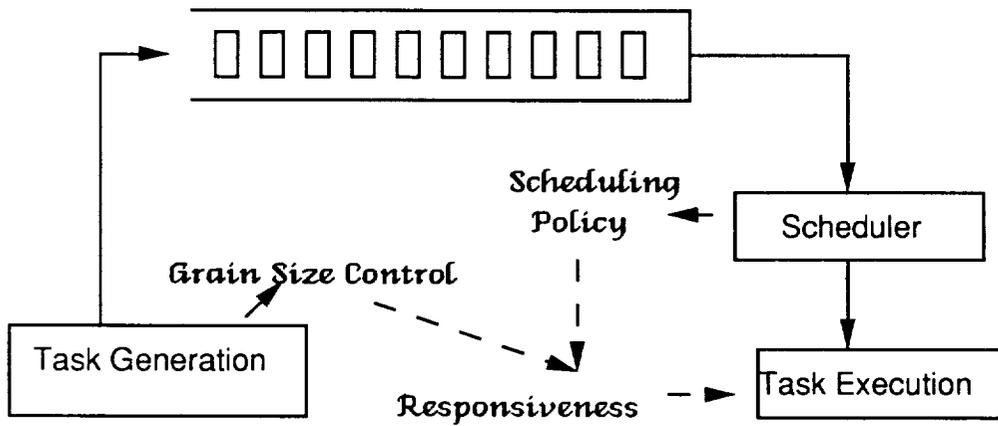


Figure 1 a. Agenda Control Architecture (RT-1)

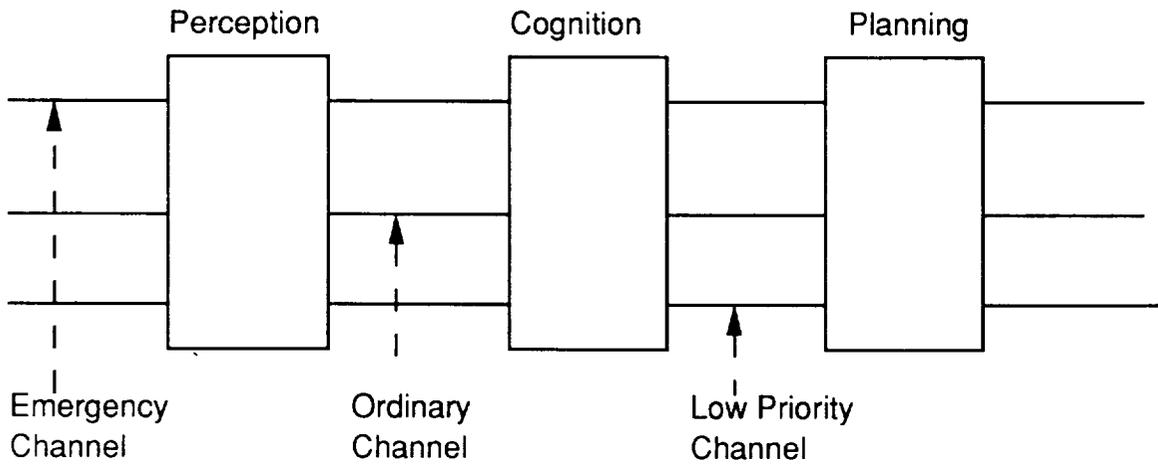


Figure 1 b. Priority Channel Based Architecture

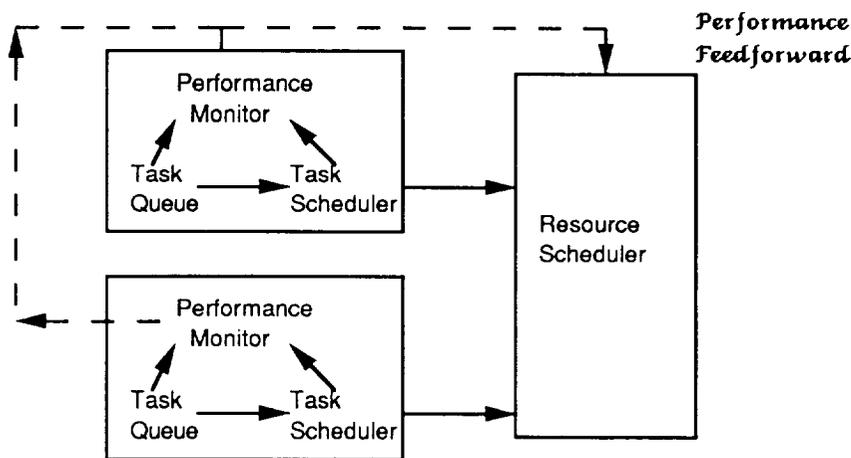


Figure 1c. Performance Controlling Architecture (QP-Net)

5 Who to Interrupt?

The question of who to interrupt needs to be answered in basically two cases: (1) There is a logical conflict between tasks, and (2) There is a need to free-up resources for more critical tasks. To solve this problem we will have to develop an approach for selecting one set of tasks over other. Such a selection needs to be based on the significance of the tasks to the overall goals of the system. We thus need to develop certain approaches for expressing and reasoning about the significance.

Task Significance Principle: The information on task significance can be used as follows: If there are n active tasks (T_1, T_2, \dots, T_n) and a new task T is awaiting execution then interrupt enough number of tasks of significance less than T so as to release sufficient resources for T .

5.1 Representation Issues of Task Significance

The significance measure of a task should express the following types of knowledge about the tasks:

1. Certain domain specific preferences or priorities. For example, in most domains the safety issue will have a precedence over productivity issues.
2. Sensitivity to real-time factors such as task deadlines. Thus even though two instances of same task (defend against an enemy missile) may have same preference but the one with a shorter deadline takes precedence over the other.
3. Sensitivity to Context. Most real-time systems operate in different modes. For example a certain fighter plane can be either in attack mode or egress mode. Depending upon the mode the tasks may have different preferences.
4. Task Interactions. In complex systems the effect of execution or termination of a task can propagate various side effects. In real-time systems the unpredictable dynamics of the environment makes it infeasible to anticipate and compile all possible interactions. The exact nature and the extent of the impact of these side effects has to be determined during run-time.
5. Task Dependencies. The basic principle here is that if a significant task depends upon the result of a specific task then that task is significant too. For example, if we interrupt the supporting tasks then the context and resources for several other tasks may be lost and they may have to be interrupted as well. In this dependency analysis we need to examine the criticality with respect to the new task T as well. Moreover, the task dependencies can be time dependent. For example, given three tasks $T_i, T_j,$ and T_k task T_j may require that task T_i be executed by some time t_1 , and task T_k may require that T_i be executed within the window (t_2, t_3) .

In summary:

1. Significance of a task depends upon the runtime environment of a system.

Determining task significance requires extensive runtime analysis. The problem being that the computational cost of such analysis may be prohibitive to real-time performance".

2. Even if the computation of significance can be done in real-time the determination of the impact of task interaction and dependencies will be imprecise. This is due to the unpredictable and dynamic nature of the real-time environments.

5.2 Reasoning Issues Related to Task Significance

Given significance measures of various tasks one can then use the Task Significance Principle as a reasoning guideline. In dynamic systems where the significance of tasks can change rapidly the recomputation of the significance of all the tasks can be prohibitive. To appreciate the implication of the last point let us consider two measures of significance: a cost/benefit measure and a utility measure.

Cost/Benefit Measure. Assume a new task T on the waiting queue and N tasks on the active queue. Should we interrupt an active task in favor of new task T? In a cost/Benefit decision making framework it will be argued that this question should be answered based on the following considerations:

- (1) What is the benefit of executing T?
- (2) What is the penalty of not executing T?
- (3) If T will be swapped with an active task T_i then what is the penalty of terminating T_i .

T should replace a T_i if there exists a T_i such that:

$$(1) - (3) > (2)$$

This requires computing at least N such relationships. The bigger problem arises in computing each of the terms (1), (2), and (3). Since the benefit and penalty depends upon the interaction between the task and the resources needed for T may be obtained by interrupting more than one active tasks the complexity of the problem becomes quite hairy. Apart from the complexity arising from the combinatorics of the problem there is also complexity due to interactions based on phenomenological knowledge.

Utility Measure

One way to escape the problem of performing runtime analysis is to define some measure of utility of allocating resources to tasks. Thus if the system has say a maximum of N tasks then we can define utility of allocating computational resources to these tasks. At any given moment let us assume there will be M (< N) tasks on the scheduling queue. Let us assume that we can only allocate resource quantity R. Then we will select the subset of tasks from M such that the total resources is less than or equal to R and the utility is maximized. Every time we reschedule resources we will recompute the utility. Apart from the problem of how we define these utility functions we

have the following problems:

1. We will have to evaluate 2^N configurations to find out the maximum utility subset.
2. The utilities typically will not be static but context dependent. Thus we may have to compute utility functions themselves. In a tightly coupled system these utilities may depend on several parameters thus creating a modelling nightmare.

In practical systems another problem arises. Not all tasks are equal and representing them on a uniform scale can lead to anomalous results. For example, suppose we have the following preference rule for a passenger plane:

Ground Landing is preferred to Crash Landing is preferred to Total Crash

Using utility theory we can get the following undersired result:

**Ground landing with probability p1
or Total Crash with probability p2
is preferred to
Crash Landing with probability p3]**

In summary:

1. We need a measure of significance which is easy to model, compute, and change dynamically. The significance measure will have to be sensitive to contexts, task interactions, and dependencies.
2. To efficiently search for preferred tasks we need a computational architecture.

6 TIPS: A Preference Based Architecture for Task Interruption

TIPS is an agenda-based architecture. It consists of a waiting queue (see Figure 2), an active tasks queue, a tasks scheduler to schedule tasks on the waiting queue, an interrupt scheduler, and a resource scheduler. In general, there can be more than one waiting and active tasks queue for each task category. For example, one can have an emergency waiting and active task queue similar to channels design in RT-1 [Dodhiawala et al., 1989]. In such a case the resource scheduler will first always check the emergency queues for work to do. In general, resource scheduler allocates cpu time slices (in multiprocess systems) or processors (in multiprocessor systems). The task scheduler is same as the scheduler in agenda based systems. It prioritizes tasks on the waiting queue. The interrupt scheduler basically solves the problem of determining whether an interrupt is needed and who to interrupt.

Interrupt scheduler will not solve the problem of responsiveness and other aspects of real-time systems for which interruption is really not needed. It is assumed that for responsiveness appropriate functional architectures will be used such as RT-1. Also the computation of deadlines and resource requirements are solved by the problem solving

architecture.

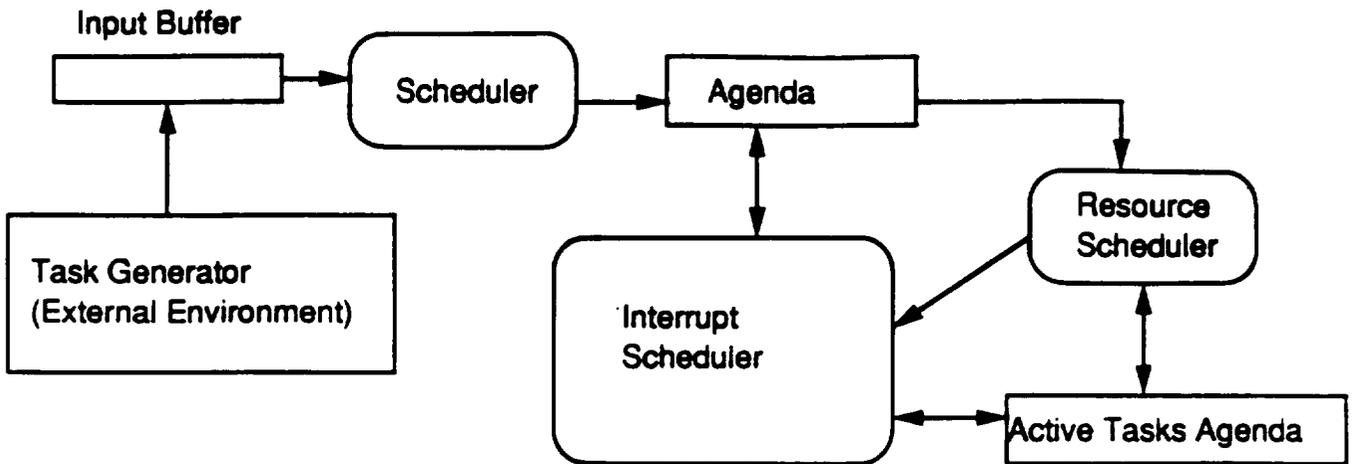


Figure 2. A Preliminary Architecture for TIPS

6.1 Design of Interrupt Scheduler

Interrupt scheduler uses a model of domain contexts, goals, and tasks defined at compile time. The model is hierarchical and consists of at least three layer as shown in Figure 3. Figure 4 shows an example from the Howitzer domain. There are two aspects to the hierarchical model:

1. Preferences of priorities of the tasks are defined with respect to the goals. In general priorities of elements in layer k are defined with respect to the elements in layer (k-1).
2. The hierarchical model is also used as a control structure for determining who to interrupt.

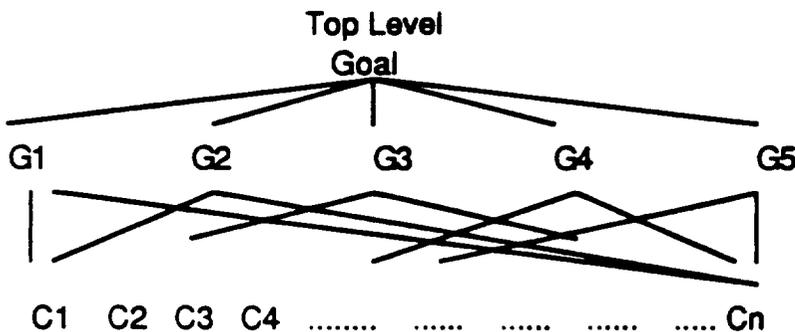


Figure 3. A Preference/Priority Hierarchy

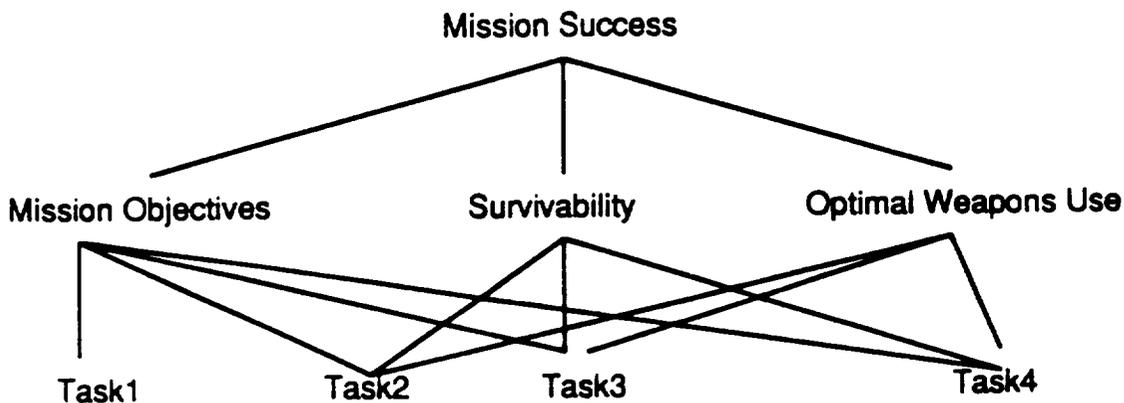


Figure 4. Goals-Task Priority Hierarchy Example

Modeling Task Priorities

The following discussion is based on Saaty's treatment of priorities in hierarchical systems [Saaty 1980]. We will not review Saaty's theory here but simply illustrate it by examples. Some of the basics are as follows:

1. Let C1, C2, C3, ..., Cn be n tasks or goals. Then priority of Ci is defined as a function which maps C -> [0, 1]. If wi is priority of Ci with respect to a high level goal then

$$w_1 + w_2 + w_3 + \dots + w_n = 1$$

2. Let xPy denote that x is preferred over y. xly implies x and y are equally preferable. Then following relations hold:

if xPy then wx > wy
if xly then wx = wy
if xPy, yPz then xPz and also that wx > wz

In case of hierarchical structure shown in Figure 3, the priority of the first task with respect to the top node is given by:

$$WC_1 = WC_1(G_1) \cdot WG_1 + WC_1(G_2) \cdot WG_2 + \dots + WC_1(G_5) \cdot WG_5$$

Where,

WC1 = Priority of C1 with respect to Top Level Goal
WC1(Gi) = Priority of C1 with respect to goal Gi
WG_i = Priority of Goal Gi with respect to Top Level Goal

Task Priorities under Change of Context

If the context changes then the preferences of goals changes with respect to the overall objective. Given new goal level priorities new task level priorities can be

computed.

Example

Consider the aircraft landing example shown in Figure 5. Under the current context the priorities are shown in Figure 5a. In this context it is more important to find an airport with adequate landing facilities. Expedient landing will help passengers feel better but that is less important. In this case the priorities computation lead to the choice of airport D. The task for performing landing at airport A is initiated.

After a while the pilot gets a low fuel level alarm and changes his priorities in favor of expedient landing. He only needs to change the priorities at the goal level. This new prioritization leads to selecting airport D.

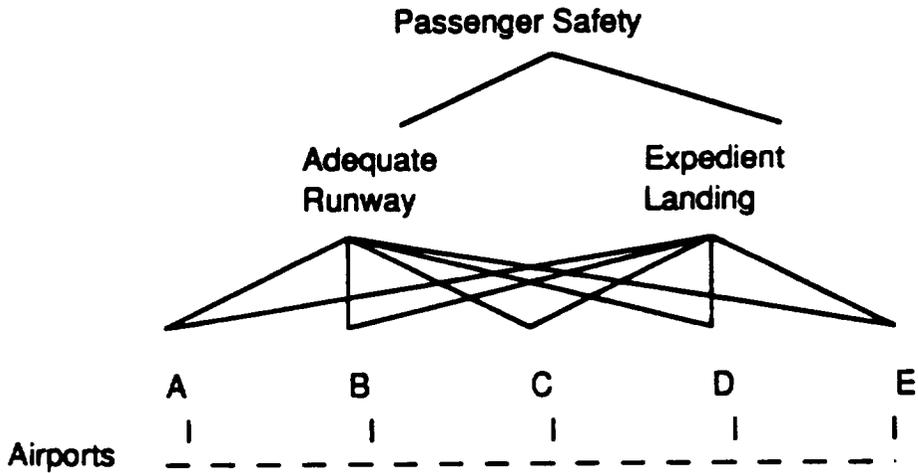
6.2 Task Models

We are essentially considering two types of tasks: tasks that are interruptible and the tasks that are not interruptible. Tasks can be composed of other tasks or micro tasks. Micro tasks once started execute to completion. The size of the micro tasks is such that their execution time is less than the overhead time associated with task interruption.

```
(Task new-task
:preconditions
:body
:sponsor
:goal
:context
:priority
:children-tasks
:Dependents
:Supporters
:Exclude-List
:deadline
:execution-time
:resources-needed
:status)
```

)

```
(Micro-task new-micro-task
:precondition
:body
:sponsor
:goal
:context
:priority
:deadline
:execution-time
:resources-needed
```



Goals Priority Matrix

Goals	Contexts	
	Default	Low Fuel Alarm
Adequate Runway	.9	.2
Expedient Landing	.1	.8

Airports Priority Matrix

Airports	Goals	
	Adequate Runway	Expedient Landing
A	.4	
B	.3	.2
C	.2	.2
D	.1	.4
E		.2

Figure 5. Airport Prioritization for Passenger Safety

:status)

6.3 Control Scheme

The control scheme works on the following principles:

1. Resolve Conflicts Locally First. If a task appears on the waiting list and was created to support goal G_i then compare the priority of the new task with respect to tasks currently active for supporting G_i . If necessary, swap the new tasks with one or more of the active tasks for G_i .

2. Fallback to the next higher level in the hierarchy. If the conflict cannot be resolved locally then it is necessary to resolve conflicts at the higher goal level.

3. Observe Task dependency and interactions constraints to prune the alternatives. The task dependency and interaction constraints are defined as a part of the task models (see slots dependents, supporters, exclude-list in task model).

4. Use the most comprehensive priority value. It is possible for the same tasks to exist on the active tasks list/agenda/schedule/queue multiple times based on the different goals it needs to serve. In that case the total priority of the task should be used.

This is how the scheme will work. In the beginning the system starts with a goal G . In fact the system may start with several goals G_1, G_2, G_3, \dots . For each of the goals priority is defined - the interpretation of preferences being the relative resources to be allocated to pursuing these goals. New tasks are either spawned by the goals or are created in support of the goals. The priorities of these tasks are defined at the time of task creation. These priorities can either be default priorities or may be based on the context at the creation time. We then use four principles defined above to solve the interruption problem.

7 Conclusion

In this paper we have described an analysis of task interruption and a paper design. We believe in the richness of task interruption process. It is our premise that task interruption should be avoided for time critical situation by using appropriate architectural and data structural features. For situations where interruption is essential we have argued in favor of minimizing runtime reasoning. We have described knowledge-based architecture (TIPS) which attempts to explore various features to reduce runtime reasoning. These features are: preference models for tasks, a domain specific task context tree, and a hierarchical control mechanism. Table 1 summarizes major points of this paper.

Acknowledgements

We wish to thank Dr. T. Hester and Dr. N.S. Sridharan for providing critical review of this paper and for many stimulating discussions.

References

1. [Dodhiawala et al., 1989] Dodhiawala, R, Sridharan, N.S., Pickering, C., Raulefs, P., "Real-Time AI Systems: A Definition and an Architecture." Proceedings of IJCAI, 1989.
2. [Hayes-Roth, 1987] Hayes-Roth, B., "A Multi-Processor Interrupt-Driven Architecture for Adaptive Intelligent Systems". Report # KSL 87-31. Department of Computer Science, Stanford University.
3. [Hester, 1989] Hester, T. Personal Communication.
4. [Saaty, 1980] Saaty, T.L., "The Analytic Hierarchy Process: Planning, Priority Setting, Resource Allocation". McGraw-Hill International Book Company, 1980.
5. [Sharma and Sridharan, 1988] Sharma, D.D. and Sridharan, N.S., "Knowledge-based Real-Time Control: A Parallel Processing Perspective". Proceedings of AAI, 1988.
6. [Sharma et al., 1989] Sharma, D.D.; Huang, S.R.; Bhatt, R.; Sridharan, N.S.; "A Tool for Modeling Concurrent Real-Time Computation". Proceedings Workshop on Real-Time AI Problems, IJCAI, 1989.

Table 1. Designs Issues and Solution Options for Interrupt Based Architectures

Design Issue	Design Options
Why Interrupt?	
Responsiveness	No Interrupt, Control of Task Grainsize, Special architectures (RT-1, QP-Net)
Free-up Resources	Reschedule tasks, Interrupt in favor of Critical tasks
Tasks Not Needed	Soft/Hard Interrupt
Who to Interrupt?	
	Preference/Priority Based Selection, Context Tree, Hierarchical Task Selection (TIPS)
Where to Interrupt?	
	Special Task Models Predefined logically safe states (between two micro-tasks) On demand (special cases only)
How to Interrupt?	
No Interrupt	Task Size, Micro-tasks, Dynamic Allocation
Soft Interrupt	Remove from Agenda or Active Tasks Queue
Hard Interrupt	Utilize RTOS support

Knowledge Representation for Commonality*

Dorian P. Yeager, Ph.D.
Department of Computer Science

The University of Alabama
Tuscaloosa, AL 35487

February 5, 1990

Abstract

Domain-specific knowledge necessary for commonality analysis falls into two general classes: commonality constraints and costing information. Notations for encoding such knowledge should be powerful and flexible and should appeal to the domain expert. The notations employed by the CAPS analysis tool are described herein. Examples are given to illustrate the main concepts.

1 Introduction

Commonality is the extent to which a system employs common designs to fulfill similar functions. A system which uses a large number of individually tailored components for specific functions thus possesses little commonality, whereas a system which maximizes the number of applications of each component is said to possess a high degree of commonality. The knowledge which allows an engineer to decide which components are able to serve in multiple applications, and thereby to eliminate unnecessary duplication of functionality, is very domain specific. Commonality knowledge capture is therefore greatly aided by a notation which appeals to the domain expert and which provides a natural way to describe commonality considerations.

With Prolog, we are able to communicate this type of information in a way which is difficult to reproduce with other types of notation. For example, to say that

*Work supported by NASA grants NGT-01-002-099 and NAG8-718.

motor A can serve in motor B's stead provided A has at least as high a power rating as B, we use the following Prolog statement:

```
can_substitute(A,B) :-  
    motor(A), motor(B),  
    power(A,A_power), power(B,B_power),  
    A_power>=B_power
```

The above Prolog production (rule) states that "A can substitute for B provided A is a motor, B is a motor, A's power rating is A power, B's power rating is B power, and $A \text{ power} \geq B \text{ power}$ ". Prolog requires, of course, that the predicates "motor" and "power" be defined. Defining the former requires that we name each motor with its own symbol, say m1, m2, etc. We then define the predicate "motor" with a series of declarations as follows.

```
motor(m1)  
motor(m2)  
motor(m3)  
etc.
```

The power ratings of the various motors are communicated in a straightforward fashion, as follows.

```
power(m1,1.7)  
power(m2,3.3)  
power(m3,3.3)  
etc.
```

Thus Prolog is a reasonable tool for communicating commonality constraints. Its drawbacks are many, however. Apart from its strange syntax, Prolog carries with it the overhead of explicitly declaring predicates like "motor" and "power" above, and using them in production rules. Also, the unification mechanism of Prolog (see Clocksin and Mellish [1]) is too costly and too general for this specific application. The knowledge to be communicated here is more along the lines of "object A is related to object B" than "statement P is true provided statement Q is true". A notation is needed which has the power of Prolog but is more familiar in form and does not carry with it the necessity to embrace a wider domain of application.

2 The CAPS Language

A tool under development for NASA, entitled "Commonality Analysis Problem Solver", or CAPS, has incorporated into its allowable input forms a notation for communicating commonality constraints. This notation was designed with the following objectives in mind: (1) it should be algebraic in nature, resembling the query languages of relational databases; and (2) it should incorporate enough of the power of the Prolog language so that the communication of a relationship between objects is easy and natural.

The nature of the knowledge to be communicated here requires that two objects be described for the purpose of relating one to the other. Therefore it is convenient to use two separate algebraic expressions. Each of these expressions is a database query which isolates a subset of the set of objects under consideration. The two general forms are as follows:

```
for <expression1> allow <expression2>
for <expression1> disallow <expression2>
```

The above notation either "allows" or "disallows" the substitution of objects satisfying <expression₂> for objects satisfying <expression₁>. Thus for a database containing information on wrenches, the statement

```
for millimeters=10 allow inches=13/32
```

might be used to permit the cost analysis phase of CAPS to consider using a 13/32 inch wrench in place of a 10 millimeter wrench.

What CAPS borrows from Prolog is its view of free variables and the comparison/assignment operator. In Prolog, the "=" symbol is used both for comparison and for assignment. If both sides of the equality can be evaluated, then a true or false value will be returned in the usual way. However, if one side of the equality is a free variable, i.e. a variable which has not been bound to a value during the current firing of the current production, then the other side's value is bound to it and a value of "true" returned. Using the same hypothetical wrench database as above, consider the following statement.

```
for millimeters=x allow inches>=0.0394*x and inches<=0.042*x
```

In this example, a larger set of wrenches is considered. The expression

`millimeters=x`

refers to the entire database, since x is a free variable and hence can be bound to any one of the values of the “millimeters” attribute. The key here is that once x is bound in that first expression to a specific value applicable to a specific wrench, the remainder of the wrenches in the database are then examined to see if any of them satisfy the expression

`inches>=0.0394*x and inches<=0.042*x`

As before, any which do are accepted as substitutes for the wrench in question. The same process is repeated for each wrench in the database.

Unlike Prolog, which is case-sensitive and dictates that all variables begin with a capital letter, CAPS is not case-sensitive and allows any identifier not already bound to an attribute name, a keyword, or a value to be used as a free variable.

3 Semantic Issues

A database query is a single boolean expression which identifies a subset of the elements in the database. It is a declarative statement about the properties of the members of that subset. For example, with a relational database it would be possible to fetch the entire set of wrenches for which

`inches > 5/16 and inches <= 7/8`

One could be sure then that in the subset fetched all wrenches would have the stated property. The semantics of a CAPS “for” statement are not so simple, however. Each “for” statement, instead of being a declarative definition of a relation on a set of objects, is a command used to *modify* such a relation. The relation in question is the substitutability relation on a set of objects. During its analysis of a given database, CAPS initially assumes that there are no allowable substitutions except the trivial ones, in which each object “substitutes” for itself. As each “for” statement is fired, the relation is altered to achieve a cumulative effect. We say that the CAPS notation is “history sensitive” in that the effect of each “for” statement depends not only on the statement itself but also on the internal state of CAPS, and that internal state is a function of all prior CAPS commands.

The usefulness of this aspect of CAPS can be seen in the following example, involving a database containing information on storage tanks.

```
for volume=x allow volume>=x
for liquid disallow gas
for gas disallow liquid
```

The net effect of firing the three “for” statements in the sequence indicated is to allow larger tanks to substitute for smaller ones, but not to allow the substitution of tanks designed to hold liquids for those designed to hold gasses, or vice versa.

Another semantic issue is that of binding times. The binding of a free variable takes place many times during the execution of a single “for” statement. For this reason CAPS must incorporate a very flexible strategy for binding variables to values. To illustrate, consider the “define” statement, with syntax as follows:

```
define <identifier> as <expression>
```

Here the identifier is associated with an unevaluated expression. If free variables appear in that expression, they will take on whatever values are current when the identifier is used in some subsequent computation, such as the firing of a “for” statement. In the same way, if there are field names in the expression, they will take on the appropriate values each time the identifier is referenced. Consider, for example, the sequence of CAPS commands that follows.

```
define rel_power as power/weight
for rel_power=x allow rel_power>=x
```

Here “rel power” is defined as the ratio of power to weight. Since these are both field names, it makes no sense to evaluate rel power at the point where it is defined. Rather, the expression “power/weight” is stored internally and evaluated each time it is needed. In the example, if there are n items in the database, the expression is evaluated n^3 times to fire the “for” statement. The wastefulness of this approach is clear, since there are in fact only n possible values for “rel power”. To avoid the extra complexity, CAPS allows its user to add “rel power” as a new field in the database. The statement

```
add rel_power
```

accomplishes this, provided “rel power” has already been defined as above.

To further illustrate, we use a variant of a type of commonality analysis problem originally formulated by Thomas ([3]). Consider a database containing information on utility interface plates. Each interface plate consists of a set of connections for utilities. The presence or absence of a given connection is represented by a TRUE or FALSE value in a boolean field. For example, a database entity having a TRUE value in its “potable water” field represents an interface plate incorporating a potable water connection. The following set of CAPS statements enforces the constraint that no interface plate may substitute for another unless the substituting plate has at least those interfaces which are present on the plate for which it substitutes.

```
allow all
for avionics_air disallow not avionics_air
for nominal_power disallow not nominal_power
for high_power disallow not high_power
for fire_detection disallow not fire_detection
for data_management disallow not data_management
for thermal_control disallow not thermal_control
for hygiene_water disallow not hygiene_water
for nitrogen disallow not nitrogen
for potable_water disallow not potable_water
for hygiene_waste disallow not hygiene_waste
```

The first statement, “allow all”, initially permits all substitutions. The subsequent statements restrict the substitutability relation until it satisfies the requirement. A cost analysis undertaken at this point will consider the elimination of any interface plate design which incorporates a set of interfaces which is a subset of those offered by some other design. Whether or not CAPS actually recommends the elimination of such a design depends on the cost information conveyed to it.

4 Communicating Cost Information to CAPS

CAPS has two separate mechanisms for determining costs. One of those is the default costing mechanism, derived from the cost functions used in NASA’s System Commonality Analysis Tool (SCAT) (see [2] and [5]). It consists of a set of relatively fast compiled-in cost functions. These functions are fairly standard and will not be discussed here. It is the data-dependent nature of costs that is of interest in the context of knowledge representation, and in recognition of that

fact CAPS incorporates a facility for describing highly tailored domain-specific cost information.

The feature of CAPS which facilitates the tailoring of cost information is the same defined variable feature discussed above, along with a set of predefined functions which have meaning only in the context of a cost analysis. A CAPS cost analysis has the task of grouping the elements of a database into subsets which are *components* of a partition. A partition of the database into such subsets amounts to a proposed solution to a commonality analysis problem. Each such subset has a distinguished representative which is proposed as a substitute for each of the other objects represented in the subset. During a typical CAPS cost analysis, many such subsets and representatives will be considered. The cost function employed by CAPS, whether it is a default cost function or user-defined, allows CAPS to attach a cost to each (subset, representative) pair. Since there is a need to identify costs associated with an entire component, rather than a single entity within the database, special-purpose functions must be employed. Following are a few such functions:

<code>cmax(<field name>)</code>	Maximum over all values of the given field within the component.
<code>cmin(<field name>)</code>	Minimum over all values of the given field within the component.
<code>csum(<field name>)</code>	Sum of all values of the given field within the component.
<code>csize</code>	Number of objects in this component.

Consider, for instance, the following user-defined cost function:

```
define linear_cost as ddt&e + csum(quantity)*unitcost
```

In this example, (1) “`ddt&e`” represents the design, development, test and engineering costs associated with the object, (2) “`quantity`” represents the number of copies of the object which must be produced, and (3) “`unitcost`” represents the marginal cost of producing each item, assuming no learning curve is used.¹ All three are field names. To make CAPS switch from its default cost function to this user-defined function, we use the following statement:

```
use linear_cost
```

¹CAPS' default cost functions are capable of incorporating a learning curve, and facilities are provided for building a learning curve component into a user-defined cost function as well.

During the analysis of the cost of a given component, the fields “ddt&e” and “unitcost” will apply only to the chosen representative, whereas “quantity” will range over the entire component, since it is used as an argument to “csum”.

As a final example, consider the interface plates database discussed in the foregoing section. In that example, the substitutability relation was designed so that plate A was allowed to substitute for plate B if B's set of interfaces was a subset of the set of interfaces on plate A. This assumption precludes the possibility of altering plate A in such a way that it incorporates additional interfaces in order to take on the functionality of plate B, or of manufacturing a new interface plate incorporating all interfaces present on both plates. If that approach is taken, it is no longer necessary to restrict substitutability so strictly. We simply adjust our concept of what it means for one plate to “substitute” for another. Regardless of which plate is chosen as the “representative” of a given component, it is assumed that a plate will be produced which incorporates the entire set of interfaces present on all plates in that component. We need, of course, a way of measuring the cost of producing such a plate. The following sequence of CAPS statements defines a cost function which is compatible with this strategy.

```
define combined_interfaces as
    cmax(avionics_air)      + cmax(nominal_power) +
    cmax(high_power)       + cmax(fire_detection) +
    cmax(data_management)  + cmax(thermal_control) +
    cmax(hygiene_water)    + cmax(nitrogen) +
    cmax(potable_water)    + cmax(hygiene_waste)
define new_plate_cost as ddt&e +
    csum(quantity) * combined_interfaces
use new_plate_cost
```

What makes the formula work is the fact that boolean attributes are assumed to have numerical values: 0 for false and 1 for true. Thus “cmax(avionics air)” has the value 1 if any of the interface plates in the component under consideration has an avionics air cooling interface. The parenthesized sum therefore amounts to a count of interfaces which would need to be present in any plate used as a substitute for all those present in the current component. Weighting factors can easily be added to the various terms to provide a more realistic estimate of the cost.

We can restrict the total number of interfaces on a given plate to a fixed number, say 7, using a “choke term” which causes the cost function's value to become unacceptably large should the program attempt to combine a set of plates which would require as its substitute a plate having more than that number of interfaces. Such a choke term appears in the following example, where the cost function is

identical to the above except for the addition of the choke term.

```
define choked_plate_cost as ddt&e +
    csum(quantity) * combined_interfaces +
    1.0E9*(combined_interfaces>7)
use choked_plate_cost
```

In this scenario, we no longer need to restrict substitutability in any way. A simple "allow all", then, would suffice to communicate to CAPS the lack of any constraints. That would not be the most prudent way to enter the cost analysis, however, because fewer constraints typically mean a more lengthy analysis. A natural way to constrain substitutability is to always choose the plate with the larger number of interfaces. This can be done as follows.

```
define number_of_interfaces as avionics_air + nominal_power +
    high_power + fire_detection + data_management +
    thermal_control + hygiene_water + nitrogen +
    potable_water + hygiene_waste
for number_of_interfaces=x allow number_of_interfaces>=x
```

After the above constraint is imposed, the solution proposed by CAPS will always use as a representative for each component the interface plate needing the fewest additional interfaces in order to serve as a replacement for all other plates in that component.

5 Conclusion

CAPS incorporates into its design notations and operations uniquely suited for describing commonality constraints and cost information in preparation for a comparative cost analysis. The notations are intuitive and easily understood, yet powerful enough to make CAPS applicable to a broad range of commonality analysis problems. For more complete information on the CAPS tool, see [4] or contact the author for more recent documentation. Generalizations of the CAPS notation to more general database applications are presented in [6].

References

1. Clocksin, W. F. and Mellish, C. S. "Programming in Prolog". New York, Springer-Verlag, 1981.
2. Marshall Space Flight Center. Commonality Analysis Study, User Manual for the System Commonality Analysis Tool (SCAT), D483-10064, March 1987.
3. Thomas, L. D. "A Methodology for Commonality Analysis, with Applications to Selected Space Station Systems". Dissertation, The University of Alabama in Huntsville, Huntsville, AL, 1988.
4. Yeager, D. P. "Development of a Prototype Commonality Analysis Tool for use in Space Programs". Section XXXI of NASA Contractor Report CR-183553, Research Reports - 1988 NASA/ASEE Summer Faculty Fellowship Program. George C. Marshall Space Flight Center, 1988.
5. Yeager, D. P. "A Formalization of the Commonality Analysis Problem and Some Partial Solutions", BER Report Number 454-69, The University of Alabama, February 1989.
6. Yeager, D. P. "A History-Sensitive Approach to the Description of Binary Relations", *Structured Programming*, vol. 10, no. 3, pp. 157-163.

A Knowledge-Based approach to Configuration Layout, Justification and Documentation

F.G. Craig, D. E. Cutts, & T.R. Fennel
Boeing Computer Services M/S JA-74
Huntsville Artificial Intelligence Center

C. Case & J. R. Palmer
Boeing Aerospace M/S JY-33
499 Boeing Blvd.
Huntsville, AL 35824-6402

COPYRIGHT 1990 THE BOEING COMPANY
PUBLISHED WITH PERMISSION

Abstract

This paper describes the design, development, and implementation of a prototype expert system which could aid designers and system engineers in the placement of racks aboard modules on Space Station Freedom. This type of problem is relevant to any program with multiple constraints and requirements demanding solutions which minimize usage of limited resources. This process is generally performed by a single, highly experienced engineer who integrates all the diverse mission requirements and limitations, and develops an overall technical solution which meets program and system requirements with minimal cost, weight, volume, power, etc. This "systems architect" performs an intellectual integration process in which the underlying design rationale is often not fully documented. This is a situation which lends itself to an expert system solution for enhanced consistency, thoroughness, documentation, and change assessment capabilities.

1.0 General Configuration Definition Issues

One of the major issues faced by any aerospace program is the need to consistently apply requirements, constraints, and resources to optimize the layout of equipment in an end item deliverable piece of hardware in the midst of changing environments. The change mandates can result from changing customer requirements, newly derived requirements, reduced program budgets, technological influences or personnel changes. All these changes tend to impact engineering processes, often rendering current approaches inappropriate or current solutions inadequate. In the remainder of this section we present a list (by no means exhaustive) of general issues which must be faced throughout a program's life cycle :

- **Fleeting expertise** : Turnover of domain experts represents a serious drain on program continuity and often causes work to be adversely impacted since significant portions of domain knowledge and program history often reside with individuals.
- **Productive use of resources** : Much layout work is both repetitive and resource intensive in nature. Allowing for automation of such repetitive tasks to be accomplished early in the process results in more resources being available for "real" engineering work to be performed later. This is usually a direct result of complimenting engineering expertise with tools which allow problems to be solved at a more abstract level and to off-load the repetitive portions of the task to the automated process. For example, engineering resources may be diverted to cost proposed changes, document accepted changes, and implement new procedures. Because of this type of required reaction, program continuity and productivity can be affected.

- Documentation of engineering rationale : All major programs have periodic requirements to review progress and to answer not only the question of "What has been done?", but also "Why was it done this way?", and "Why can it not be done this way?". The last two questions require the documentation, presentation and defense of engineering rationale. The problem is to provide a sound defense in areas where adequate documentation is generally missing, the expertise may have been lost, or rules may not have been codified, consistently applied, or documented.
- Multi-discipline inputs : Decisions made during the configuration process typically originate across several disciplines and organizational boundaries. Disciplines may or may not be aware of the impact of their decisions on other disciplines. These multi-disciplinary inputs to the engineering process highlight the need for a uniform approach to acquiring and representing those inputs.
- Explicit decision parameters and criteria : For engineering problems of any significant complexity, there is a need for the consistent application of clearly defined problem parameters and dynamic criteria to the engineering process. This is particularly true when these parameters and criteria come from various disciplines.
- Limited alternatives : Often the iterative engineering process is not fully utilized beyond a baseline "satisficing" solution (where the result is not optimal but merely satisfies most of the criteria). Little time is left to consider alternative analyses, configurations, or development paths. Better options may be overlooked because no tool/capability exists for quickly modelling and analyzing engineering alternatives.
- Problem of scale : Unfortunately, major program setbacks often occur because engineering solutions which worked well for small problems (or subsets of the larger problem) do not scale up well. This is particularly true when manual engineering approaches which were controllable and acceptable for the smaller problem are applied to large integrated programs.

As mentioned, the above list is not intended to be exhaustive, but is presented to serve as a reference for the next section.

2.0 Rack Layout Problem Description

As an initial test problem we have selected the Space Station Freedom module configuration task. This effort is similar to that required in many aerospace configuration layout applications in terms of complexity, constraints, and resources. It is above average in the number of expected major changes and long period of implementation. These factors make the configuration problem an ideal candidate for a knowledge based system.

The particular test domain area is that of rack placement aboard station modules. The racks provide the physical packaging for station services and functions. The objective of the rack placement process is to position a group of racks aboard modules in a configuration that minimizes utilization of resources, optimizes operational efficiency, and meets as many requirements and constraints as possible. The rack layout problem is representative of various configuration layout problems faced within many aerospace programs. Currently three other potential applications for this type of system have been identified within Work Package 1 of the SSFP, and it is expected that a number of additional spinoff applications will surface. Also, work performed on this project could be applied to areas external to the SSFP (other suggested areas include the outfitting of Commercial Aircraft and the Manned Mars Mission).

We are currently researching knowledge-based systems approaches to aid in this problem. The purpose of the research is to attempt to overcome the following perceived problems.

Fleeting Expertise : Currently, only one person in the Space Station program is identified as an "expert" on rack placement.

Claim : Development of an expert system to document the analysis, criteria, and engineering processes used by the expert will allow knowledge needed to solve the problem to be preserved and to be available for review by "non-experts".

Productive use of resources : The current manual approach to this process is quite time consuming and labor intensive. Rack layout reconfiguration for the station must be performed in step with other changes to the program. Due to lack of time, changes to rack configuration often represents an "acceptable" rather than an "optimal" solution.

Claim : The expert system is expected to significantly reduce the amount of time required to produce a new configuration. Additionally, the rules and procedures used by the system will be applied consistently through the automated program. Also, the expert system doesn't "forget" the rules or procedures during periods when the expert is busy with other task. Indeed, such an expert system could be used to train less skilled personnel to perform the task and can be used by the expert to explain the required analyses and procedures for the rack placement process.

Defense of engineering rationale : SSFP has a requirement for periodic reviews where design decisions must be justified.

Claim: In a rule-based system, the engineering rationale for a particular configuration is implicit in the set of rules used to generate that configuration. This engineering rationale provides placement justification and explanation. One of the objectives of the current work is to extract intelligible rationale from the set of rules used to generate the configuration.

Multi-discipline inputs: A large number of constraints exist between racks within and across modules. When these constraints are imposed on a large number of racks, a difficult constraint problem emerges. This problem is compounded by the fact that these constraints are imposed by different domain areas (such as power, thermal, cost, safety, etc.) and may be physical, functional, or operational in nature.

Claim : Experts from all applicable domain areas provide input to the rules and procedures used for the automated placement process. An additional advantage is that a unified approach to the acquisition, analysis, and representation of this domain knowledge can be developed and more easily verified by the experts from the various disciplines.

Explicit decision parameters & criteria : The lack of a uniform approach to explicitly identify applicable parameters and then consistently apply domain rules for rack placement hinders both the ability to quickly produce optimal rack layouts, and the ability to provide justification for a particular configuration.

Claim : The objects, rules, and associated parameters can be printed, queried interactively, and dynamically changed. This makes explicit the answers to questions such as:

What impact did the rule

"IF
the rack is rated as 'noisy' ,
THEN
don't place it near the crew
sleeping quarters."

... have on the decision to place the rack?

Limited alternatives : Currently, analysis of rack configurations takes anywhere from several hours for the simplest changes to several weeks for more common changes. As expected, this does not leave much time for analyzing "What if...? situations."

Claim : Once the applicable parameters have been identified, and domain expertise has been captured, this expert system would support the ability to "tweak" priorities and constraints allowing engineers to analyze alternative configurations. The comparative "goodness" of rack configurations could be determined, and the tool could be used to suggest or support engineering change requests. Obviously, this does not imply that human expertise would no longer be needed. Rather it implies that more engineering analysis could be performed and human intuition could be used to fuller advantage by allowing the engineer to work at a higher level of abstraction.

Problem of scale : Currently the Phase 1 SSFP calls for only two modules and 4 nodes on the American portion of the program. Even this first phase of the program requires over 144 racks which must be assigned within a full range of physical, functional, and operational constraints. The multiplicity of constraints and the number of racks makes a manual approach to solving the problem nearly intractable.

Claim : While the number of racks and other "real world" objects is expected to remain relatively constant, it is anticipated that the number of constraint and control rules in the knowledge base will expand. No reliable data was available to estimate the bounds on the number of these rules. The tool selected for implementation set no upper bound on the size of the knowledge base (other than memory limitations). Speed was not a primary issue in this application, but reasonable response time was expected. The expert system incorporates domain expertise to control the focussing of rules which helps to limit the solution search (see the control layer in figure 1). Additional constraints can be easily added (or deleted) as knowledge about the racks and their interactions increases.

3.0 Implementation

The Nexpert expert system building tool from Neuron Data was selected for this project because it offered a number of desired features. Nexpert is a hybrid system supporting the representation of knowledge in objects and rules. It supports full inheritance and procedural attachment of methods as well as forward and backward rule chaining capabilities. It interfaces to user developed external routines as well as PC databases and spreadsheets. In addition, links to large databases such as Oracle and Informix are supported. Within this project we are currently a beta test site for a Hypercard/Nexpert "bridge" which allows communication between Nexpert and Hypercard facilities on the Macintosh II platform. Much of the explanation and training research is currently being performed using Hypercard. Nexpert is C based, runs on a wide range of hardware platforms, and offers a number of relatively inexpensive delivery options.

The prototype system software was implemented using a layered architecture to represent system knowledge (see figure 1). This layered architecture separates different types of knowledge and aids in development, debugging and maintenance of the system. Chandrasekaran [Ref 3] proposes a similar architecture in which tools might be developed for "problem classes" such as diagnosis or design. He proposes that particular problems within these "problem classes" share similarities and that "generic" approaches to solving them might be appropriate.

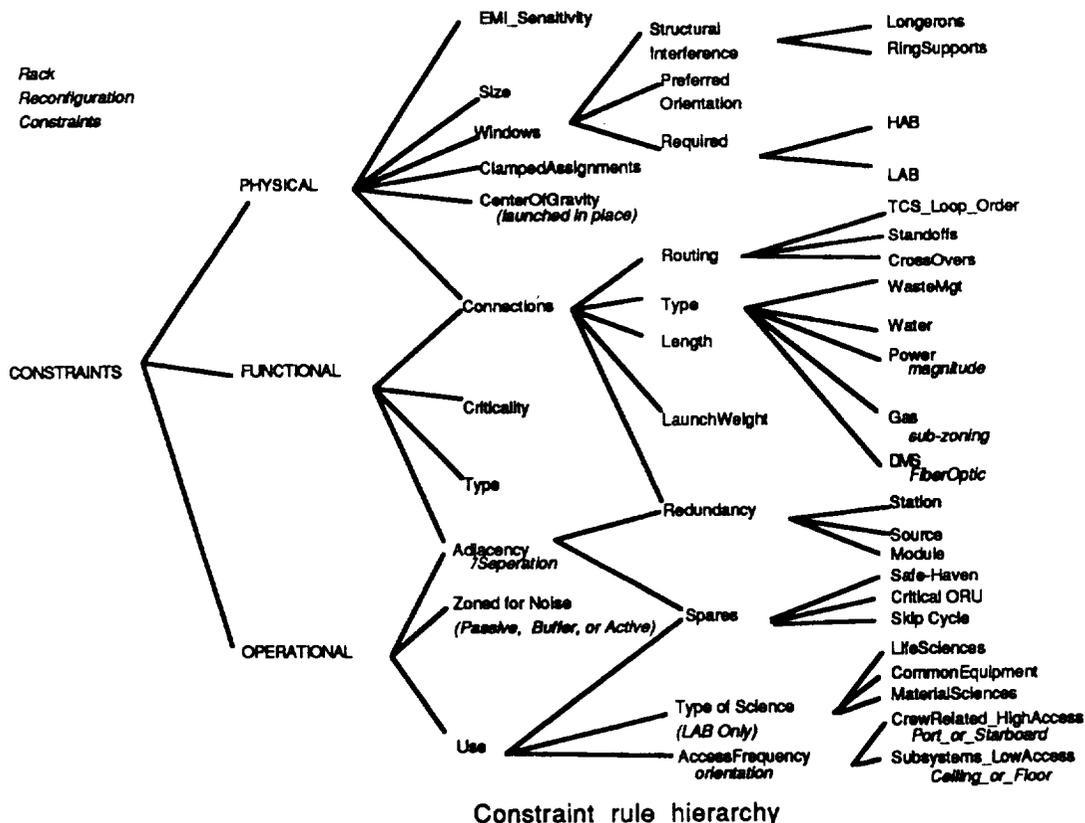
Data resides in the lowest layer. The data is currently stored in a spreadsheet format, but facilities exist in the Nexpert tool to retrieve data from a number of sources including PC spreadsheets, PC databases, Oracle and Informix. Data is used to support the next layer representing objects and their associated attributes. These two bottom layers together might be thought of as Object-Attribute-Value (O-A-V) triplets. "Real world" entities such as modules, racks, standoffs, utilities, etc., are

represented as objects in the system. Most of this type of knowledge was obtained directly from SSFP documentation. It should also be noted that this data might be obtained during the inference process or from some external source. This external source might well be an external routine which calculates a value and returns it to the object. This is analogous to attaching a "method" to an object.

The constraint layer contains all the constraint knowledge about the particular domain under consideration. This constraint knowledge is stored in rules and is a relatively "flat" knowledge base since this type of knowledge is concerned primarily with only a few "focal" objects. This knowledge base consists of a collection of "microscopic" rules to be used in solving the problem, and do not embody higher level "control knowledge" which a human expert would follow in applying the constraints.

The control knowledge (or meta knowledge) at the next level controls the direction of focus for the constraint knowledge. This level "prunes" the search space so that inappropriate constraint knowledge is not considered. Control knowledge is used to apply the constraint knowledge in much the same way a human expert would. An interesting offshoot of this project has been that the codification of this type of knowledge often helps to better define the problem solving process. This layer is also important in the explanation/justification of design decisions since explanations of design decisions made by the system need to be conveyed in much the same manner as a human expert's explanation.

The user interface represents the user's view of the system. For this application we have designed a "point and click" user interface in which the user manipulates racks within a module configuration. This interface provides input to the control layer about rack(s) to be moved. The control layer applies appropriate constraint knowledge at the next level. The constraint layer, in turn, obtains needed information from the lower levels. The final result (**no constraints violated**, **"soft" constraints violated** or **"hard" constraints violated**) is passed to the user interface where the user can query the system about the particular decision and what support was used in making the decision.



The explanation/justification layer supports access to all the lower levels of the architecture. Queries can be made of objects, constraint knowledge or control knowledge. Explanations differ in content for these different layers and in level of detail based on the level of expertise of the user. A more detailed discussion of this layer can be found in [Ref 1].

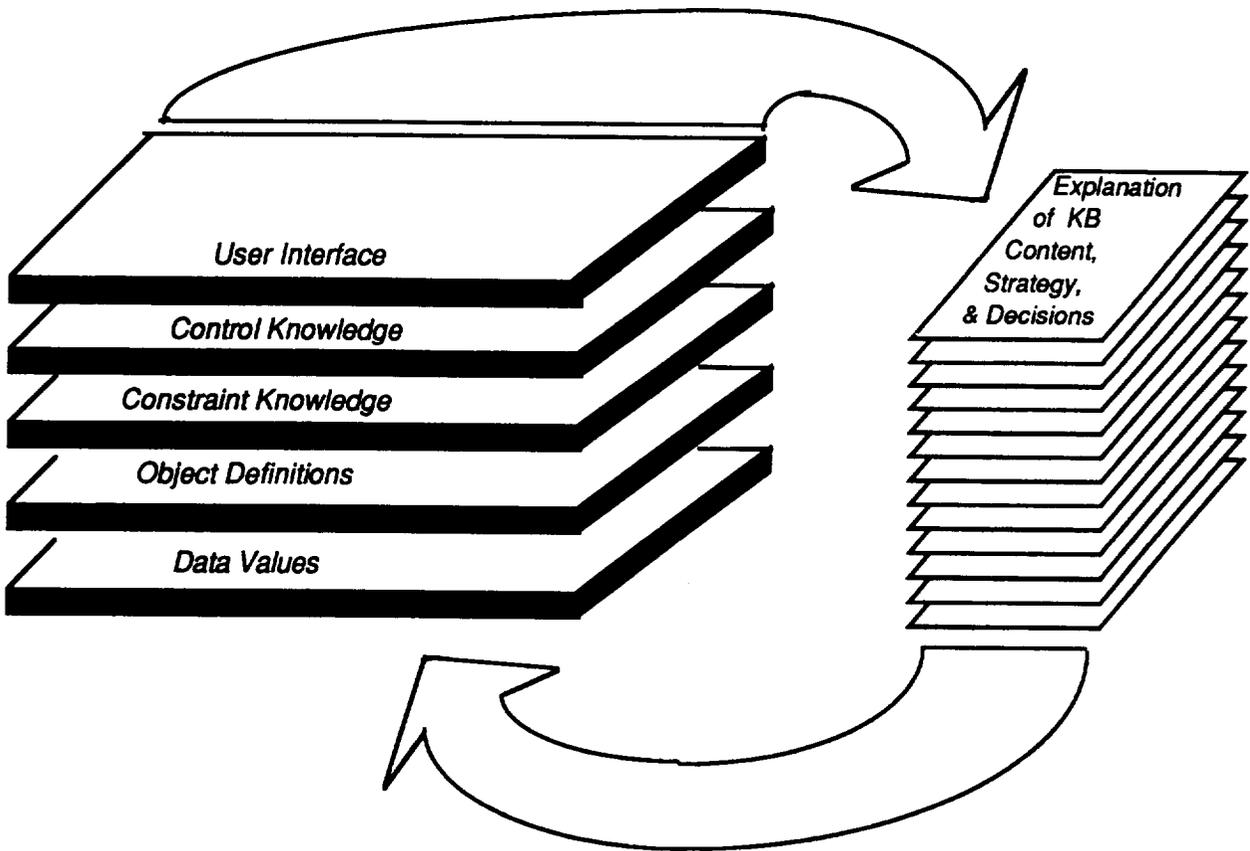


Figure 1

4.0 Issues

As with any expert system project, there were a number of issues critical to success. Some of these issues are described below:

Expert availability: From the project's inception, a "domain expert" was identified and has been available at every step in the development process. This expert understands the problem to be solved and is able to articulate his method(s) for solving the problem.

Knowledge Acquisition: The data required for this project comes from both SSFP documents and domain experts. Traditional interview techniques with the primary rack placement expert as well as other experts in related fields have been very successful. Domain experts have recognized the

potential utility of such an expert system and have supported it fully. In addition to interviews, the domain experts submitted "test cases" for the system to solve along with their conclusions/justifications as to why the move was good or bad (or could/could not be made). These test cases helped identify many weak areas in the system and helped to build the explanation facilities [Ref 1]. Early in the development process we began to use the system itself to acquire domain knowledge by running test cases against the system. This approach uncovered weak areas in the captured domain knowledge or incorrect assumptions. Thus, the tool itself has been used extensively in the acquisition process.

Verification and Validation: Test cases supplied by the domain experts as well as "working" interviews in which the system is used to test particular rack configurations have been used to test the system for correctness as well as its ability to provide meaningful decision justification. No work has been done to perform tests on the knowledge base for rule subsumption, rule contradiction, or cycles. Much of the system testing will continue to be empirical in nature.

5.0 Future work

One result of both the data acquisition and validation activities was that a larger number of people became aware of the project and began to look for ways to apply the work performed in the project to particular problems in their domain. As a result we anticipate that a number of spinoff projects will emanate from this IR&D work. Problems similar to the rack placement problem include resource allocation problems in which rack resource requirements are matched with resources supplied in the module to maximize the resource utilization. Another similar problem deals with the placement of payload racks (experiments, etc) within the lab module. This task must be performed repeatedly since experiments will continually be moved in and out. Another spinoff of this work may be in the training area. Much of the work being done to provide intelligent design justification and explanation could carry over into training new engineers on the SSF program.

As a component of a training system as well as design justification, we plan to interface this system with simulation systems to provide "deeper" justification or explanation by allowing the user to perform a simulation of a particular configuration during the design process. Adeli [Ref 2] reports on efforts to couple AI techniques with traditional mathematical techniques to aid in the engineering process.

6.0 Summary

Systems incorporating AI technologies to aid design will enhance the engineering process and will ensure that the decisions (and rationale behind them) are available in an intelligible format for future applications. Research in this area and prototype systems such as the one described here will help to clarify and define the engineering design knowledge capture requirements.

References:

- [1] Fennel, T.R., et.al, "Graphical Explanation in an Expert System for Space Station Freedom Rack Integration", 5th Conference on Artificial Intelligence for Space Applications, Huntsville, AL, May 1990
- [2] Adeli, H. & Balasubramanyam, K.V., "A Novel Approach to Expert Systems for Design of Large Structures", AI Magazine, Winter 1988, pp. 54-63
- [3] Chandrasekaran, B. "Generic Tasks in Knowledge-Based Reasoning: High-Level Building Blocks for Expert System Design", IEEE Expert, Fall 1986, pp. 23-30

**KIPSE1 - A KNOWLEDGE-BASED INTERACTIVE PROBLEM SOLVING
ENVIRONMENT FOR DATA ESTIMATION AND PATTERN
CLASSIFICATION**Chia Yung Han*, Liqun Wan⁺, William G. Wee⁺

*Department of Computer Science

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

A knowledge-based interactive problem solving environment called KIPSE1 is presented in this paper. The KIPSE1 is a system built on a commercial expert system shell, the KEE system. This environment gives user capability to carry out exploratory data analysis and pattern classification tasks. A good solution often consists of a sequence of steps with a set of methods used at each step. In KIPSE1, solution is represented in the form of a decision tree and each node of the solution tree represents a partial solution to the problem. Many methodologies are provided at each node to the user such that the user can interactively select the method and data sets to test and subsequently examine the results. Else, users are allowed to make decisions at various stages of problem solving to subdivide the problem into smaller subproblems such that a large problem can be handled and a better solution can be found.

I. INTRODUCTION

One of the major goals in computer-based problem solving is to provide computer users more flexibility and guidance in using the available computing resources. To facilitate user select proper methodologies and decide what computing tools to use in solving problems of a particular problem domain, specialized knowledge of expertise needs to be incorporated. A current research thrust in achieving this goal is to build a knowledge-based problem solving system where numeric processing is integrated with symbolic processing (4, 7, 10).

A problem solving environment (PSE), as broadly stated in (3), is an integrated multitasking system that supports and assists user in the solution of a given class of problems. Normally, within a more established discipline area or domain, collections of numerical programs in various forms and capabilities are available for use. These programs may be in the form of callable subroutines that can be accessed within an application program or through the use of specialized high level languages. There are currently many examples of PSE available in some problem areas. For instance, in the area of statistical computing, there exist packages such as SAS, BMDP, and others. In the arena of CAD/CAM where numeric computing routines for data display and geometric information manipulation are incorporated, there exist systems such as AutoCAD, GeoMod, just to name a few. Although these PSEs employ well-proven techniques of the problem area and may relieve the burden of code writing from the user, knowledge about the numeric algorithms and techniques used in these programs and facilities that allow a user to interactively define the process and examine the various results are not readily available to the user.

To augment PSE with capabilities to incorporate knowledge to manage symbolic manipulations, numerical routines, and problem solving strategies AI techniques are used.

AI techniques, specifically, methodologies from the expert systems technology enhance greatly the capability of PSE which include strategies for finding nondeterministic solutions, techniques for constraint propagation and search, and various knowledge representation schemes such as semantic nets and frames.

Many features are needed in a knowledge-based PSE. Basically, it should contain the following components: a friendly user interface, a bank of numeric programs, a kernel for system control with a variety of data structures for handling different data types, output display for both text and graphics, preprocessor and postprocessor modules for interfacing the various numeric programs, and a knowledge base. In Section II, a PSE architecture for data estimation and pattern classification is presented and its components explained. In Section III, the detailed implementation issues of our system KIPSE1 are discussed. Section IV discusses some features of the KIPSE1 system. Concluding remarks will be given in Section V.

II. A PSE ARCHITECTURE FOR DATA ESTIMATION AND PATTERN CLASSIFICATION

Problem solving process is a highly intelligent behavior. It is domain dependent and user involvement is extensive. For any new problems, the questions the user wants to answer are usually non-routine and many time very difficult (2). To model the general problem solving strategy and to build a system for all problem areas is almost impossible or at least very inefficient. Therefore, only PSE concentrated on a specific application domain is built in practice. As an environment the user involvement issue requires that it be convenience of use with effective user interfaces.

A PSE for data estimation and pattern classification tasks that are common in many areas of application has been proposed by the authors. A data estimation problem is given below to illustrate this. In data estimation, one is often asked to find a mathematical model to best explain the given data set. Many statistical packages, which may reside in many different computer systems, can be used for this purpose. For example, the multiple linear regression PIR routine in the BMDP statistical package and the regression REG routine in the SAS packages. After choosing the package, user still needs to have some statistical knowledge about each model, its assumptions, requirements, and so on, in order to pick the right model. User also needs to know the specific language for each package in order to use these routines. Many pages of numerical output are generated from these routines and a casual user usually needs expert assistance for a meaningful interpretation of the results. The process may repeat many times before an acceptable solution is determined. A system which integrates all these in one environment will greatly increase the productivity of the user in solving the problem. The PSE architecture for data estimation and pattern classification is shown in Figure 1.

The entire problem solving process can be divided in the following steps: problem formation, process generation, process setup, and postprocess. At the first stage, the problem formation includes problem parameter definition, data format conversion, and system initialization. In the process generation step, the strategy for solving the problem is formulated by the user based on the domain knowledge and previous results. Once a process is defined to perform certain task, the system will automatically set up the interface to the application program and initialize the computing processes. During the postprocess stage, useful information from the computing routines are extracted, converted into proper format and stored to the database for later retrieval. Output display routines are invoked to display the results both in text and graphical forms. A rule-based result interpreter is used to interpret the results. User examines the results and makes decision to either redesign the process or stop.

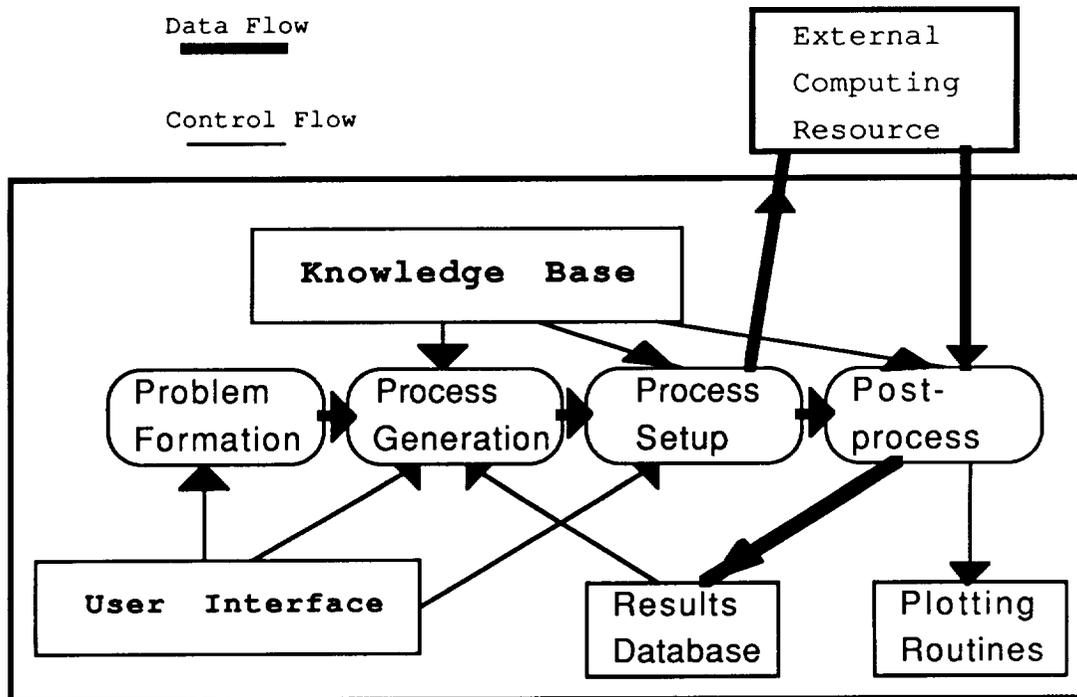


Figure 1. An Architecture for PSE for Data Estimation and Pattern Classification

III. SYSTEM IMPLEMENTATION

It has been determined from our past experiences in designing various expert systems that higher end expert system shells can provide the necessary tools for the implementation of the KIPSE1. The system requirements essentially consist of 1)powerful data structure to represent complex process, 2)flexible knowledge representational schemes to represent domain knowledge, 3)user interface with graphical capability, and 4)interface to other computer systems.

The KIPSE1 system is built on top of the Intellicorp's Knowledge Engineering Environment, the KEE system. Frame-based structure is used to represent the attributes of individual process. This frame-based structure is an object-oriented system where objects are structured as units with slots. Attributes are specified in slots. Slots may contain descriptive, behavioral, or procedural information, and relations are expressed using slot values (5). The KIPSE1 is organized as a hierarchical tree structure where each individual process is incorporated as a node within the structure. As an example, the structure of pattern recognition (PR) process is shown in Figure 2. At each node the control of the problem solving is coded in one of attached procedures associated with that node. Figure 3 shows an example of a procedural method written in Lisp. This procedure is the control procedure for the classifier design process.

The objective of pattern classification problem is to determine to which class a given data sample belongs. The process of designing a pattern recognition system can be divided as data gathering, normalization, data structure analysis, classifier design, and testing (error estimation) (11). The main purpose of the data structure analysis is to explore the data set structure in order to design the classifier in a later stage. The operations in this stage include feature extraction, clustering, statistical tests, and modeling. The classifier design is also

called training or learning stage in which data samples are collected from various classes and the class boundaries are determined. There are several common classifier design methods used in this stage. They include linear classifier, quadratic classifier, piecewise linear classifier, and nonparametric classifier.

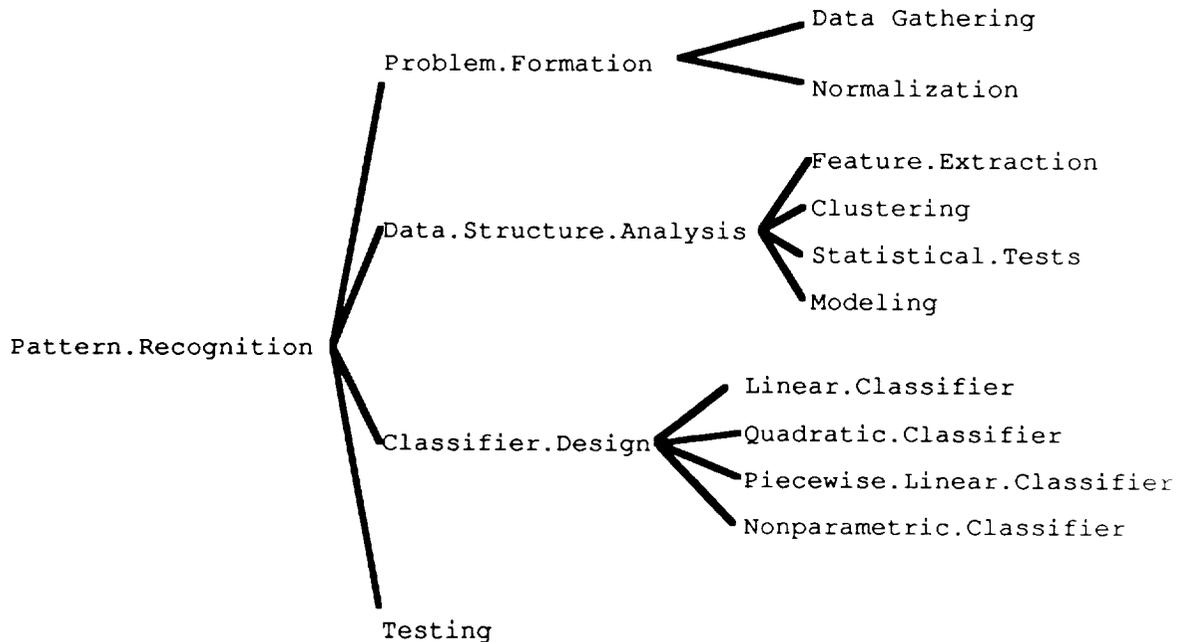


Figure 2. A Structure for the Pattern Recognition Process

Our philosophy is to provide as many methods as possible to the user to try out at each node. Just as no general data structure is suitable for all problems, there is no general method applicable for all the pattern recognition problems. The best that one can do is to provide a good environment with all available methods so the user can pick up right method for the problem under consideration based on both the nature of that problem and the user's previous experience. A user might even try different methods, compare the results, and select the best one among the various trials. For those users who are not familiar with those methods provided in the system, heuristic rules are provided to help the user find appropriate one to use based on the general nature of the problem and usage requirements and limitations of each method.

The KIPSE1 system is menu-driven. Figure 4 shows the screen menu for the selection of classifier design methods as an example of a stage of running process. Four types of classifier methods, which include linear, quadratic, piecewise linear, and nonparametric classifier are provided to the user. In addition, expert knowledge is available in the HELP option.

Different types of knowledge are imbedded in the environment. In addition to the one of selecting a method as mentioned above there is specific knowledge for helping user to choose right parameters for each selected method, knowledge to help interpret the results, and knowledge for error handling. These knowledge types are coded explicitly in the rule form and is structured as shown in Figure 5.

```

(lambda (self node)
  (let (process)
    (self process (pop-up-cascading-menu-choose (make.cascading.menu
      :POP-UP '((" Linear Classifier" 'Linear.Classifier)
        (" Quadratic Classifier" 'Quadratic.Classifier)
          (" Piecewise Linear Classifier" 'Piecewise.Linear.Classifier)
            (" Nonparametric Classifier" 'Nonparametric.Classifier)
      :PARENT *kee-root-window*
      :TITLE "CLASSIFIER DESIGN: Choose One of the Following Method"))))
    ; Display the menu on the screen, ask user to choose method
    (cond ((not (equal process 'help)) (put.value self 'process process))
          ; Save user selection in slot process
          (t (query '(THE PROCESS OF CLASSIFIER.DESIGN IS ?X)
                    'Classifier.Design.Help.rule.class)
              ; Help user find right process using the knowledge provided by the expert
              ))
    )
    (unitmsg (get.value self 'process) 'control node)
  )
)

```

Figure 3. Control Procedure for Classifier Design Process

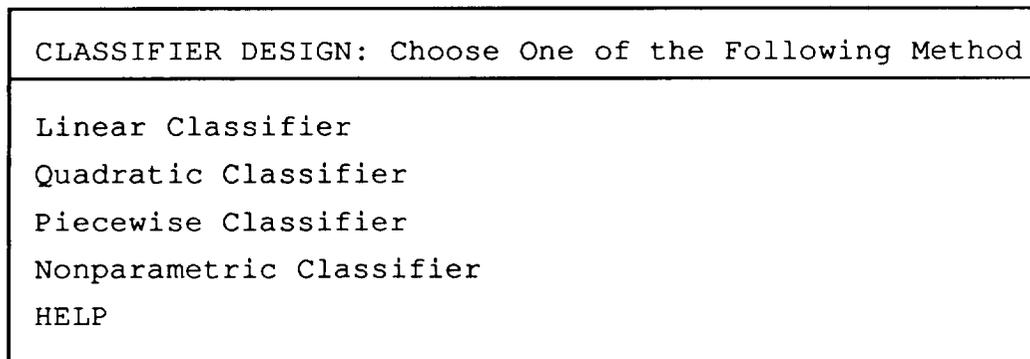


Figure 4. A Running Menu

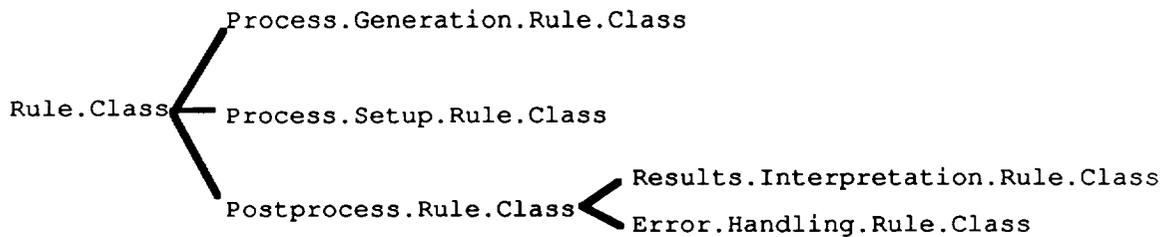


Figure 5. The Structure of the Rule System

Rules are used to represent knowledge. An example of a rule is given here. This is a result interpretation rule used in the clustering process. The purpose of the clustering process in pattern classification or recognition is to find a method for combining objects

into groups or clusters such that objects in each cluster are similar. Similarity is defined by a measure of similarity provided by the user. If this measure is set improperly, say, it is too small, many meaningless small clusters will result. In this case the measure needs to be adjusted. From the domain expert it has been determined that the total number of clusters in most cases should be kept under four. A rule that convey this expert knowledge may be included as shown in Figure 6.

```
CLUSTERING . RESULTS . INTERPRETATION . RULE . 001  
  
(IF (THE RESULTS OF CLUSTERING IS ?X)  
    (LISP (> ?X 4))  
    THEN  
    (THE PERFORMANCE . EVALUATION OF CLUSTERING IS NOT . ACCEPT))
```

Figure 6. An Example of Rule

IV. SYSTEM FEATURES

In this section, some of the basic features of the system are discussed. They are ease to integrate new software, availability of various methodologies, and user involvement in generating hierarchical decision tree.

An important feature of the system is its extensibility. General procedure is set up to integrate any software to the system. As explained earlier, a new software is defined as a new node in the system tree structure. The process of adding a new software is equivalent to inserting a new node to the proper position of the existing tree structure.

The information associated with each node is categorized into two classes. One is the global information used in interacting with the system. It includes software location, description, usage, results, performance evaluation, and the rule class that provides conditions for accessing the methods. Another category is the local information relating to actual computing process initialization. It includes preprocess, postprocess, result interpretation rule class, and error handling rule class. The general node structure is shown in Figure 7.

As an example, to integrate the method of multiple linear regression, say, the P1R routine of the BMDP package, which resides in a node called 'ucaicv' on the network, the following relevant information is stored in the pertinent node of P1R and shown in Figure 8.

It has long been noted in the field of pattern recognition that the key to problem solving does not rely only on the sophistication of the available algorithms but also lies heavily on the proper representation of the pattern structure of the data (6). Quite a few complex problems can be broken down into simpler problems. Each subproblem becomes easier to be solved. As stated in (1), hierarchical structure from a very important data representation and decision tree are naturally suited for problem solving with such a data representation.

In the KIPSE1 system both the problem and its solving process are also represented in the form of frame of structure. One of the important features of the KIPSE1 is that it provides a mechanism to allow user to break a problem node down into several subproblem nodes. These problem nodes can be organized hierarchically by setting up two special slots:

parent slot and children slot. Each problem node captures the partial decision process and the decision tree represents the problem solution.

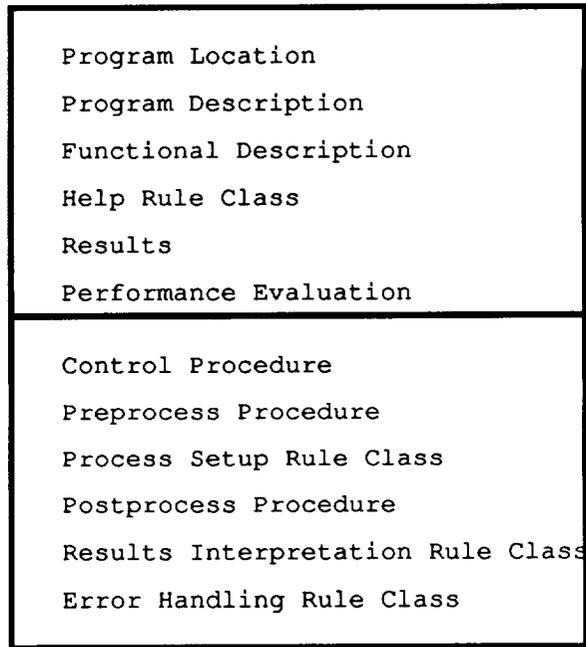


Figure 7. A General Node Structure

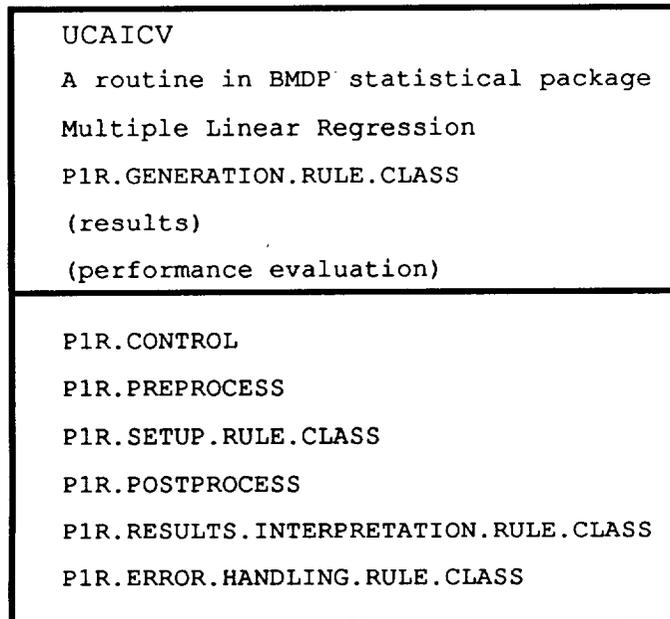


Figure 8. An Example of Integrating an External Program to the System

Several strategies are incorporated into the system. They include ad hoc strategies defined by the nature of the problem based on a priori experience, by visualization of input data set, by performance evaluation, and by exploration and trade-off study.

A decision tree solution to a data estimation problem is shown in Figure 9. The original set of 91 data points has been partitioned into four subproblem spaces of 22, 20, 24, and 25 data points. Shown along each node are the regression equation associated with the sample data set, the respective mean squared error, and the decision made to subdivide the problem space.

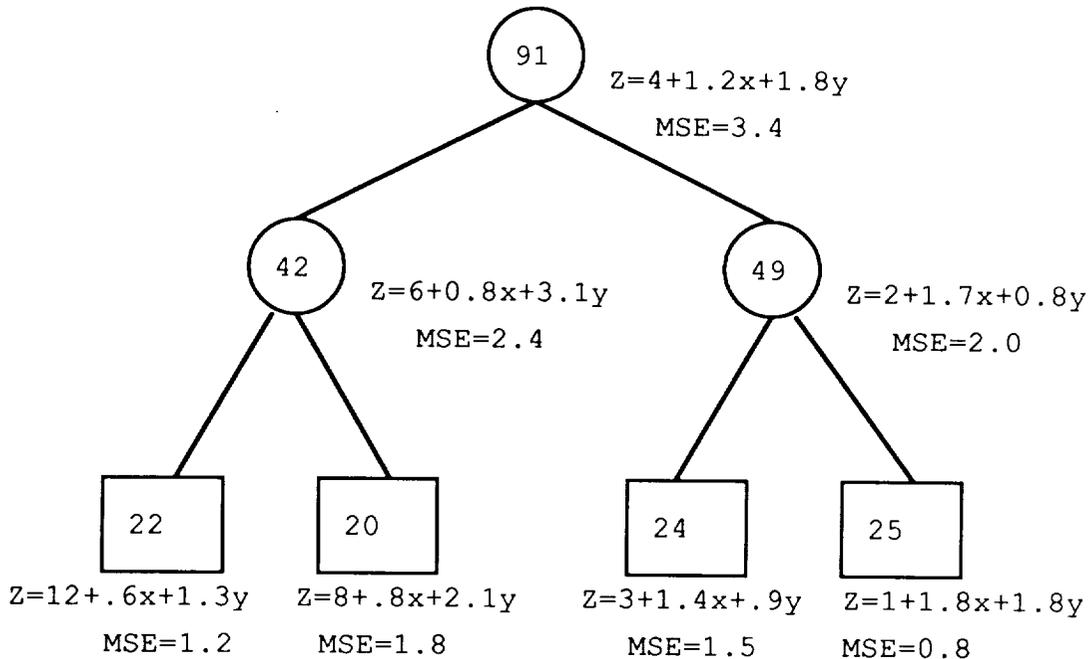


Figure 9. An Example of Solving a Data Estimation Problem Using Decision Tree

V. CONCLUDING REMARKS

A knowledge-based interactive problem solving environment for data estimation and pattern classification has been presented. The entire problem solving process is viewed as a multi-step decision making process. It is a learning process that at each step some kind of exploratory actions take place. The KIPSE1 provides this powerful problem solving capability. It is easy to use, easy to expand, and special knowledge is include to assist the user throughout the problem solving process. The system has been implemented on top of a powerful commercial expert system development shell. Its flexible knowledge representation and useful interface facilities have provided means to organize the numeric computing tools, to incorporate symbolic knowledge, and thus create a powerful problem solving environment.

VI. REFERECES

- [1] Dattatreya, G. R. and Kanal, L. N., "Decision Tree in Pattern Recognition," *Progress in Pattern Recognition 2*, Kanal and Rosenfeld, eds, North-Holland, 1985, pp. 189-239.
- [2] Feldman, S., "The How and Which of Problem Solving Environment," *Problem Solving Environment for Scientific Computing*, Ford and Chatelin, eds, Elsevier Science Publishing Co., 1987, pp. 23-32.
- [3] Ford, B. and Iles, R. M. J., "The What and Why of Problem Solving Environments for Scientific Computing," *Problem Solving Environment for Scientific Computing*, Ford and Chatelin, Eds, Elsevier Science Publishing Co., 1987, pp.3-22.
- [4] Gladd, N. T. and Krall, N. A., "Artificial Intelligence Methods for Facilitating Large-Scala Numerical Computations," *Coupling Symbolic and Numeric Computing in Knowledge-Based Systems*, Kowalik, J. S., editor, North-Hollad, 1987, pp.123-136.
- [5] Intellicorp's KEE version 3.0 manuals, 1986.
- [6] Kanal, L. N., "Interactive Pattern Analysis and Classification Systems: A Survey and Commentary," *Proc. IEEE*, vol.60, Oct. 1972, pp.1200-1215.
- [7] Kitzmiller, C. T., Kowalik, J. S., "Coupling Symbolic and Numeric Computing in Knowledge-Based Systems," *AI Magazine*, Summer, 1987, pp.85-90.
- [8] Lusk, E. L. and Overbeek, R. A., "Automated Reasoning and Knowledge Based Design in the Scientific Programming Environment," *Problem Solving Environment for Scientific Computing*, Ford and Chatelin, eds, Elsevier Science Publishing Co., 1987, pp.83-98.
- [9] Machura, M., "Issues in the Design of Problem Solving Environments," *Problem Solving Environment for Scientific Computing*, Ford and Chatelin, eds, Elsevier Science Publishing Co., 1987, pp.263-280.
- [10] Tompkins, J. W., "Using an Object-Oriented Environment to Enable Expert System to Deal with Existing Programs," *Coupling Symbolic and Numeric Computing in Knowledge-Based Systems*, Kowalik, J. S., editor, North-Holland, 1987, pp.161-168.
- [11] Young, T. Y. and Fu, K. S., eds., *Handbook of Pattern Recognition and Image Processing*, Academic Press, 1986.



An Architecture for Rule Based System Explanation

T. R. Fennel
J. D. Johannes
University of Alabama in Huntsville
Huntsville, AL 35899

Abstract

This paper presents a system architecture which incorporate both graphics and text into explanations provided by rule based expert systems. This architecture facilitates explanation of the knowledge base content, the control strategies employed by the system, and the conclusions made by the system. The suggested approach combines hypermedia and inference engine capabilities. Advantages include: closer integration of user interface, explanation system, and knowledge base; the ability to embed links to deeper knowledge underlying the compiled knowledge used in the knowledge base; and allowing for more direct control of explanation depth and duration by the user. User models are suggested to control the type, amount, and order of information presented.

Introduction

One of the earliest claims of expert system developers was that the resulting systems could "explain" their actions. These claims were often effectively backed up by the textual presentation of traces of rule firings which could explain "how" the system had made a decision.[Pople, '77] Additionally, systems could answer "why" the system was asking for information by presenting as explanation an English text description of the rule which required the information [Clancey, '83]. However, complete explanation requires addressing the problems of what, how, when and to whom knowledge is to be communicated.

[Wick and Slagle, '89] suggest that explanation capabilities could be greatly enhanced by the introduction of supplementary knowledge and by allowing variations of queries over time. For example, the user could ask not only "Why do you want to know this now?", but could also ask "Why would you ever ask me for this information?". Similarly the user could ask not only "How did you know?" , but also "How could you find out?". To answer these questions the system must keep extended histories, or traces, of actions taken by the expert system and based on dependencies be able to generate responses of a forward looking nature.

[Chandrasekaran, Tanner, and Josephson, '89] emphasize that explanation should be provided not only at the low levels (exemplified by presenting the conditions associated with a single specific rule) but that high-level explanation of overall system goals should also be available. Their suggestions are supported by work on automatic generation of textual explanations through specialized grammars [Bridges and Johannes, '89] . An underlying truth here is that humans tend to be much better at explaining their actions because they are able to convey both their abstract goals and detailed information -- but with the significance of the details "slanted" towards satisfying the stated goals. Therefore, the grammar used by humans during explanation goes beyond that used for simply explaining system details.

It has been suggested that much of the difficulty in developing expert systems is that the early recorded sessions provide better explanation knowledge than actual problem solving knowledge. To obtain better explanations, the overall system must be designed and developed with explanation as an integral part in all project stages. Particularly for rule based systems we advocate working with a representation of "IF, THEN, BECAUSE" rules as opposed

to "IF, THEN" rules. By giving early knowledge acquisition sessions very heavy weight, useful explanation knowledge can be gathered. Subsequently, this explanation knowledge can be used for verification of solutions provided by the expert system.

A final point needs to be made in this introduction. There is a difference between "justification" and "explanation". For justification we must provide a formal proof of completeness. For explanation, we must focus on and explain only the part of a solution that was not understood. [Clancey, '83]

An Architecture for Explanation

Figure 1 presents a knowledge based system architecture which emphasizes explanation capabilities on an equal par with capabilities provided to solve the original problem. The three main components of the architecture are an expert system intended to solve the "original" problem, an explanation system, and a user interface which acts as a bridge between the two. We will now discuss these three major components and suggest finer grained architectures for each.

The Expert System

The expert system architecture presents a bottom-up separation of four main layers for conventional data structures, object definitions, flat domain knowledge rules, and meta (or control) rules. This is the system proposed to solve the original task. Imposing the defined structure aids in system development, delivery and maintenance. For example, it aids in development (particularly for larger systems) by allowing for clear delineation of tasks among multiple members of a development team and by making interfaces explicit. The imposed structure aids in delivery by forcing incorporation of conventional database technology (and therefore exploits existing databases) at the lowest level and by separating the interface layer for early attention. By including the general architecture in the support documentation, maintenance tasks requiring different skills can be anticipated.

The lowest level of the expert system represents an underlying database with basic facts about the problem or about the current state of the world as the knowledge base knows it. At the next highest level, an object hierarchy is provided and the object definitions are all linked to conceptual definitions. The values for object attributes are updated through a link with the underlying database. A common mistake in the past has been to assume that the object layer was the lowest layer required for expert systems. This mistake delayed the integration of many systems with conventional databases and often resulted in significant efforts to rewrite the system or duplication of data. The third layer represents a relatively flat body of rules which typically represent a hierarchy of symptoms or constraints and are the result of knowledge engineering. Strategies for applying the constraint rules are represented in the fourth level by another hierarchy of meta-rules (these typically control the inferencing process at a very high level and depend on the built in mechanisms of forward or backward propagation for results at the constraint level).

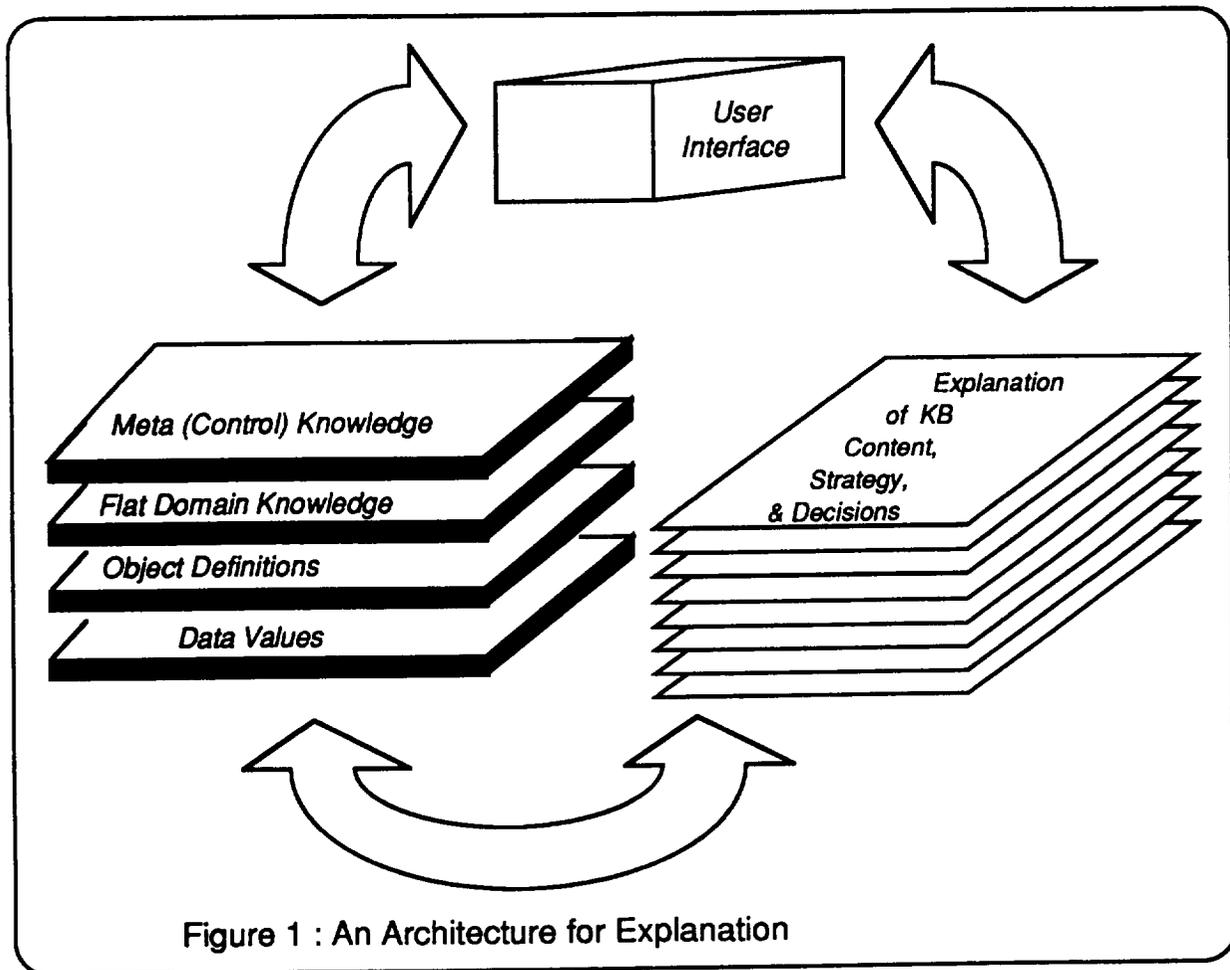


Figure 1 : An Architecture for Explanation

It turns out that imposing a structure upon the expert system can help provide structure to explanations. For example, more sophisticated implementations can use this imposed structure along with knowledge about each layer and the user to guide planning for explanation construction.

The Explanation System

The second major component of the architecture is the explanation system. Although our architecture would facilitate such an implementation, we do not insist that the portion of the system which provides explanation be a "complete" separate expert system. That is, we do not insist that the explanation system be capable of solving the original problem, as does [Wick and Thompson, '89].

At the lower levels in the architecture, the explanation system layers correspond to those in the expert system. For example, the bottom layer consists primarily of simple information about the data values layer of the expert system and contains typical information typical of that found in a "data dictionary", such as data value types, ranges, sources, etc. The next layer corresponds to the object definition layer in the expert system. It contains information about the objects and knowledge about fundamental mechanisms such as inheritance through the class structures of the implementation system. At the next two levels (the flat domain knowledge and control knowledge layers) the explanation system can contain even more information than the corresponding expert system

layers. The internal architecture for these layers is dependent upon the depth of background material included, the representation used for depicting strategy, and the extent of any user models.

To understand how the architecture may change it is important to specify the type of explanation is to be supported. In the following three sections we present approaches working within the depicted architecture for explaining knowledge base content, strategies, and decisions. [Chandrasekarn, et al, '89] provide details regarding this three pronged approach for explanation from introspection of knowledge and inference.

Explaining Knowledge Base Content

The architecture presented provides for explanation of knowledge base content at all levels. Starting with the lowest level, an underlying database represents basic facts about the problem or about the current state of the world as the knowledge base knows it. Explanation content for the database is typical in that it should provide information on the data sources, last update, units of measure, and validity intervals.

At the next highest level, an object hierarchy is provided and the object definitions are all linked to conceptual definitions. Graphics depicting component and subcomponent details are used where appropriate. Information provided about each object class include its importance in the problem to be solved and how it is used in the problem solving process. Each object attribute is similarly treated with the addition that each object attribute is also flagged to indicate whether its value is simply read in from the database or can be changed by the problem dynamics. The idea of assigning values of LABDATA to data that typically requires no explanation other than source was suggested in [Davis and Buchanan, '77]. Where attributes can have multiple values, the meaning of the multiple values is explained, along with expected consequences on the problem solving process.

The constraint rules form the third level of the knowledge base and serve to emphasize that in a rule based system oriented towards explanation the rules themselves should be thought of as objects. For explanation of content, one successful implementation uses an "index" that graphically shows the constraint hierarchy as composed of only keyword phrases. Additionally, each rule should be captured in hypertext form, so that the user can select any rule from the keyword hierarchy, then any part of the rule can be selected to explain the contents in more detail. Rule attributes include static English text which restates the rule, the rule originator, last update, a list of pointers to any related "cases" or "tests" from which the rule was derived, the relation to other rules, an understandable English text prompt used in conjunction with the rule when requesting information, and a graphical representation of the rule where possible. For systems which use confidence factors, it is imperative to note that the confidence factors themselves convey knowledge that should be explained.[Davis and Buchanan, '77] A confidence factor of one indicates that a "shallow" explanation may suffice since the rule is most likely definitional in nature, while confidence factors not equal to one represent the application of judgement and the relevant ranking of its importance and therefore requires more explanation.

Explaining the Knowledge Based System Strategy

Explaining a strategy involves in part the explaining of a perspective for relating rules hierarchically and then showing how these relations provide leverage for managing a large amount of data or number of hypotheses. The meta-rules at the fourth level of the expert system form the core of solution strategy and can also be represented in the explanation system by a graphic hierarchy. At this level the source for the rules becomes critical as these are the rules which control the order for checking the constraints at the next lower level. These rules explicitly determine which constraints are checked under varying circumstances. The strategies implemented intentionally mimic those used by experts from various areas within the domain and are one of the areas where having multiple

experts can be an advantage and where explaining the source of a rule can make a significant difference in acceptance of the final system.

It is clear that strategy and structure are intimately related in the expert system. The explanation system should make this as explicit as possible. For example, where screening clauses are used in the system for internal control as part of a rule, the rationale behind their placement should be documented and available for explanation.

Explaining Knowledge Based System Decisions

The ideal situation when explaining decisions is to employ any material a person may use, the point being to represent the "bottom line" as clearly as possible. For example, the rule hierarchies presented to explain the knowledge base content can be enhanced by highlighting information (the computers equivalent of pointing) used in the decision process. Where graphical keyword "indices" are available, they can be used to highlight a single keyword representing a rule or group of rules while presenting the constraint hierarchy. This will often serve as sufficient explanation for domain experts, while hypermedia links from the keyword hierarchy provide the "back pocket" type of information needed for explanation to other audiences.

Following the example set by [Brown, Burton, and de Kleer, '82], system developers should attempt to anticipate what are most likely to be the more difficult areas involved in making the decisions and provide even more depth and tutorial information for explanation of decisions in some areas. The areas can be highlighted by assigning individual measures of complexity or importance to individual rules and viewing the hierarchy of rules with aggregate weighting above a threshold value.

Even artificially constructed "trees" representing the HOW information for decision explanations can be very useful. That is, the tree presented as explanation need not reflect the reasoning process in its entirety (Wick and Thompson's work argues that it may not reflect it at all). However, we feel that it should demonstrate at least a "feeling" for the structure of the problem space and the nature of the search strategy used.

One of the more difficult tasks may be tying explanations to a general abstract level or task (such as in Dr. C's work), especially for strategy rules. Developers of explanation systems must realize that any rule, no matter how obvious or clear, is only a single step in the explanation. Other steps include setting strategy context (such as generic task identification), focusing on state information (particular values of object attributes at a point in time), and elucidation of outcome.

The User Interface

The user interface is the third major component of our suggested architecture and spans the gap between the expert system and the explanation system. In the past, most expert systems have typically relied on simple menu driven interfaces and textual presentation of explanations. Notable exceptions to this include the STEAMER [Hollan, Hutchins, and Weitzman, '84] system which used an underlying simulation model with incorporated graphics and the General Electric DELTA expert system for diagnosing diesel electric locomotive failures which incorporated video storage as part of the system [Bonissone and Johnson, '83]. More recent work by [Sue, '89] has focused on a mixture of text and graphics in the explanations.

We suggest that the interface should be some implementation of hypermedia. The recent emergence of robust hypermedia systems argues favorably for the integration of graphics and text. An ancient Chinese proverb states "It is better to see a thing once than to read about it one hundred times." The wisdom of this statement has been proven repeatedly by people who while trying to explain their actions to others resort to the use of a graphic as part of their clarification [Berry and Broadbent, '87]. Therefore, perhaps the best rationale for incorporating graphics and text is simply to mimic reliance upon them as

humans do. By using figures which have been scanned in, and then adding "buttons" or links to additional information and text fields we can allow for perusal of a tremendous amount of information at a level dynamically controlled by the user. It is important to realize that the links created for explanations tend to be more specific than those created for a purely informative stack -- at least at the beginning of the explanation. However, as the user traverses links away from the starting point the bounds on what type of information is presented is left up to the system developers (for example, it may be desirable to restrict the amount of autonomy afforded to students where the explanation system also serves as part of an intelligent tutoring system).

Modern portable computers, optical discs, and graphics software make it possible to quickly and easily capture and integrate graphic material. The architecture suggested combines database, hypermedia and inference engine capabilities. These capabilities are readily available on conventional PC hardware and recent announcements in the area of integration across diverse packages makes it practical to expect easier access to such tools.

User Models and Explanation in Intelligent Tutoring Systems

An additional level of complexity is added to the problem of explanation when we introduce the need for models of the user so that the information presented will be both understandable and timely. Related work [Wenger, '87] in the rapidly expanding field of intelligent tutoring systems demonstrates repeatedly that it is the communication of knowledge (not just data) that is important and that the presenter of knowledge must make allowances for student abilities. Most explanations are presented to a single individual, or at least to a group with focused attention in a common setting. For example an expert system developed as an engineering aid may be used repeatedly by individual engineers who are experts in the domain. However; when explaining the actions of the system (which have led to specific decisions) during a formal review, the experts must be able to integrate background information, current focused information, and their overall goals into explanations at a level their audience will understand. The point is that the same explanations given by the system to the expert during its normal use will not suffice as explanations given to a broader audience. The task of trying to model even the typical user (in an effort to know what to present and how to present it) is often not straightforward.

We would like to continue to investigate the use of expert systems as intelligent tutors. Conceptual definitions of objects and rule hierarchies are used extensively in explanations, and serve as excellent starting places for those using the system as a tutor. These hierarchies can be used for quickly identifying areas of interest to different users and for providing a type of dialogue from which students can ask for more detailed explanations [Moore and Swartout, '89].

Any good explanation must "make contact" with previously known concepts. It's a good idea to include a core of short tutorials on the fundamental concepts in an explanation system. These provide the needed basics upon which everyone can view the explanations. They also provide a good example of what should not be included in the system used to solve the problem, but should be included in the explanation system.

Capturing the "Link" between Compiled and Deep Knowledge

The deliberate separation of the explanation system component from the expert system highlights the fact that knowledge required for explanation often lies outside of the knowledge incorporated into the expert system designed to solve the original problem. One basic reason for this is that the expert system represents compiled knowledge. The idea here is similar to that found in conventional compiled languages where efficiency is gained by stripping away non-executable code such as comments. In several ways, explanation knowledge is very similar to the knowledge often gathered into documentation for conventional programs. Just as conventional program documentation has many levels, the explanation system is a knowledge based system with multiple layers (although not always true in practice, there should be major differences between a user guide and detailed programmers guide to the same software).

It is the high level and abstract knowledge (such as originally intended use, goals, or even current events such as budgetary constraints) that is often compiled out of the final version of a knowledge base. As a result, explanations associated with expert system will most likely be later questioned regarding completeness, accuracy, or accountability -- and the true explanations may not be available. We've found that the most difficult part of this is indeed deciding how to tie explanations to the higher level goals. In many cases we recommend simple English text statements as they seem most appropriate. The more abstract problem solving goals (such as the strategy rules) are depicted using process flow diagrams. A fairly simple mapping allows for capturing the link between the strategy rules and the rules at lower layers.

Future Directions

It has been suggested [Brown, Burton, and de Kleer, '82] that links to conceptually faithful simulations can provide for a form of continuous explanations and could thereby represent a deeper knowledge of the domain. We would like to pursue this area by providing links from the hypermedia interface to an application written for simulating processes in the domain.

Construction of an appropriate grammar for describing the relationships among objects and rules within the domain and specialized for use in explanations is being considered for future research. The grammar definition would help ensure future applications would find the embodied knowledge in machine intelligible formats and could be used to limit the scope of explanations which must be generated. It has been suggested [Paris, '88] that explanation for expert systems provides a rich domain in which to study natural language generation. The simple architecture presented in this paper would be refined to accommodate a more intelligent architecture for the explanations system.

Summary

An architecture has been suggested for combining an expert system, explanation system and hypermedia based user interface. Components of explanation include explaining knowledge base content, strategy, and decisions. By emphasizing explanation as a major system goal which requires knowledge and effort aside from solving the original problem, the systems can benefit : by being more readily received in the end user environment; by also serving as a beginning platform for instruction; by providing links to the deeper knowledge underlying that which would normally be compiled out of the knowledge base; and by providing for smoother integration of interface, knowledge base, and data which helps ensure they will continue to be used.

References

- Abu-Hakima, S., A Generic Environment for Constructing Diagnostic Hierarchies, in *Proceedings of the IJCAI-89 Workshop on Integrated Human-Machine Intelligence in Aerospace Systems*, pp. 90-99, 1989.
- Berry, D.C., and Broadbent, D.E., Expert Systems and the man-machine interface. Part Two: The user interface, *Expert Systems*, Vol. 4, No. 1, February 1987.
- Bonissone, P.P. and Johnson, H.E., Expert system for diesel electric locomotive repair, *Knowledge-based Systems Report*, General Electric Co., Schenectady, N.Y., 1983.
- Bridges, S., and Johannes, J.D., Integration of Knowledge Sources for Explanation Production, *1989 ACM*, 1989.
- Brown, J.S.; Burton, R.R.; and de Kleer, J., Peagogical, Natural Language and Knowledge Engineering Techniques in Sophie I, II, and III, in *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, eds., Academic Press, London, UK, pp 227-282, 1982.
- Chandrasekaran, B.; Tanner, M.C.; and Josephson, J.R., Explaining Control Strategies in Problem Solving, *IEEE Expert*, pp. 9-24, Spring 1989.
- Clancey, W.J., The Epistemology of a Rule-Based Expert system -- A Framework for Explanation, *Artificial Intelligence*, pp. 215-251, May 1983.
- Davis, R. and Buchanan, B., Production Rules as a Representation for a Knowledge-based Consultation Program, *Readings in Knowledge Representation*, Brachman and Levesque eds, 1977.
- Hollan, J.D.; Hutchins, E.L.; and Weitzman, L. STEAMER: an interactive inspectable simulation-based training system, *AI Magazine*, vol. 5, no. 2, pp. 15-27, 1984.
- Moore, J.D., and Swartout, W. R., A Reactive Approach to Explanation, presented at the AAAI Workshop on Explanations, 1988.
- Paris, C.L., Generation and Explanation: Building an explanation facility for the Explainable Expert Systems framework, submitted to the 1988 Workshop on Text Generation, 1988.
- Pople, H.E., The Formation of Composite Hypotheses in Diagnostic Problem Solving, in *Proc. Fifth IJCAI*, Morgan Kaufmann, Los Altos, Calif., pp. 1030-1037, 1977.
- Weld, D.S. , Explaining complex engineered devices. *BBN Report 5489*. Bolt Beranek and Newman Inc., Cambridge, Massachusetts, 1983.
- Wenger, E., *Artificial Intelligence and Tutoring Systems* , Morgan Kaufmann Publishers, Inc., 1987.
- Wick, M.R. and Slagle, J.R., An Explanation Facility for Today's Expert Systems, *IEEE Expert*, pp. 26-36, Spring 1989.
- Wick and Thompson, Reconstructive Explanation: Explanation as Complex Problem Solving, in *Proc. Eleventh IJCAI*, Morgan Kaufmann, Los Altos, Calif., pp. 135-147, 1989.

An Automated Tool for the Design and Assessment of Space Systems

Lois M.L. Delcambre and Steve P. Landry
The Center for Advanced Computer Studies
University of Southwestern Louisiana
P.O. Box 44330
Lafayette LA 70504-4330
(318) 231-5697 (318) 231-6768
lmd@usl.usl.edu spl@usl.usl.edu

Abstract

Space systems can be characterized as both large and complex but they often rely on reusable subcomponents. One problem in the design of such systems is the representation and validation of the system, particularly at the higher levels of management. This paper describes an automated tool for the representation, refinement, and validation of such complex systems based on a formal design theory, the Theory of Plausible Design. In particular, this paper describes the steps necessary to automate the tool and make it a competent, usable assistant.

1. Introduction

The process of design relies heavily on human creativity and judgement. Design is particularly difficult when the artifact being designed is large and complex, such as with satellites, computer systems, etc. At the highest level, the design, development and evolution of such complex systems must be managed through appropriate validation and assessment techniques. Although numerous tools exist to aid in the detailed design and development of individual components through CAD technology, there is little automated support for management and development of complex systems at the higher levels.

The design and development of artifacts can be viewed as the process of satisfying a set of constraints or requirements. Constraints are satisfied either by detailed elaboration into subconstraints and/or by providing evidence that the constraint is satisfied. As an example, a new satellite program begins with the specification of the scientific aspects of the satellite, termed the *mission requirements*. The mission requirements must be refined so that the end-product will meet the scientific objectives. Thus, if the mission requirements demand a sensor with a particular sensitivity, the designer (or program manager) may choose an off-the-shelf component, with a track record of high sensitivity. The documented track record then provides *evidence* that the sensor sensitivity requirement has been met. However, there may be other constraints that interfere with the choice of the sensor. The selected sensor may require too much power, may be too large or heavy for the satellite, or may be so delicate that it will not survive the intended launch procedure. The fundamental problem in the development of complex systems is the representation, refinement, evolution, and assessment, and ultimately the validation of a myriad of diverse constraints.

This paper addresses this problem by describing an automated tool that relies on the Theory of Plausible Design, TPD [D89]. A plausible design, at any stage in the design process, is represented by a set of constraints with associated plausibility states organized into a directed, acyclic Constraint Dependency Graph (CDG). The

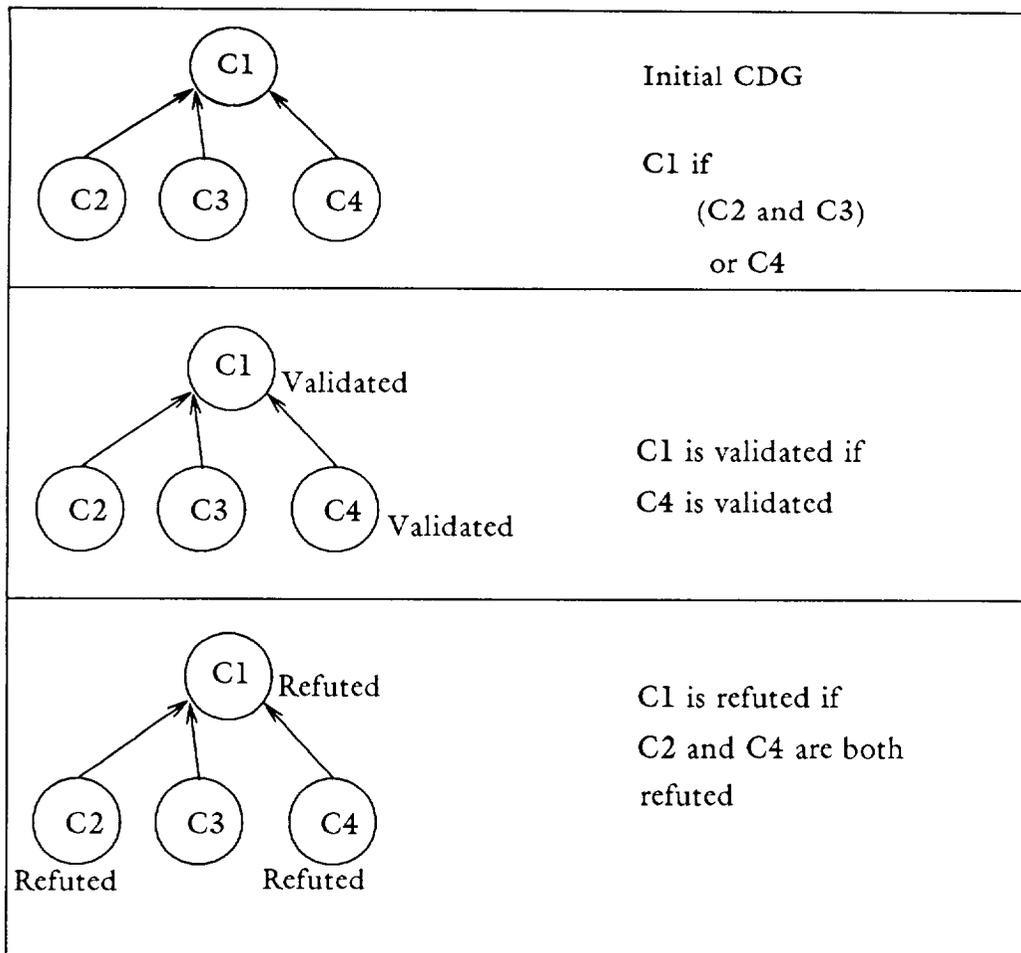
plausibility state takes the following values: unknown, assumed, validated, or refuted. The final two states, validated and refuted, are established through the presentation of evidence for or against the constraint, respectively. The CDG represents the successive refinement of constraints into subconstraints and supports the automatic computation of the plausibility state for each constraint.

The contribution of this paper is the description of how TPD can be used as the formal framework for an automated tool to manage, develop, refine, assess, and validate complex systems. TPD provides the framework for the tool through the CDG with automatic propagation of plausibility states. The remainder of this paper is organized as follows. TPD is presented in Section 2 along with several examples. The challenge is first to implement the CDG and plausibility state propagation and then to provide appropriate interfaces and structure to make the tool easy to use and to facilitate the development of correct systems. The major components of the automated tool are presented in Section 3. The paper concludes with a discussion of the work in progress, in Section 4.

2. The Theory of Plausible Design

The Theory of Plausible Design represents a design as a set of plausibility statements. Each plausibility statement contains a representation of the actual constraint, expressed in English or other, perhaps formal, language. The plausibility statement also includes the plausibility state for the constraint. When a constraint is first formulated, the state is initialized to *unknown*. At this point, the designer may provide direct evidence that the constraint can be satisfied. On the other hand, the designer may develop one or more alternative refinements such that if the refinements can be satisfied then the original constraint will be satisfied. When a constraint is refined into one or more subconstraints, the plausibility state of the original constraint depends on the plausibility state of subconstraints. The details of how a given constraint is related to the alternatives described in its subconstraints is given by a well-formed formula connecting the subconstraints with *and* and *or*. Eventually, every constraint at the lowest level of detail (i.e. a constraint with no subconstraints) must be validated by the presentation of evidence. Evidence can be quite precise (e.g. a proof), heuristic (e.g. expert opinion), or experimental (e.g. results from simulation). One strength of TPD is the automatic propagation of the plausibility state from the lowest level upwards to all affected constraints. The propagation of plausibility state enforces the semantics associated with the well-formed formula of the non-leaf constraints. In addition to effectively capturing the current design state, TPD provides a paradigm that captures the design history and integrates various forms of evidence to validate constraints.

One important and equivalent view of the design is the representation of the plausibility statements (or constraints) with the Constraint Dependency Graph (CDG). In this view, the nodes of the graph represent the constraint, the plausibility state of the constraint, and the evidence that either refutes or validates the constraint. The arcs of the CDG represent the subconstraint relationship and includes a graphical representation of the *and* and *or* relationship. As an example, consider the CDG presented in Figure 2.1. The upward arcs that meet at a single point indicate that the subconstraints must all be validated in order to validate the constraint (i.e. the *and* relationship). Multiple sets of inward arcs indicate the *or* relationship.



A Sample Constraint Dependency Graph

Figure 2.1

At the lowest level of the CDG, individual constraints are validated or refuted through the presentation of evidence in favor or against the constraint, respectively. Once the plausibility state of constraints at the lowest level is either validated or refuted, then the state is propagated to the next higher level according to the appropriate connectives (*and* or *or*) implied by the CDG. For example in Figure 2.1, constraint C1 may be satisfied if C2 and C3 are satisfied or if C4 is satisfied. If evidence is presented for C4 to be validated, then the plausibility state propagation would then record C1 as validated. Similarly, if evidence were presented to refute both C2 and C4, then the plausibility state of C1 would also be refuted. These three scenarios are shown in Figure 2.1.

A significant problem faced by the program manager is that of assessment. Note that the process of design happens over time and that the assessment function relies primarily on the performance of human experts. The primary objective of the review team is to certify that the current design representation meets the original constraints. Among the major obstacles that the review team must overcome are the lack of direct connections from the current components of the design to the original constraints and

the information that describes the interactions among current designed components. TPD and the resulting design paradigm provides solutions to both of these obstacles by capturing a full history of the design and maintaining the interactions between components of the design at all phases.

3. The Automated Design and Assessment Tool

TPD provides a powerful framework to manage the design of complex systems. However, the automation of TPD to serve as a competent assistant (e.g. to the program manager) presents a number of challenges. The specific components of the tool that address these challenges are presented here. First, the components intended to support TPD directly are presented in Section 3.1. Then the opportunities for using application-specific structure in the form of a semantic network are presented in Section 3.2. Finally, the use of previous, plausible designs as a knowledge-base to guide and to facilitate the design process is discussed in Section 3.3.

3.1. Automating the Theory of Plausible Designs

TPD is intended to capture the design process and the design rationale through the constraints represented in the CDG. The TPD tool must interface (upward) to the designer and (downward) to other automated design/development systems. The user interface is a graphical interface with support for easy creation and manipulation of the CDG and a variety of abstraction mechanisms. The CDG is displayed graphically with the plausibility state at each node clearly highlighted. The design/refinement/validation process is driven by the current state of the CDG and thus the work remaining to complete the design is easily presented to the designer.

During the development of a satellite or other complex system, the program manager may develop the system specification (from the mission requirement) and then rely on other development groups (or subcontractors) to develop the satellite, the launch vehicle, and the ground system, for example. In this case, the current state of the CDG represents the specification of the entire system under design. The (downward) interface provides the appropriate constraints from the lowest level of the CDG as the formal specification for the detailed design and development of the subcomponent to be communicated to the subcontractors. Then, after the subcomponent is designed or implemented (e.g. in another automated system), standard practices of testing, simulation, etc. can be used to successfully validate that the completed design (or system) satisfies the associated leaf constraints. This evidence is then recorded in the CDG to complete the design.

The emerging design maintained by the TPD tool is valuable in as much as it accurately reflects the subtle interaction among constraints. Said another way, the benefits of TPD and the confidence in a validated plausible design require that all of the appropriate connections among constraints be explicitly recorded in the CDG. The tool includes a synonym facility that can make suggestions concerning potentially relevant constraints. As an example, the constraint concerning the sensitivity of the sensor ultimately affects the power system, the control system, and the size and weight of the satellite. These connections can be suggested by the tool based on the recognition of synonyms present in the statement of the constraint and subconstraints associated with the actual sensor selected. The final responsibility for a correct CDG ultimately rests with the designer. The synonym facility just tries to help.

3.2. Application-Specific Structure

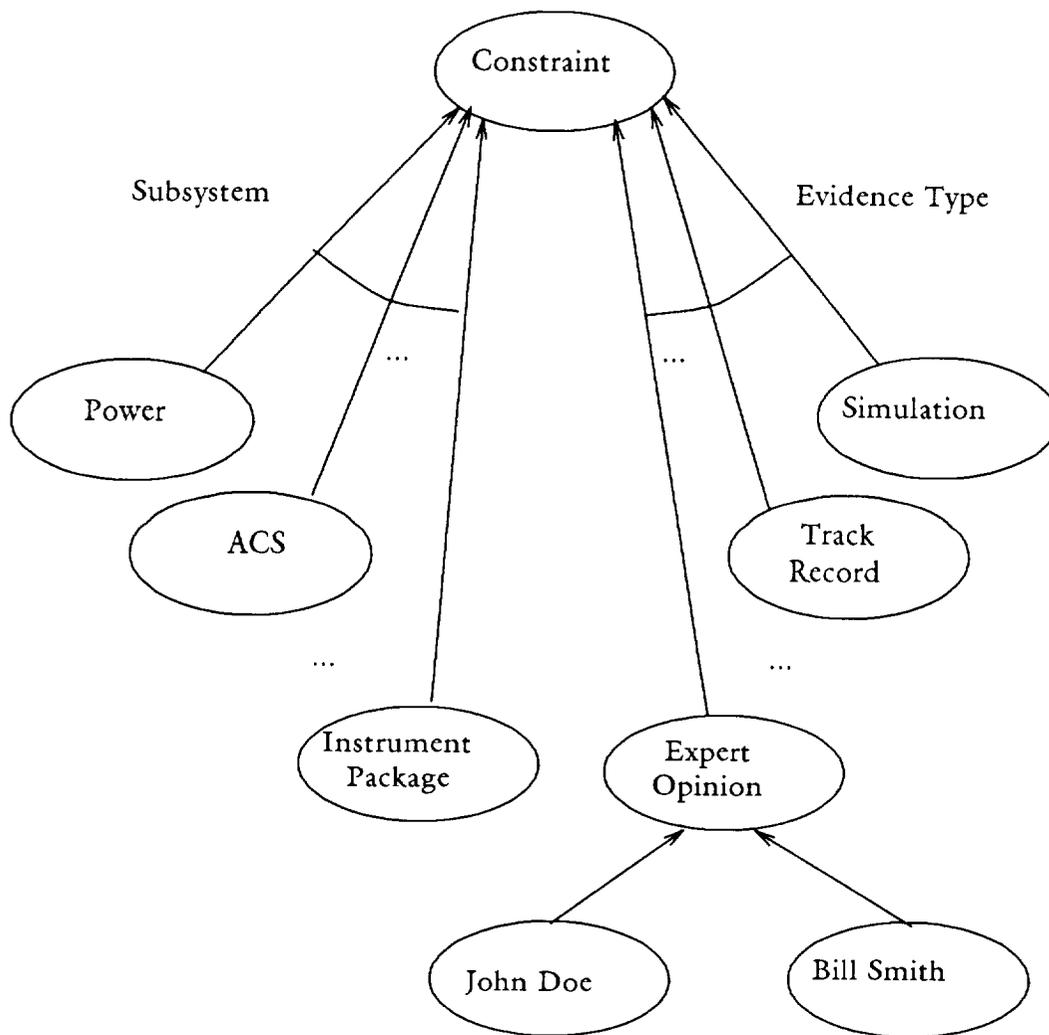
In general, a single CDG represents the system at any particular state in the design. The CDG may be arbitrarily deep, depending on the level of refinement of the various constraints. The CDG may also be arbitrarily large, depending on the complexity of the system. The CDG is built according to the refinement and interrelationship among constraints. As in the example concerning the sensor sensitivity, a single constraint may influence many other components of the system. Thus the CDG can include arbitrarily complex interconnections. The challenge then is to manage this complexity through the use of abstraction. The problem is: how can the tool help the designer focus his attention?

When the TPD tool is used for a particular application, additional structure can be used to facilitate the development process. The structure is introduced at a level above the CDG through the use of a semantic network. At the meta level, the nodes of the CDG can be grouped in a variety of ways. As an example, consider the semantic network in Figure 3.1. The individual constraints are all members of the most general class, Constraint (at the top of the semantic network). Additionally, the constraints can be placed in any or all of the subclasses. One way to group constraints is according to the subsystem that they belong to. For a satellite, the subsystem structure is shown on the left side of Figure 3.1 with three subclasses shown for Power, ACS (attitude control subsystem), and the Instrument Package. This set of subclasses is marked as being a "Subsystem" classification by the labeled arc. Similarly, the constraints can be classified according to evidence type, as shown on the right side of the figure. A given constraint could then be entered into the appropriate subsystem class and also into the appropriate evidence type class. If the evidence type is "expert opinion", then the constraint can also be entered into the subclass according to the actual expert. The structure shown in Figure 3.1 can be used to highlight the CDG according to the needs of the user. For example, if the user wants to work on the power subsystem, the user can request that only the constraints that are in the Power subclass in Figure 3.1 be shown. The remaining constraints in the CDG can then be made invisible. Similarly, the user may wish to highlight all constraints that were validated by Bill Smith. Then the constraints with an evidence type of "Expert Opinion" that were validated by "Bill Smith" could be highlighted (perhaps in color). The application structure can be subdivided (or classified) in many other ways. The strength of this approach is that the additional structure can be easily introduced at the meta level as shown in Figure 3.1 and then can be used to drive the user interface.

The structure of the CDG and also of the meta level suggests that a database style query language can be used to navigate around these structures. Within a CDG, a query language can be used to locate all subconstraints, all superconstraints, all assumed constraints, etc.

3.3. The Design History

If the TPD tool is used to develop multiple satellite systems, for example, then the design history serves as a valuable knowledge base. In particular, the previous designs can serve as a library for reusable subcomponents and also as a rich source of suggestions to solve design problems. For the first part, any subcomponent for which there exists a plausible design (e.g. a detailed design for the sensor), can be incorporated into the current design. This process is facilitated both by the application-



Application-Specific Meta Level Structure
Figure 3.1

specific structure and by the synonym facility. Each constraint in the CDG of the subcomponent can be labeled with the appropriate keywords (e.g. to indicate the appropriate subsystem of the satellite). Then, when the CDG for the subcomponent is merged with the CDG of the design in progress, the synonym facility can suggest the appropriate connections and can also populate the meta level classes.

Another powerful use of the history database is for access to lessons learned. A previous mission may have encountered a similar design problem and may have solved it. By the appropriate pattern matching process, the current CDG can be matched against portions of the CDG from previous designs. The evidence used to validate the previous design can then be suggested as a way to validate the current design. Note that the pattern matching process required to access the history is very complex, in general. However, within a specific domain with well-defined subsystems and components, a useful pattern-matching facility can be implemented. Once the appropriate portions of the history are located, then the CDG and the evidence used to validate or

refute previous design can be used to drive the current design and suggest possible testing procedures.

Finally, the tool is part of a knowledge-based framework that supports various design methodologies. The choice of what to do next can be viewed as the choice of which part of the CDG to concentrate on. The user may choose a top-down or bottom-up methodology or some combination. The user may be driven according to the constraints that currently have a "refuted" plausibility state. The criteria used to guide the design can be easily recorded in rules that act on the current state of the CDG. The use of the TPD tool to support design methodologies is described in more detail in [DL90].

4. Work in Progress

Plausibility theory has been formally defined and a number of case studies have been developed. However, so far all such plausible designs have been developed manually. The use of TPD as the basis for an automated tool is currently being investigated in conjunction with Orbital Systems, Ltd. for an Air Force project [O90]. The strength of TPD stems from the automatic maintenance of plausibility states. When the CDG is correctly represented, then the presentation of evidence to validate (or refute) subconstraints is automatically reflected in the entire CDG. *All* affected constraints are immediately identified. This paper describes the steps required to make the tool usable. The design and development of the tool is currently underway. The work with the Air Force includes the collaboration of a variety of space system customers through the clientele of Orbital Systems, Ltd.

References

- [D89] Dasgupta, Subrata, "The Structure of Design Processes", *Advances in Computers*, Vol. 28, 1989.
- [DL90] Delcambre, L.M.L., Landry, S.P., and Dasgupta S., "A Knowledge-Based Framework for an Integrated Engineering Design System", 1990 Eastern Multi-Conference, April 1990.
- [O89] Orbital Systems, Ltd., "Reduction of Space Costs Using Expert System Technology", SBIR Phase II Grant Proposal, Air Force, Mar. 1989.

ATS Displays - A Reasoning Visualization Tool for Expert Systems

William John Selig

Magnetospheric Physics Branch
NASA/Marshall Space Flight Center
Huntsville, Al 35812

James D. Johannes

Computer Science Department
University of Alabama in Huntsville
Huntsville, Al 35899

ABSTRACT

Reasoning visualization is a useful tool that can help users better understand the inherently non-sequential logic of an expert system. While this is desirable in most all expert system applications, it is especially so for such critical systems as those destined for space-based operations. A hierarchical view of the expert system reasoning process and some characteristics of these various levels is presented. Also presented are Abstract Time Slice displays, a tool to visualize the plethora of interrelated information available at the host inferencing language level of reasoning. The usefulness of this tool is illustrated with some examples from a prototype potable water expert system for possible use aboard Space Station Freedom.

Introduction

We interact with expert systems for a variety of reasons: to develop/debug them; to analyze them; to use them to obtain answers in their programmed area of expertise; to learn a tutored subject from them; or just to understand the underlying inferencing process. During all these uses, one can benefit from an understanding of the reasoning process of the expert system. While it can be argued that this understanding is useful for all expert system applications, it becomes increasingly important for systems in critical application environments, such as space-based systems.

There are two major impediments to obtaining this understanding. First, humans have a basic limitation on the number of concepts that can be maintained in immediate attention, the combination of short term memory and the processing done therein. This limit is the oft cited seven \pm two "chunks", where a chunk is some unit concept. Second, the information germane to acquiring this understanding is usually presented at too low a conceptual level. There is so much detail presented that one expends significant mental effort trying to combine the detailed information into a coherent "picture", a higher level conceptualization (7). The first limitation is innate. The second limitation arises because current information presentation methods present too much information at too detailed a level. If the information were to be presented at an appropriately higher conceptual level, the basic human cognitive limitation could be at least partially circumvented.

As an example, consider a set of real numbers related by the equation, $y = \cos(10 * \pi * x / 100) \exp(-x / 20)$, $x = 0, 99$, an exponentially decaying sinusoid. If this set of numbers is conveyed as the list in Figure 1,

00	1.00000	0.904673	0.732029	0.505911	0.253002
05	-3.40425e-08	-0.228926	-0.414205	-0.542300	-0.606420
10	-0.606531	-0.548712	-0.443998	-0.306851	-0.153453
15	5.63291e-09	0.138850	0.251228	0.328922	0.367813
20	0.367879	0.332811	0.269298	0.186114	0.0930739
25	-9.67717e-08	-0.0842171	-0.152378	-0.199501	-0.223090
30	-0.223130	-0.201860	-0.163338	-0.112884	-0.0564522
35	1.15318e-07	0.0510803	0.0924217	0.121004	0.135311
40	0.135335	0.122434	0.0990693	0.0684676	0.0342401
45	-1.04287e-07	-0.0309818	-0.0560566	-0.0733924	-0.0820701
50	-0.0820850	-0.0742601	-0.0600886	-0.0415277	-0.0207676
55	-3.78490e-08	0.0187914	0.0340001	0.0445147	0.0497780
60	0.0497871	0.0450410	0.0364456	0.0251878	0.0125962
65	-2.66556e-08	-0.0113976	-0.0206221	-0.0269996	-0.0301919
70	-0.0301974	-0.0273188	-0.0221053	-0.0152772	-0.00763999
75	1.40223e-09	0.00691299	0.0125079	0.0163761	0.0183123
80	0.0183156	0.0165697	0.0134075	0.00926608	0.00463387
85	8.10506e-09	-0.00419293	-0.00758645	-0.00993257	-0.0111070
90	-0.0111090	-0.0100500	-0.00813209	-0.00562015	-0.00281059
95	6.15401e-09	0.00254315	0.00460141	0.00602442	0.00673673

Figure 1

roughly equivalent to the textual traces available from most current expert system shells, one would be hard put to mentally determine the interrelationships of those numbers. If instead this data is presented in a graphical form as in Figure 2,

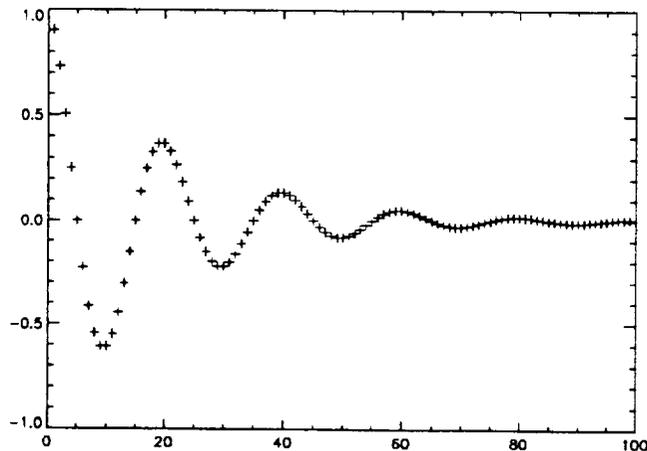


Figure 2

the interrelationships of the data values are immediately obvious. The detail of the exact individual values has been lost, but information on a higher conceptual level has been obtained, and with much less work on the part of the observer.

This is exactly the problem facing the individual trying to understand the reasoning processes of an expert system. Current environments present too much data at too low a level of detail for easy understanding of the processes involved. This problem in another form is the basis for the explanation systems/language generation systems field (4, 6). For similar reasons, in the realm of procedural programming, algorithm animation research is flourishing in order to illustrate the underlying processes of various sequential algorithms (3, 5, 11).

This paper focuses on the reasoning processes of expert systems and a tool for visualizing a subset of those processes. While there are various paradigms for expert system reasoning (1, 12), it was decided to initially focus this research effort on the forward-chaining rule-based paradigm. Per this focus, CLIPS, a readily available expert system shell, was chosen as the research vehicle.

Levels Of Reasoning

The reasoning of an expert system may be viewed on a continuum from that of the microcode of the hardware up through the programming language in which the host inferencing language is implemented on up to a "black-box" view of the application in which one sees only inputs and outputs. Pragmatically, the lowest level view which is of interest is that of the host inferencing language in which the expert system is implemented since this is the most primitive level at which the reasoning of the application may be specified.

Given an existing (or planned) expert system application, it can be viewed as having two distinct reasoning components. One is that of the application itself and is represented by the most abstract processes that define the reasoning of the expert system application. The result of the reasoning of this component consists of the inputs and outputs of the application, some subset of which are provided by and/or to the user. These inputs and outputs comprise the black-box view of the application. The other component is that of the inferencing language in which the application is (or will be) actually implemented. This is the part seen more by the expert system developer.

At the reasoning level of the host inferencing language, the reasoning is in terms of the primitives of that language. In most rule-based expert systems, these primitives are the facts and rules. The operations on these primitives are the assertion and retraction of facts, an initialization operation which asserts a set of predefined facts, the input and output of information, and the activation, deactivation and firing of rules to/from the agenda. While some rule-based expert system development languages (e.g. ART) also include the operation of existing rules defining new rules, for this research effort that operation is not considered.

The specifics of the application do not affect the actual reasoning primitives or the operations upon them, just the sequence order of their occurrence. Thus, at this reasoning level one can provide an ad hoc reasoning visualization. Users of this level would be system implementors desiring a gestalt of the low level reasoning in order to detect anomalous behavior and indications of application (in)efficiency.

At the highest level, that of the abstract reasoning process, the application reasoning is viewed in its most abstract form. This typically corresponds to metarules in more complex applications. At this level one is considering the application as conceived by the designer, not as it will be implemented.

Between these two endpoint levels are various mixtures of the two which are conceptually grouped into a realized application level. At this level the reasoning primitives are application oriented, even if they are associated with host inferencing primitive operations. One is interested in the concepts which rule firings represent, not merely that rules have fired. However, the visualization of the reasoning processes involved is in a form that is related to the inferencing language implementation of the application as opposed to the abstract reasoning processes, devoid of implementation considerations. This distinction between the abstract and realized application levels is similar to the conceptual versus implementation distinction made by Buchanan and Smith(2). For visualization of these

application based reasoning levels an ad hoc method is not possible. Instead, one must provide flexible facilities to allow users to construct their own custom visualizations(9).

The above discussion leads to the proposal of a hierarchy of reasoning levels. Conceptually, one can identify a continuous hierarchy of the reasoning process, based on a decreasing amount of detail, grouped into three ranges as shown in Figure 3. This hierarchy is viewed as a continuum since the abstract application level covers a range of detail as does the realized application level. One may also consider a range of detail at the host inferencing language level (e.g. all rules, some rules only, rule groupings). Note that this is but a more abstract view of our previous work on this subject(10).

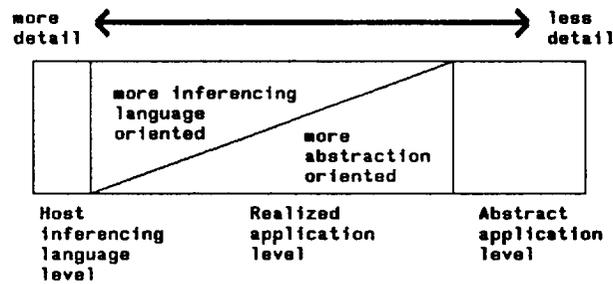


Figure 3

In focusing on the lower level primitive reasoning of an expert system, such as would be of interest during development/debugging and efficiency analysis, one is interested in the reasoning at the host inferencing language level. A model of a rule-based host inferencing language may be represented as the 4-tuple

$$M_{hil} = [F, R, A, O]$$

where

F = the set of facts, initially empty

R = the set of rules with antecedent patterns to be matched by facts $\in F$

A = the agenda; an ordered set of instantiations of rules $\in R$ matched by facts $\in F$; initially empty

O = the set of operations that modify F or A, or perform I/O

The operation set, O, consists of fact assertion, fact retraction, rule activation, rule deactivation, rule firing, input, output, and an initialization operation which causes predefined facts to be asserted.

Fact assertion and retraction are associated with the execution of the consequents of a rule instantiation that has been removed from the agenda for "firing". The initialization operation may be considered a pseudo rule firing in this respect since it also asserts facts. I/O is associated only with a rule firing.

Rule activation and deactivation are associated with changes to the set F. If a fact is added to F such that some rules $\in R$ are completely matched resulting in new instantiations of those rules, these instantiations are added to the set A. That is rule activation. If a

fact $\in F$ is retracted and the previous assertion of that fact had resulted in the addition of instantiations of some rules being added to the agenda, then any of those instantiations remaining on the agenda are removed. That is rule deactivation.

Of particular interest at this reasoning level is the traffic over time on both the fact list and the agenda, and the cause/effect interrelationships of that traffic. One would like to see a gestalt of the overall flow of the host inferencing language reasoning process and thus identify various aspects of operation, both normal and abnormal. One would like to be able to see sequences of rule group firings indicating phased rule operations along with the phasing control. Errant rule firing from inappropriate rule groups is also of interest. Also desirable would be the ability to identify excess activations and deactivations which may be caused by inefficient consequent sequencing. Fact and rule effectivity and agenda traffic density should also be easily observable. The information of interest at this reasoning level is similar to the information of interest to a person monitoring a wide area network system. The content of the actual data moving about the network does not matter. What is of concern is being able to tell that it is moving correctly and efficiently and that current and hopefully even potential problems can be identified easily.

Abstract Time Slice Displays

Abstract time slice (ATS) displays provide solution to the problem of the visualization of the host inferencing language reasoning level of a forward-chaining rule-based expert system. In providing a visualization of the above described reasoning there are two key issues, presentation at the appropriate level of detail and prevention of information overload even at that appropriate level. The appropriate level of detail is defined as that level at which the user immediately grasps the concept being presented. There should be little or no mental processing involved in assimilating the symbols. For presentation at the appropriate level of detail, ATS displays use unique symbols to represent the individual primitives and operations of the host inferencing language reasoning model. This obviates the need for the user to perform text to concept transformation. Instead the information is presented graphically. To prevent overload, ATS displays are static, thus providing support to the user's limited short term memory. They depict rules being activated and deactivated as the result of facts being either retracted or asserted (not respectively) as a result of other rule firings. They also show I/O as a result of rule firings. All of this information is displayed in an interrelated manner over time.

The program that generates ATS displays presently runs as a separate program taking as input two files, a segmented list of the rule names in the application and a file containing the full trace output from a run of the application. ATS outputs five files, the main display file and four adjunct files. The main display file is an ASCII file of PostScript code that creates the ATS displays on a laser printer. The adjunct files contain the details of the facts and I/O on both a time-slice and a sequential time-compressed basis. Since this capability is ad hoc and at the host inferencing language level, it could be integrated into CLIPS itself providing file output and/or direct graphical display. To understand the ATS displays, a few symbols must be defined. These are displayed and notated in Figure 4 while Figure 5 depicts various other information in the displays.

The causative symbol at a particular time slice indicates that some fact operation has caused either a rule activation or deactivation to happen (at some later time slice). A solid line connects the causative symbol to the effect caused.

The causing symbol indicates that some fact or I/O operation has been caused as a direct action of a fired rule's consequent. A solid line connects all of the actions caused by a particular rule firing, thus giving some indication of the scope and activity of a fired rule.

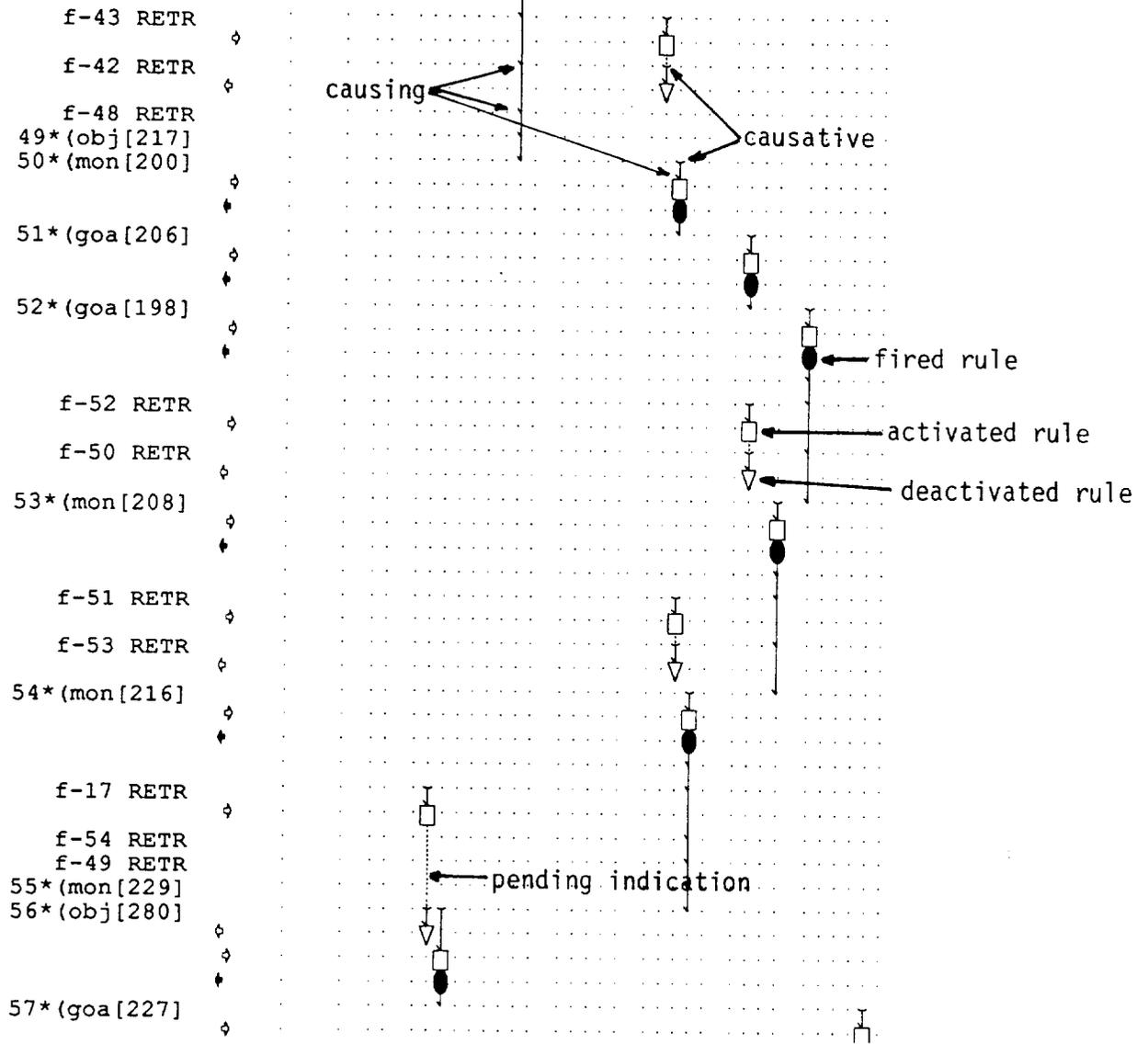


Figure 4

The causing symbol is also used to indicate a rule activation/deactivation associated with a fact operation.

The symbols for a rule activation, deactivation and firing are shown. A dotted line after a rule activation symbol is used to indicate a rule pending on the agenda. Since it is possible that more than one instance of a rule may be pending on the agenda at a given time, this is indicated by a widening of the dotted line. This indication is meant to be qualitative, not quantitative.

At the left of the rule portion of the display are symbols indicating agenda traffic. A hollow right arrow indicates a rule activation, while a hollow left arrow indicates a rule deactivation. A solid left arrow indicates a rule firing. Thus, the information inherent in the rule symbols elsewhere has been collapsed on the left side of the rule display. Note the

consistency between the hollow and solid symbols for both indications of rule activity.

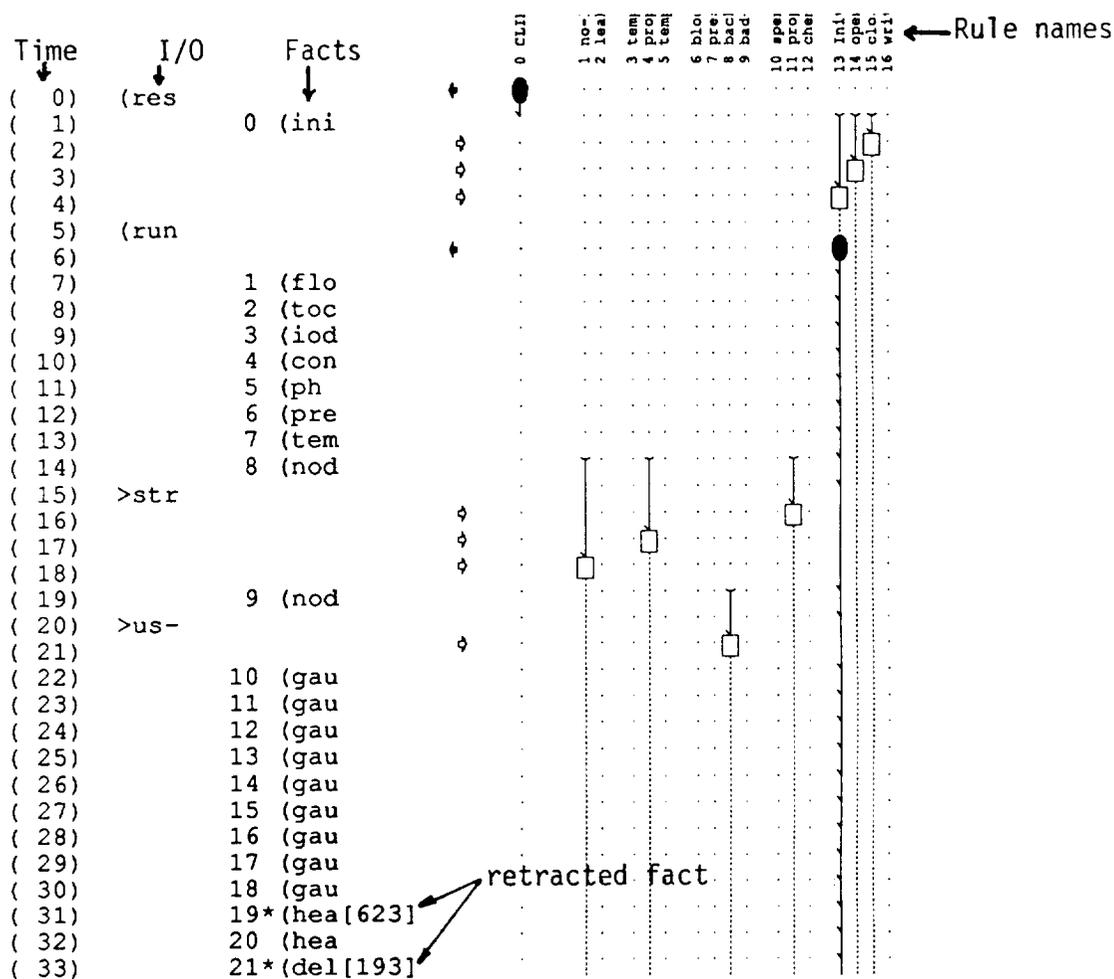


Figure 5

The rule names are listed across the top of the display while the leftmost columns indicate time, I/O, and fact information respectively. The time column is straightforward. A time slice is defined to be one trace output line. The I/O column contains both CLIPS directives, such as 'reset' and 'run', and actual application I/O. The CLIPS directives are indicated by a leading left parenthesis. Actual I/O is indicated by an imposed leading '>'. In the fact column is the information of the fact number and the first few characters of the fact itself. For the full details of the fact (or the I/O) one can refer to one of the adjunct files. Additionally, if the fact has been retracted, there will be an asterisk after the fact number and the retraction time in brackets after the initial fact characters. There will also be, at the retraction time, an indication of that fact number being retracted.

To aid in visual grouping of the abstract symbols, the user may specify in the segmented rule list the order of the rules defined and a spacing between their display. Thus, one could tell when rules of different types are active and, based on one's knowledge of the intended reasoning, be able to identify rules firing out of place.

Examples

In Figure 6 one can see several instances of rules activating due to some fact operation and then deactivating due to another fact operation within the consequent range of the same rule. Investigation showed that this was caused by inefficient consequent sequencing in the rules involved. Rearrangement of the consequents of the affected rules resulted in the disappearance of the excess agenda traffic.

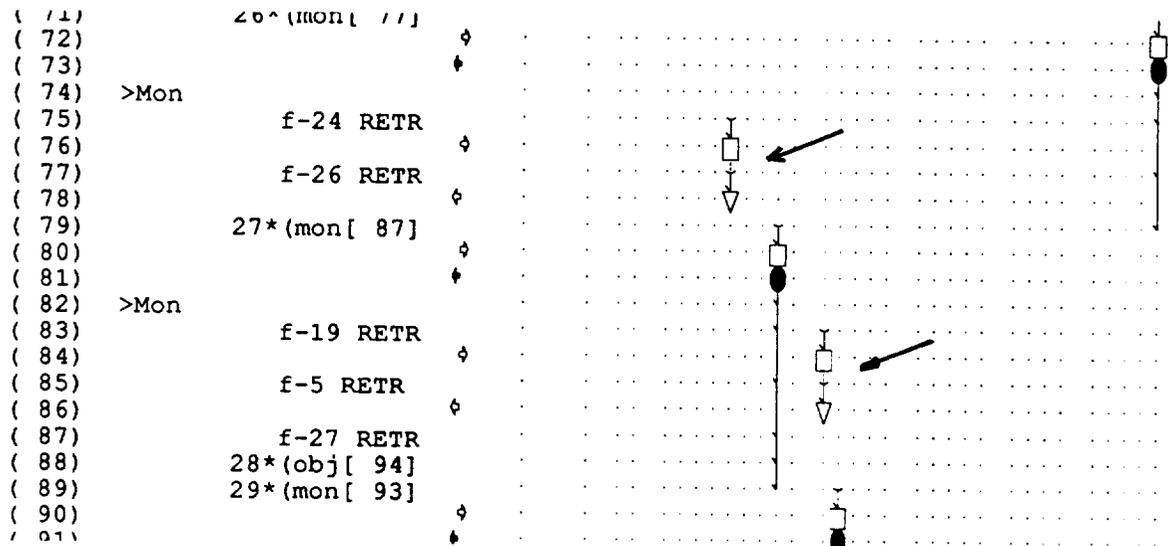


Figure 6

In Figure 7 the rules in the far right group are phase and sequencing control rules. The sequencing control rules are those that are cycling on a short time frame. The phase control rules are those that go on the agenda and then sit for a long time before firing. It can also be seen that after this transition the grouping of the remaining rule firings has changed.

In Figure 8 a rule is firing without any effects from its consequents, neither I/O or fact operations. While this appears to be an error, analysis showed that this rule was asserting a fact already in the fact list. Thus, CLIPS did not show that fact as being reasserted. In general, fact reassertion is inefficient and indicates a possible need for rule logic modification.

ATS displays were used during the development of a demonstrational prototype of an expert system application for controlling the potable water subsystem of the Environmental Control and Life Support System (ECLSS) aboard Space Station Freedom(8). There were two main contributions of ATS displays to that effort. First was the identification of inappropriate fact list changes causing rules to deactivate immediately after being activated. Second, the displays provided a picture of the overall patterns of rule activations giving a quick "state of health" view during subsequent knowledge base enhancements.

Conclusions

ATS displays have shown themselves to be useful for reasoning visualization at the host inferencing language level of a forward-chaining rule-based expert system by providing a global interrelated view of the large amount of available information. This

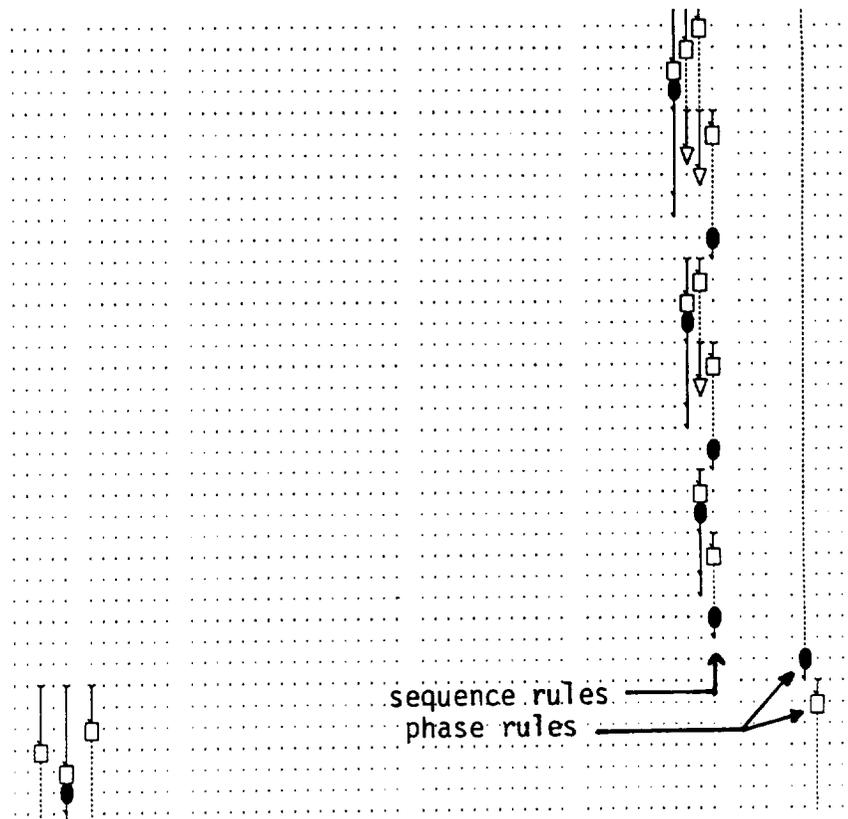


Figure 7

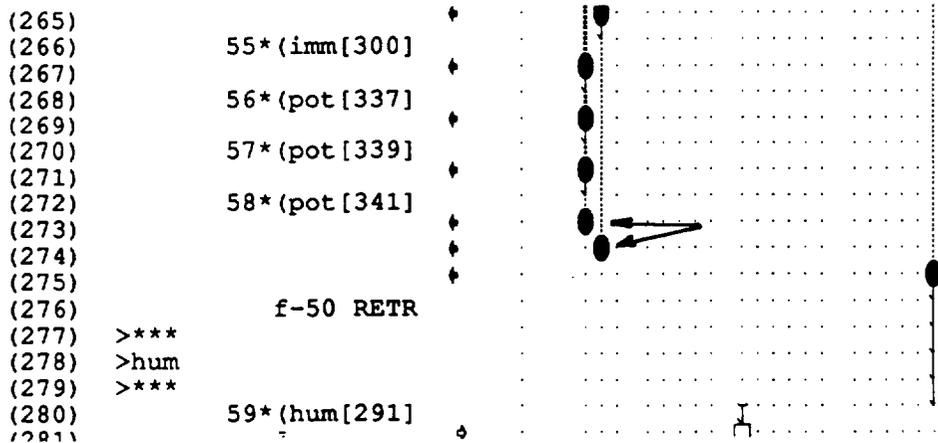


Figure 8

visualization is particularly useful during the development/debug and analysis phases of the expert system application building process. This provision of a gestalt not previously available allows the user to more easily identify and isolate problem areas than would be possible using detailed traces alone.

Acknowledgement

The authors would like to thank Dr. Dan Rochowiak of the Johnson Research Center at the University of Alabama in Huntsville for his application of ATS displays to the ECLSS potable water subsystem demonstrational prototype and for his many discussions of that effort.

References

1. Barr, A. and E. A. Feigenbaum, *The Handbook of Artificial Intelligence - Volumes I-III*, William Kaufmann, Los Altos, 1982.
2. Barr, A., P. R. Cohen, and E. A. Feigenbaum, *The Handbook of Artificial Intelligence - Volume IV*, Addison-Wesley, New York, 1989.
3. Bentley, J. L. and B. W. Kernighan, "A System for Algorithm Animation - Tutorial and User Manual," Computing Science Technical Report No. 132, AT&T Bell Laboratories, Murray Hill, NJ, 1987.
4. Bridges, S. M., "A Theory and Strategy for Justification Production by Expert Planning Systems," PhD Dissertation, University of Alabama in Huntsville, 1989.
5. Brown, M. H., *Algorithm Animation*, The MIT Press, Cambridge, Mass., 1987.
6. McKeown, K. R., *Text Generation*, Cambridge University Press, Cambridge, 1985.
7. Model, M. L., "Monitoring System Behavior In a Complex Computational Environment," Tech. Rep. CSL-79-1, XEROX PARC, 1979.
8. Schunk, R. G. and W. R. Humphries, "Environmental Control and Life Support Testing at The Marshall Space Flight Center," Proc. 17th Intersociety Conference on Environmental Systems, 1987.
9. Selig, W. J. and J. D. Johannes, "Reasoning Visualization in Expert Systems - The Applicability of Algorithm Animation Techniques," Proc. Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, 1990. In Preparation
10. Selig, W. J. and J. D. Johannes, "Towards Reasoning Visualization in Expert Systems," Proc. Second International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 1001-7, 1989.
11. Stasko, J. T., "TANGO: A Framework and System for Algorithm Animation," PhD Dissertation, Brown University, 1989. Also as Tech. Rep. CS-89-30
12. Tanimoto, S. L., *The Elements of Artificial Intelligence: An Introduction Using LISP*, Computer Science Press, Maryland, 1987.

ESTABLISHING A COMMUNICATIONS-INTENSIVE NETWORK
TO RESOLVE ARTIFICIAL INTELLIGENCE ISSUES
WITHIN NASA'S SPACE STATION FREEDOM
RESEARCH CENTERS COMMUNITY

E. Davis Howard, III
MSOR (candidate)

Johnson Research Center
University of Alabama in Huntsville
Huntsville, Alabama 35899

ABSTRACT

MITRE Corporation's *A Review of Space Station Freedom Program Capabilities for the Development and Application of Advanced Automation* (1) cites as a critical issue the following situation, extant at the NASA facilities visited in the course of preparing the review:

The major issues noted with regard to design and research facilities deal with cooperative problem solving, technology transfer, and communication between these facilities. While the authors were visiting lab and test beds to collect information, personnel at many of these facilities were interested in any information they could collect on activities at other facilities. A formal means of gathering this information could not be identified by these personnel. While communication between some facilities was taking place or was planned, for technology transfer or coordination of schedules (e.g., for SADP demonstrations), poor communication between these facilities could lead to a lack of technical standards, duplication of effort, poorly defined interfaces, scheduling problems, and increased cost. Formal mechanisms by which effective communication and cooperative problem solving can take place, and information can be disseminated, must be defined.

It is our purpose here to offer a proposed solution to the communications aspects of the issues addressed above; and, to offer at the same time a solution which can prove effective in dealing with some of the problems being encountered with expertise being lost via retirement or defection to the private sector. The proffered recommendations are recognizably cost-effective and tap the rising sector of expert knowledge being produced by the American academic community.

INTRODUCTION

It is well to note two factors at the outset of our examination of "communications" links among the various NASA research centers engaged in preparing for the launch of Space Station Freedom [SS Freedom]. The first is that "communications" as a concept in the artificial intelligence [AI] community has radically different implications than those toward which conventional management systems tend. The second is that developing trends in management science allow for this different notion of "communications"--indeed, the literature strongly encourages such differentiation of goals, via structural changes and adaptations.

Miles and Snow (8) trace the early benchmarks of management styles from 1800 to their belief of what organizations will look like in the year 2000. They have seen developments occur every 50 years from 1850, with the advent of functional structures to produce limited lines of goods, to the 1950's when, largely through military innovations as a result of World War II and the subsequent space explorations at NASA, the matrix form of management evolved to handle complex tasks associated with both standardized and highly innovative goods being produced by the same organization. Miles and Snow posit a new structure, the dynamic network, arising by 2000 in order to handle not only production but very specialized design of goods within the framework of a global marketplace. Structure itself will then become temporary, manifested by design problems, and will require immense levels of motivation of workers in order to obtain optimum performance.

In many ways, the work of Miles and Snow parallels the insights into the development of science from Newton [natural science, subject to laws] through the later discoveries of relativity and quantum mechanics and on into the current schools of epistemic priority of mind over nature [transcendental science]. (9) That is to say, that as technology advanced to a degree which allowed for an increasingly reductive view to be taken of nature, eventually all conventional observations became either known or predictable [Hawkings, quoted in Gleich (6)]. What is emerging is the study of the larger view taken of physical phenomena by scientists of chaos. Management theorists have gone so far as to "adopt" the highly scientific meaning of this new field taxonomically, as in Tom Peters's (10) latest work, *Thriving on Chaos*.

ISSUES

The purpose of this lengthy digression in a brief exposition is seminal: just as theoretical and experimental science have evolved into a more fluid study of chaos and the fundamental notion of a "grand unity" or "superstring" that encapsulates physical order completely, so management scientists and systems engineers, faced with the inevitability of global organizations, have sought out "dynamic networks" to enable them to cope with vast and changing complexities. Zee (12) goes so far as to diagram the "Drive Towards Unity" in physics.

From the 1950's onward, two forms of higher order programming languages have developed. One of these forms is the conventional, highly-driven, precisely-algorithmic and *computational* group of languages, FORTRAN, Pascal, C and Ada. These languages represent a kind of ideal in man's harnessing machines to take over the iterative tasks required for the further development of predictable technological models. The second group of languages is *knowledge-based* and *object-oriented* [vs. computational] to an extent that renders comparison of the two methodologies rather difficult.

While both are referred to as "computer languages,": the second category actually "reason" rather than taking sums or differentiating equations. Pagels (9) once asked Minsky, who with McCarthy is responsible for LISP, the language of AI endemic to American research facilities:

why he...chose to call their enterprise "artificial intelligence" rather than "cognitive science," which [Pagels] thought more appropriate. [Minsky] replied characteristically, "If we ever called it anything other than artificial intelligence, we wouldn't have gotten into the universities. Now that we're in and the philosophers and psychologists know that we're the enemy, it's too late.

The point here is that two very different world views have been rather "jumbled" together into a single field called "computer science"; and, that when Pagels refers to computers as "the primary research instrument of the sciences of complexity" (9) he is referring to the devices of cognitive science and not the devices of computation.

This author was once asked by a distinguished professor under whom he was working if he had come across a good definition of "software engineering" in the course of research. After much investigation over a year's time, no really good definition that did not encompass some degree of skepticism (2) emerged. Clearly, most practitioners of software engineering are working in the fields of general government, military or industrial applications; and, this limits their viewpoints towards the algorithmic languages, which can certainly be structured, if not exactly constructed. And, because this author's professor works almost exclusively within the fields of simulation, robotics and other AI applications, there was no common ground.

That NASA is clearly aware of these differences is implicit in the amount of time, energy and brainpower, to say nothing of the dollars, that it has devoted to the further development of LISP cultures through such agencies as DARPA and the various research centers NASA and the Department of Defense [DoD] have scattered throughout the United States.

"Communications" among these centers is critical for three reasons. There exist within the management of any enterprise, public or private, as America moves into the 1990's, three crises:

COMPETENCE: the loss of expertise that is not being replaced by technically-oriented majors in United States universities;

COMMUNICATIONS: in the broadest sense, management's ability to direct and motivate its workforce; and,

COST: the final critical issue, which entails getting the most effective output from every dollar of resources input with the aim of keeping a dynamic, project matrix organization afloat in a highly unstable and often unfriendly environment.

Of course, the paradigmatic solution to these issues of competence, communications and cost will be the knowledge-based [expert] modules, the inferencing engines and the neural networks [communication over a global span] inherent in the Fifth Generation of computers. The United States is uniquely placed to capitalize on the technologies that have arisen within its research laboratories [both governmental and private] and its academic circles. That NASA is keenly aware of the importance of the issue is evidenced by the findings of the *Space Station Advanced Automation Study: Final Report* (4) of 1988.

RECOMMENDATIONS

The first recommendation, in light of the foregoing analysis of SS Freedom as a dynamic organization, heavily tied to AI implementations, is that the structural design of the various research centers be investigated. Galbraith (5) recognizes the necessity for two types of structures within management frameworks of complex organizations. There must be first an "operating" side, which one might equate with the algorithmic philosophy of Ada, to handle the day-to-day iterative production; but, there needs to be a second group of "innovating" suborganizations to generate as well as to handle the implementation of new ideas. The innovating organization should be buffered from external pressures by an *orchestrator*: a power figure with the ability to enforce decisions that favor the changes, even when unpopular for reasons of politics or cost; they should be routinely managed by a *sponsor*: a management figure who handles budgets, requisitions and the like; and, finally, they must be lead and inspired by a *champion*: the person whose first responsibility is to guide the production process from idea to complete and tangible innovation.

The necessity for a champion is recognized by the Friedland group (4) in its insistence that each considered project first and foremost own the "presence of a strong user champion for the application" before being given further study for implementation at Baseline. At the same time, the Friedland group deal with institutional issues that confront the success of implementing AI innovations into SS Freedom. The issues are either mythaic or political in nature; and, as such, could best be addressed effectively by an orchestrator.

Thus, the conclusions of the first recommendation here, that all AI research work structures be examined within the NASA networks, is that similar models be adopted throughout the NASA system in order to protect on-going knowledge-based systems [KBS] projects. There is the strong recommendation from the Friedland report that an Operations Management System [OMS] to eventually take control of the complete SS Freedom system be investigated. Further, the Bayer report establishes 1996 as a benchmark year for demonstration of a distributed system for SS Freedom. In light of NASA's experience with AI systems, which far outstrips that of any Japanese agency, for instance (11), particularly where innovation is held to be a primary factor, this should be an obtainable goal. But, it is key to establish separate AI cultures within conventional management structures in order to protect America's technological lead.

The second recommendation is the establishment of a program similar to the Presidential Management Internships which serve to attract graduates of MBA programs into government careers. The candidates for acceptance as "Presidential AI Interns at NASA" would be graduate scholars finishing their degrees with experience and

strong LISP backgrounds. These scholars could be rotated on a three-month basis from one Research Center to another, each serving for the time in one of the innovating AI cells.

There would be established a central office for NASA to act as a clearing house for coordinating reports of on-going work and progress within each of the NASA centers and within each of the suborganizations engaged in an AI activity at those centers. As part of the rotational program, the scholars would meet for roundtable discussions at the end of their service time at each center. The bottom line for this idea is that AI does not operate algorithmically and cannot be transmitted by driven means. It is best expressed in the spirit of that which it emulates, the human intelligence; intelligent exchanges among experts in the LISP and AI fields will rectify the causes of concern that were evinced in the Mitre Corporation study. Until such time as the DMS and OMS are operational, this kind of program would not only ensure that a full level of communication of the state-of-the-art progress at the various NASA AI centers was in effect. It would also fill the gap of vanishing competence.

In terms of the cost effectiveness of such a program, the current level of adoption and experimentation with AI in the private sector is the clearest indication that AI is an imperative in the face of global competition. *The Rise of the Expert Company* (3) gives a fascinating account of developments within aerospace and computer firms as well as government agencies. Friedland cites several examples of improvement in operations for both cost savings and quality.

CONCLUSION

While it is impossible to fault the logic or the *politesse* of Friedland's group and their conclusions, the case for AI as an imperative rather than as an option on SS Freedom--if that enterprise is to flourish--has perhaps not been stated strongly enough. Elsewhere, this author has undertaken a comparative study of Ada, C and LISP (7). The conclusions from that study lead one to espouse the notion that if America maintains its current *innovative* lead in technology, it will do so through AI. NASA and DoD can find ready customers in the private sectors of the United States, the Pacific Rim countries and the new European group, for AI design innovations. Encouraging the sound management of current AI resources at NASA, and funding new efforts, is urgent. As Pagels (9) notes:

...the nations...who master the new sciences of complexity will become the economic, cultural and political superpowers of the next century.

ACKNOWLEDGMENT

This research has been partially funded under NAS8-36955 (Marshall Space Flight Center) D.O.50, "FNAS Space Station Automation."

REFERENCES

1. BAYER, S., R. Harris, L. Morgan, and J. Spitzer. *A Review of Space Station Freedom Program Capabilities for the Development and Application of Advanced Automation*. NAS9-18057. The MITRE Corporation, Civil Services Division, McLean, VA, 1988.
2. FAIRLEY, R. *Software Engineering Concepts*. McGraw-Hill Book Company, New York, 1985.
3. FEIGENBAUM, E., P. McCorduck, and H. Nii. *The Rise of the Expert Company*. Times Books, New York, 1988.
4. FRIEDLAND, P., et al. *Space Station Advanced Automation Study: Final Report*. Office of Space Station, Strategic Plans and Programs Division, NASA Headquarters, 1988.
5. GALBRAITH, J. "Designing the Innovating Organization," in *Organization: Text, Cases and Readings on the Management of Organizational Design and Change*. Kotter, Schlesinger, & Sathe [Eds.], 2nd Ed., IRWIN, Homewood, IL, 1978.
6. GLEICH, J. *CHAOS; Making a New Science*. Viking, New York, 1987.
7. HOWARD, E., and D. Rochowiak. *SOFTWARE: LANGUAGES AND MANAGEMENT*. Johnson Research Center, The University of Alabama in Huntsville, Huntsville, AL, 1988.
8. MILES, R., and C. Snow. "Fit, Failure and the Hall of Fame," in *Organization: Text, Cases and Readings on the Management of Organizational Design and Change*. Kotter, Schlesinger, & Sathe [Eds.], 2nd Ed., IRWIN, Homewood, IL, 1978.
9. PAGELS, H. *The Dreams of Reason: The Computer and the Rise of the Sciences of Complexity*. Simon and Schuster, New York, 1988.
10. PETERS, T. *Thriving on Chaos*. Alfred A. Knopf, New York, 1987.
11. WYSOCKI, B. "The Wall Street Journal Reports. Technology: The Final Frontier. Japan Assaults the Last Bastion: America's Lead in Innovation," *The Wall Street Journal*, 14 November 1988, Section 4, R1-R46.
12. ZEE, A. *Fearful Symmetry: The Search for Beauty in Modern Physics*. MacMillan Publishing Company, New York, 1986.

EXPERT SYSTEM DEVELOPMENT METHODOLOGY (ESDM)

Charisse Sary (301) 572-8719
 Lewey Gilstrap (301) 572-8335
 Computer Sciences Corporation
 System Sciences Division
 4600 Powder Mill Road
 Beltsville, Maryland 20705

Larry G. Hull (301) 286-3009
 NASA Goddard Space Flight Center
 Greenbelt, Maryland 20771

ABSTRACT

The Expert System Development Methodology (ESDM) provides an approach to developing expert system software. Because of the uncertainty associated with this process, an element of risk is involved. ESDM is designed to address the issue of risk and to acquire the information needed for this purpose in an evolutionary manner. ESDM presents a life cycle in which a prototype evolves through five stages of development. Each stage consists of five steps, leading to a prototype for that stage. Development may proceed to a conventional development methodology (CDM) at any time if enough has been learned about the problem to write requirements. ESDM produces requirements so that a product may be built with a CDM. ESDM is considered preliminary because it has not yet been applied to actual projects. It has been retrospectively evaluated by comparing the methods used in two ongoing expert system development projects that did not explicitly choose to use this methodology but which provided useful insights into actual expert system development practices and problems. This fiscal year, the methodology will be field-tested by applying it to two pilot projects.

INTRODUCTION

An expert system is a computer system that simulates, to the extent possible or practical, the way human experts solve cognitive problems, i.e., those for which no algorithm is known to solve the problem, but that are routinely solved by a domain expert using rules of thumb or heuristics. The task of developing an expert system is, therefore, to acquire information from the domain expert as to how the cognitive tasks are performed and then model the information in a form suitable for the computer. Because of the uncertainty associated with this process, an element of risk is associated with developing expert systems. ESDM has been designed to address the issue of risk and to acquire the information needed for this purpose in an evolutionary manner.

Note that it is not possible to develop specifications for an expert system before the expert's reasoning processes have been analyzed. This lack of specifications presents the manager with special problems in controlling an expert system development project. These problems are addressed in ESDM.

Analyses reveal differences between expert systems and conventional software development. Compared to conventional development, the expert system life cycle more nearly resembles rapid prototyping. The major differences between the expert system life cycle and the rapid prototyping of conventional development are

- Greater uncertainty in feasibility and suitability of the problem to an expert system implementation

- Highly iterative nature of expert system development

The major similarity is that the system produced is a prototype. More specifically, all systems produced by ESDM are considered prototypes. ESDM does not produce any of the standard products (e.g., feasibility study, requirements analysis) that are produced for a conventional software system.

The methods used to implement expert systems differ from those used for conventional systems. However, once the uncertainty of the feasibility of an expert system is reduced to a low level (i.e., after several iterations of prototype development), it will be possible to undertake development of an expert system in a conventional development cycle and to expect it to present no more difficulty than any other system. ESDM precedes conventional development and terminates when requirements can be produced.

ESDM is intended to be applied to the development of National Aeronautics and Space Administration/Goddard Space Flight Center (NASA/GSFC) expert systems. It is based on a survey of existing methodologies and an analysis of the expert system life cycle. Dr. Barry Boehm introduced a risk-driven methodology for conventional system development in his spiral model for software development (1). ESDM, while independently generated, is also a risk-driven methodology that can be represented as a spiral model. ESDM focuses more on knowledge acquisition rather than on product development. Figure 1 illustrates the spiral nature of ESDM.

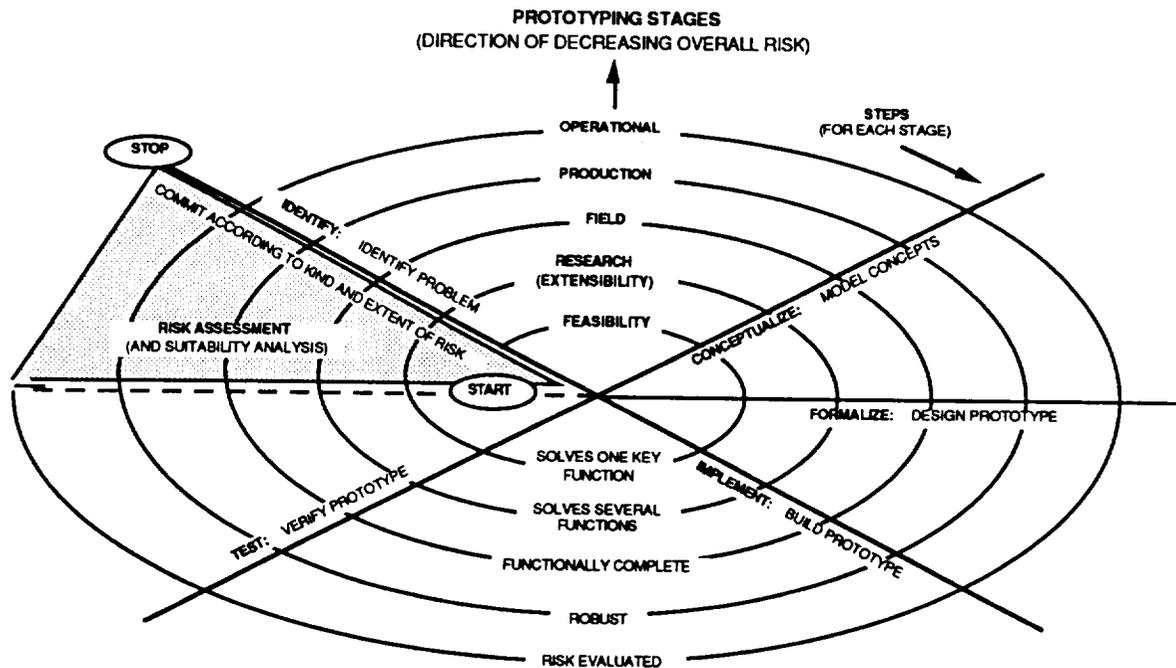


Figure 1. Spiral Model of ESDM

ESDM was developed to address large projects. Therefore, when applying ESDM to projects that are small, ESDM recommendations should be used as a guideline and adapted to suit the smaller-size project.

The proposed methodology is considered preliminary because it is untested. It is a synthesis of methodologies that have been used in conventional and expert system software development. A standard (4), a user guide (6), a reference manual (3), and a set of training materials (5) have been developed to describe the methodology. The methodology has been retrospectively evaluated by comparing the methods used in two ongoing GSFC expert system development projects: the Ranging Equipment Diagnostic Expert System (REDEX) and the Backup Control Mode Analysis and Utility System (BCAUS) (2). These projects did not explicitly choose to use the methodology but provided useful insights into actual expert system development practices and problems. This fiscal year, the methodology is being field-tested by applying it to two other GSFC expert system development projects: the Generic Expert System and the Systems Test and Operations Language (STOL) Intelligent Tutoring System (ITS). Next fiscal year, the methodology will be revised based on the results obtained.

DECISION POINTS OF ESDM

ESDM consists of five stages, each accomplished in five steps, and is geared to reducing the uncertainty of feasibility and risk in development. The ESDM life cycle is illustrated in Figure 2, which shows five decision points that determine whether work must move to one or another of the five ESDM stages. In contrast with CDM, iteration of a stage or iteration of steps within a stage may also occur. (Iteration of steps is not shown explicitly in the figure.) Higher risk issues are addressed first, and lower risk issues are addressed with each subsequent stage. Earlier stages focus on knowledge acquisition, and later stages focus on performance issues. The stages and stage products described serve as a guide and should be adapted to suit the goals and objectives of each individual project. If an alternate area is of higher risk for the project, this area is prototyped first.

In addition, there are three risk-based decisions, as follows:

- **Apply ESDM?** Are expert system development techniques suitable for the problem?
- **Stop ESDM?** Has enough been learned to decide whether to move to a CDM or to abort development? If the answer is no, then a stage is selected. A stage may be repeated or a new stage selected. If the answer is yes, the next decision is addressed.
- **Move to a CDM?** Should development move to a CDM or be aborted? Are the requirements now known, or is the risk too high to continue?

Development need not proceed through all five stages if these questions can be answered in an earlier stage. ESDM provides a metric--the Test for Application of Risk-Oriented Technology (TAROT)--to provide guidance to both managers and developers at each decision point. TAROT assists in evaluating a project for suitability to an expert system implementation and in estimating the risk involved. At the completion of the project using ESDM, a transfer to product cycle is undertaken and development continues using a CDM.

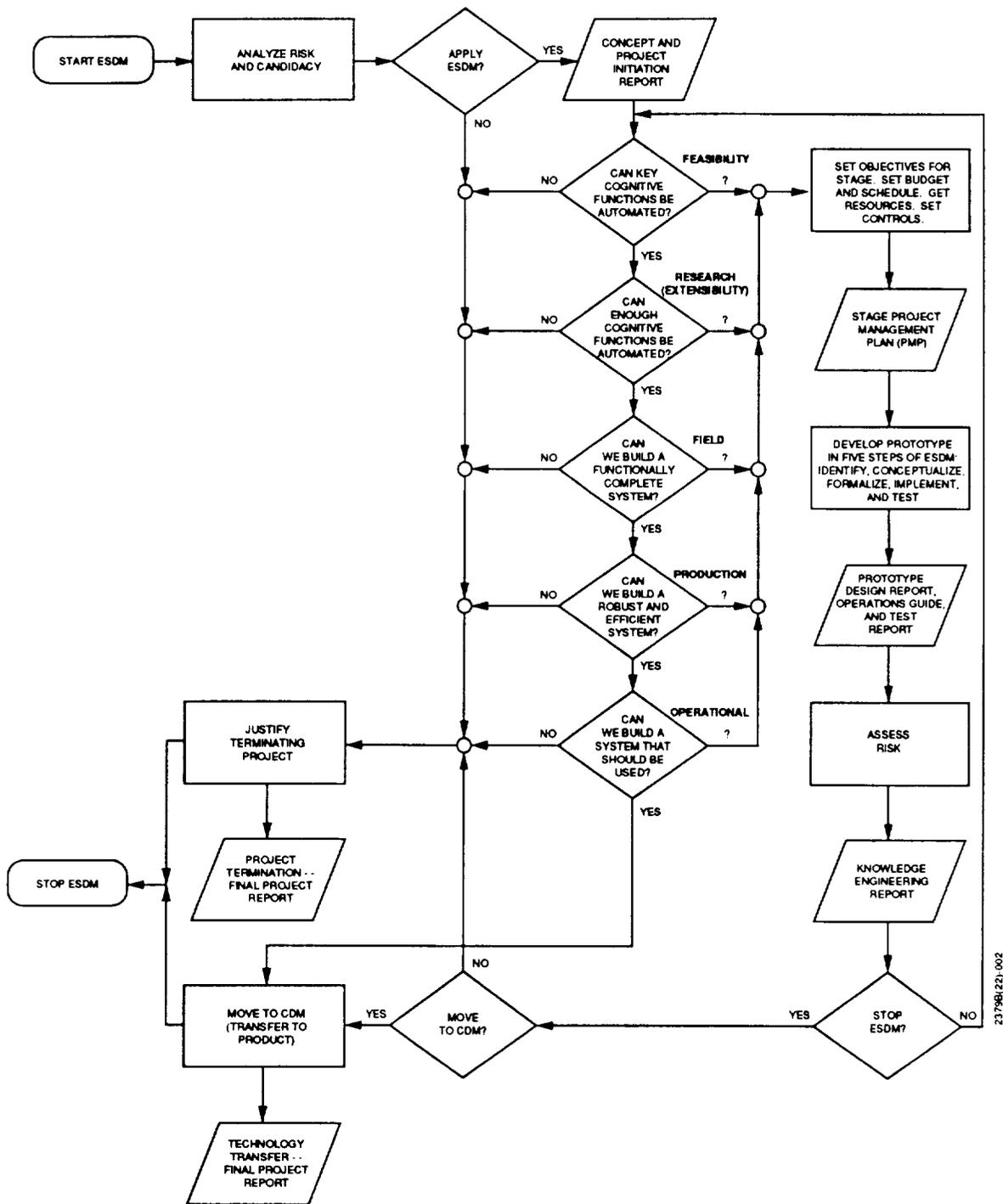


Figure 2. The ESDM Life Cycle

FIVE STAGES IN EXPERT SYSTEMS DEVELOPMENT

The proposed ESDM consists of five stages of prototype development (8). Each stage consists of five steps (7). The stages are discussed in the following subsections; the steps for each stage are discussed in the next section.

Stage 1: Feasibility

Model one or more key cognitive functions performed by domain expert.

A key cognitive function is one that is central to the overall problem the expert solves. Selecting a key function is important but not crucial because ESDM is highly iterative.

Stage 2: Research (Extensibility)

Model remaining functions.

The goal of this stage is to determine the extent to which the remaining cognitive functions performed by the domain expert can be modeled.

Stage 3: Field

Model additional functions required in the field.

This stage determines how to model the remaining cognitive functions performed by the expert. Specifically, this stage determines what combination of conventional and expert system techniques should be used to construct an automated replacement of the manual system. The goal of this stage is to design and construct a fieldable prototype that can be tested and used in a realistic setting. Performance issues have not been addressed yet.

Stage 4: Production

Address performance objectives.

This stage determines if it is possible to refine the design of, recode, or transport the system to a new host to achieve the desired scope of expertise and performance objectives. The successful production prototype is able to solve nearly all required problems and is robust and is easy to use.

Stage 5: Operational

Analyze and evaluate risks and costs of deploying the production prototype.

To reach this stage in ESDM, the development team has learned it is possible to build a prototype that simulates the functions performed by the domain expert. A knowledge acquisition process is still involved in this stage, but the focus is now on the use of the product rather than on the structure of the product.

FIVE STEPS IN EXPERT SYSTEMS DEVELOPMENT

Within each stage of expert system development, five steps are required to produce the stage prototype. These steps are highly iterative and are not necessarily performed in a simple, sequential manner--developers often work on several steps concurrently. Only in a general sense

does the work progress from one step to the next. Typically, work can and does move from a later step to an earlier step as knowledge is acquired.

Step 1: Identification

Determine problem characteristics.

In this step, the knowledge engineer identifies the problem the expert system will solve for the current stage. In the feasibility stage, the sources of expertise are identified. These sources may include one or more experts, reports, or manuals. Knowledge acquisition for the stage begins at this point.

Step 2: Conceptualization

Find concepts to represent the knowledge.

Knowledge acquisition continues in this step and includes identifying the concepts, objects the expert reasons about, the relationships between objects, control flow, constraints, and problem-solving strategies used by the expert. The knowledge is organized and modeled using a technique suitable to the problem (i.e., tables, informal rules, flow charts, hierarchies).

Step 3: Formalization

Map these concepts to formal representations based on the selected implementation tool or language.

This step involves formalizing the concepts chosen in the conceptualization step, designing the system, and selecting the hardware and software tools for implementation. The suitability of the tool or language selected for representing the problem determines the difficulty of subsequent steps.

Step 4: Implementation

Formulate rules that embody knowledge.

In the implementation step, a prototype is coded from the formal representations developed in the formalization step using the selected software tool or language.

Step 5: Test

Validate rules that embody knowledge and verify that the prototype implements the design.

This step involves evaluating the prototype's performance through testing. In the early stages of ESDM, the prototype's performance is compared against that of the domain expert. In later stages, performance issues are evaluated (e.g., robustness, speed of execution, ease of use).

TRANSFER TO PRODUCT CYCLE

An ESDM project terminates and proceeds to CDM when the risk of feasibility and use are reduced to an acceptable level and when requirements can be sufficiently defined to continue development in a sequential manner. The findings of ESDM serve as the basis for CDM and are formalized and transferred in a formal transfer review meeting. Expert system project personnel should assist in, monitor progress of, or perform the conventional product development to handle problems or issues that might arise.

An ESDM project terminates and development is discontinued if the risks of either implementation or use are considered unacceptable. A final lessons learned review meeting is conducted to preserve knowledge acquired on the project and to explain this decision.

THE ESDM PRODUCTS

The products produced at the start of an ESDM project include the following:

- **Concept and Project Initiation Report**—A high-level strategic plan for the project. It explains the results of initial risk and suitability analysis, describes the system to be automated, and justifies the project.

The products produced in ESDM in each stage of development include the following:

- **Stage Project Management Plan**—A lower level, more detailed plan than the Concept and Project Initiation Report. It is produced at the start of each stage and describes work to be completed for that stage.
- **Knowledge Engineering Report**—A summary of the knowledge acquired and the lessons learned during the stage, as well as recommendations for later stages of work.
- **Prototype Design Report**—A detailed description of the rules or other knowledge structures used in the prototype.
- **Prototype Operations Guide**—A description of prototype operation for subsequent testers.
- **Test and Evaluation Report**—A description of the test results. Evaluates the prototype.
- **The prototype.**

The products produced at the end of a project include one of the following:

- **Technology Transfer Report**—Produced following the decision to continue development with CDM. The report summarizes project findings and lessons learned, and presents recommendations for CDM development.
- **Project Termination Report**—Produced following the decision to abort development. The report justifies this decision, summarizes lessons learned, and documents results of knowledge acquisition.

ACKNOWLEDGMENTS

This work was funded by NASA/GSFC under contract NAS 5-31500. The authors wish to acknowledge J. Retter, J. Baumert, A. Critchfield, and K. Leavitt for their early work on this project.

REFERENCES

1. Boehm, B. W., "A Spiral Model of Software Development and Enhancement," IEEE Computer, May 1988, pp. 61-72
2. Computer Sciences Corporation, Expert System Development Methodology: Framework for Evaluation of ESDM, March 1989
3. Computer Sciences Corporation, Expert System Development Methodology Reference Manual, Revision 1, DSTL-90-006, August 1989
4. Computer Sciences Corporation, Expert System Development Methodology Standard, Revision 1, DSTL-90-005, August 1989
5. Computer Sciences Corporation, Expert System Development Methodology Training Materials, CSC/TM-89/6091, July 1989
6. Computer Sciences Corporation, Expert System Development Methodology User Guide, Revision 1, DSTL-90-004, August 1989
7. Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., Building Expert Systems, Addison-Wesley Publishing Company, Inc., 1983
8. Waterman, D. A., A Guide to Expert Systems, Addison-Wesley Publishing Company, Inc., 1986

A HYBRID APPROACH TO SPACE POWER CONTROL

E. W. Gholdston D. F. Janik K. A. Newton

Rocketdyne Division, Rockwell International
Canoga Park, California 91303

Abstract

Conventional control systems have traditionally been utilized for space-based power designs. However, the use of expert systems is becoming important for NASA applications. Rocketdyne has been pursuing the development of expert systems to aid and enhance control designs of space-based power systems. The need for integrated expert systems is vital for the development of autonomous power systems.

Introduction

The Rocketdyne Division of Rockwell International Corporation, Canoga Park, California, is pursuing research on space-based and lunar power systems design. Large scale power systems for NASA's future space and lunar missions pose significant challenges for autonomous control on-orbit. Conventional computer control techniques utilize highly structured algorithmic control code which can deal with a predetermined set of specific contingencies. In addition to control algorithms and power load flows, security analysis, system maintainability, and predictive simulations can also be used for power system control.

Conversely, expert systems and artificial intelligence technology offer great flexibility for control systems due to their response to unspecified situations. Research in the field of expert system applications for power systems is being pursued. However, acceptance and verification of these systems is an emerging discipline that does not have established criteria. Because expert systems may be less deterministic, control designs that depend solely on them may have reduced reliability. Integration of conventional control techniques with expert systems, or hybrid systems, could increase reliability of power systems utilizing this architecture.

Promising applications of hybrid systems include monitoring and diagnosing power systems. The potential use for hybrid systems is significant and offers great flexibility for system control. Additionally, challenges are introduced for development, test, and verification of expert control systems. This paper will describe research at Rocketdyne in the area of expert systems and hybrid systems.

Previous Research

Rocketdyne has been researching the development and integration expert systems for the evaluation of autonomous power applications. This research was conducted in the Space Power Electronics Laboratory (SPEL) facility at Rocketdyne. The SPEL contains a breadboarded space power distribution testbed where power system control methodologies can be evaluated.

Using the SPEL, development and testing of expert systems for hybrid computation has been explored. One expert system (ES) project concentrated on a single power switchgear component: a remote bus isolator (RBI) [3]. The ES developed could detect an anomaly within the RBI, based on power measurements of the component. From the resulting analysis, a single fault could be diagnosed for the RBI.

Building on this research, an expanded expert system effort was started [4]. Power hardware was added and diagnostic capabilities of the expert system were extended. The scope of the project was increased and new expert system development and interface environments were selected and used. This expert system, the Diagnostic Expert System (DES), diagnosed a larger component within the SPEL that was more complex than a single remote bus isolator (RBI): a power distribution control unit (PDCU).

The failures detected by this diagnostic system included short circuits, over-currents, loss of power, power surges, and loss of communication. The expert system integrated power system diagnosis with basic control algorithms.

While the capabilities for the DES were increased, there were problems during the development and execution. Accessing and converting the testbed measurements to an object-oriented structure resulted in timing problems. Additionally, not all knowledge acquired during the project was incorporated, due to the specific scope of the project. Expansion of DES needed to address additional fault diagnosis and detection, increased interactions with power system simulations, and study the effects of system loads and multiple failures.

Current Research

Based on the above assessments, the third expert system project was started at Rocketdyne, enhancing the capabilities of the DES. Increasing the scope over previous work was the first step of the project. The expert system's utility was also changed. This change was based on a recommendation that an expert system design relying solely on knowledge-base diagnoses would not be sufficient for intelligent implementation in a dynamic environment, such as a power system.

The expert system project would need commonality with existing or projected space-based power systems standards. For example, using an 80386-based computer would be more appropriate than using specialized artificial intelligence hardware. Because of emerging NASA software standards, integration with Ada control software needed to be explored and this priority maintained as the expert system developed. As expressed by current power system researchers, it will be a step-by-step approach to have expert systems be part of a closed loop control design for power systems [9]. Based on the above considerations, the enhanced expert system is called IPAC for Integrated Power Advisory Controller.

Facilities

For the development of IPAC, the SPEL testbed was used as shown in Figure 1. Two power sources, a simulated Photovoltaic source and Solar Dynamic source, produce single phase 440 volt electrical power distributed at a frequency of 20 KHz, which is being converted to dc.

Real and reactive programmable load elements are used to draw electrical power through the distribution grid enabling test engineers to emulate various user load profiles. The power distribution network is comprised of main bus switching units (MBSUs) and power distribution control units (PDCUs). The MBSU is an assembly used to supply electrical power to main

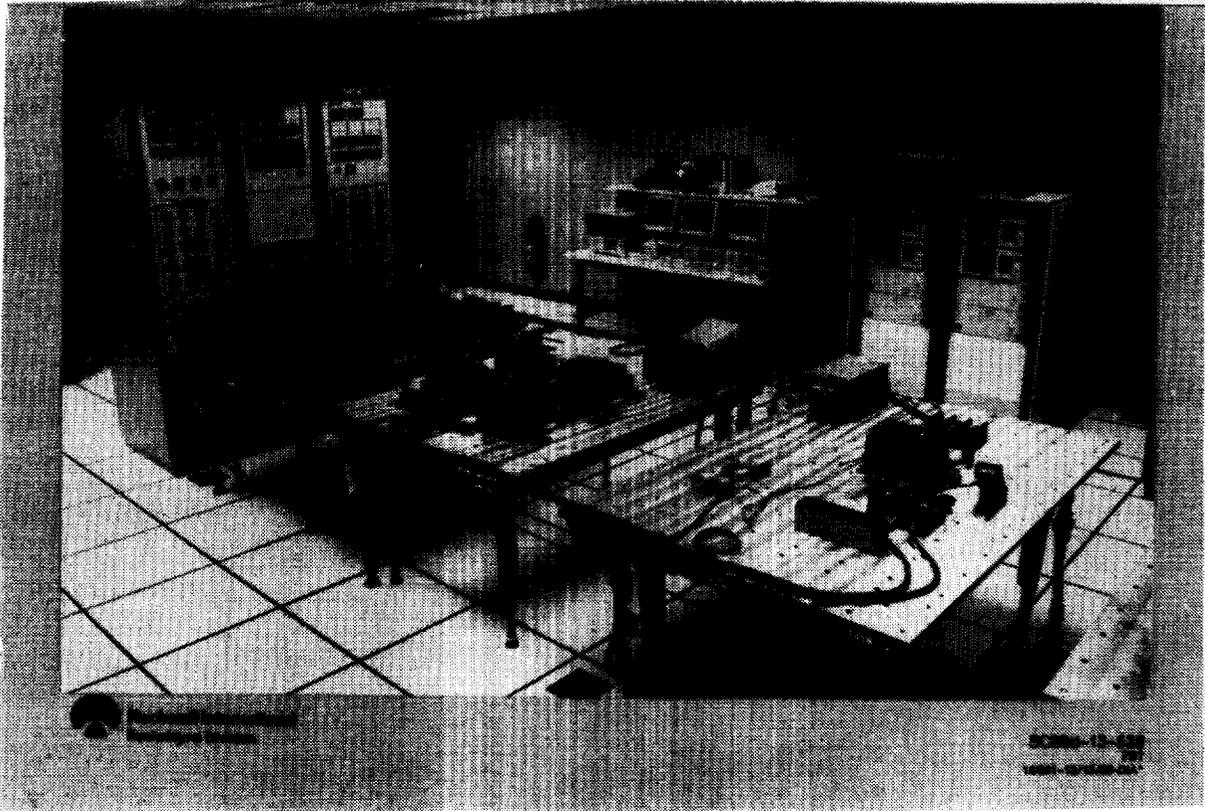


Figure 1. Electrical Power Testbed
in Space Power Electronics Laboratory.

feeder lines (power buses). The PDCU is an assembly used to distribute power directly to user loads. The fundamental building blocks for each of these assemblies are remote bus isolators (RBIs) and remote power controllers (RPCs).

The SPEL testbed was built using RBIs and RPCs. The current configuration divides the testbed into two sides representing the port and starboard sides of a space power system, with redundant power sources. This configuration is partitioned as shown in Figure 2 to distinguish outboard and inboard components. Outboard electrical components model distribution of electricity from the power sources to the first major switching assemblies. Inboard electrical components place distributed power into the feeder network.

Communication with components on the SPEL testbed is performed using a Mil-Std 1553 data bus. The component measurements transmitted are voltage, current, phase, temperature, and status. Due to power requirements of some NASA missions, the testbed is planned to be operated in parallel with a dc testbed. Changes to the dc testbed include new dc switchgear components and configurations.

Hardware and Software

The hardware used for IPAC, located in the SPEL, consists of a Compaq 386/20 personal computer and the SPEL testbed. The computer has 4 Megabytes of Extended Memory, a 60 Megabyte hard-disk, and a 80387 math co-processor. The computer has serial and parallel ports which can communicate with Mil-Std 1553 data bus and a RS-232 bus connected to the SPEL testbed. This computer hosts the expert system software, as well as the Mil-Std 1553 data communication software.

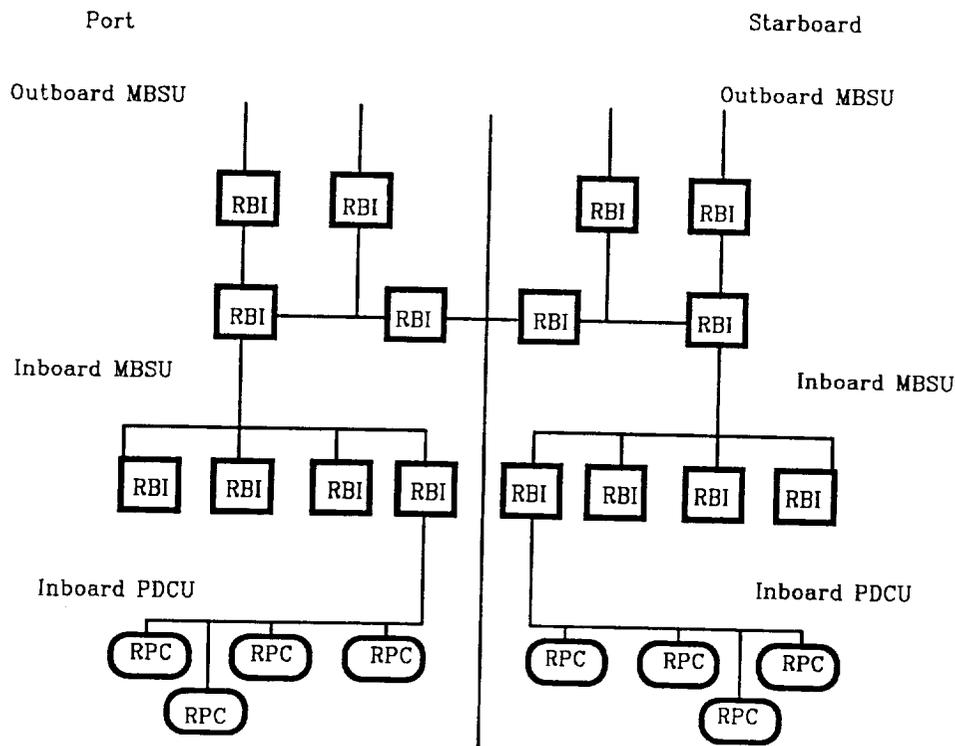


Figure 2. Testbed configuration.

Interacting with the SPEL is the expert system software. The expert system shell is Nexpert Object. This object-oriented environment is appropriate for dynamic representation of hierarchical power systems. An interactive graphical database program is the user interface for the expert system. This program, Ease +, can display dynamic updates of acquired data or consultation results. This interface reads data buffers generated by the component measurements on the SPEL testbed. An overview of IPAC's environment is shown in Figure 3.

Testing

To allow for increased capabilities for IPAC, knowledge acquisition using the SPEL testbed was started. The testing techniques are similar to ones used in previous research [4]. In particular, predictive indicators of faults are being assessed. The more accurately a fault can be defined, the better a power system may be utilized [7]. This philosophy translates to monitoring multiple points on a bus to detect variations within the power system. Basic diagnostics that were researched for a PDCU are now being applied to a MBSU. Switchgear interactions, load profiles, and multiple failures are being added to the test sequences. Some tests used last year are being conducted again to verify the results and look for additional indicators or responses.

A preliminary result from continuing knowledge acquisition reveals the testbed is governed by meta-knowledge that will be incorporated into the expert system structure. This knowledge consists of information that distinguishes which diagnostic or numerical methods programs or both should be used to check for anomalies. Constraint suspension models would indicate similar results, but these models address only functional representation [2]. IPAC's knowledge

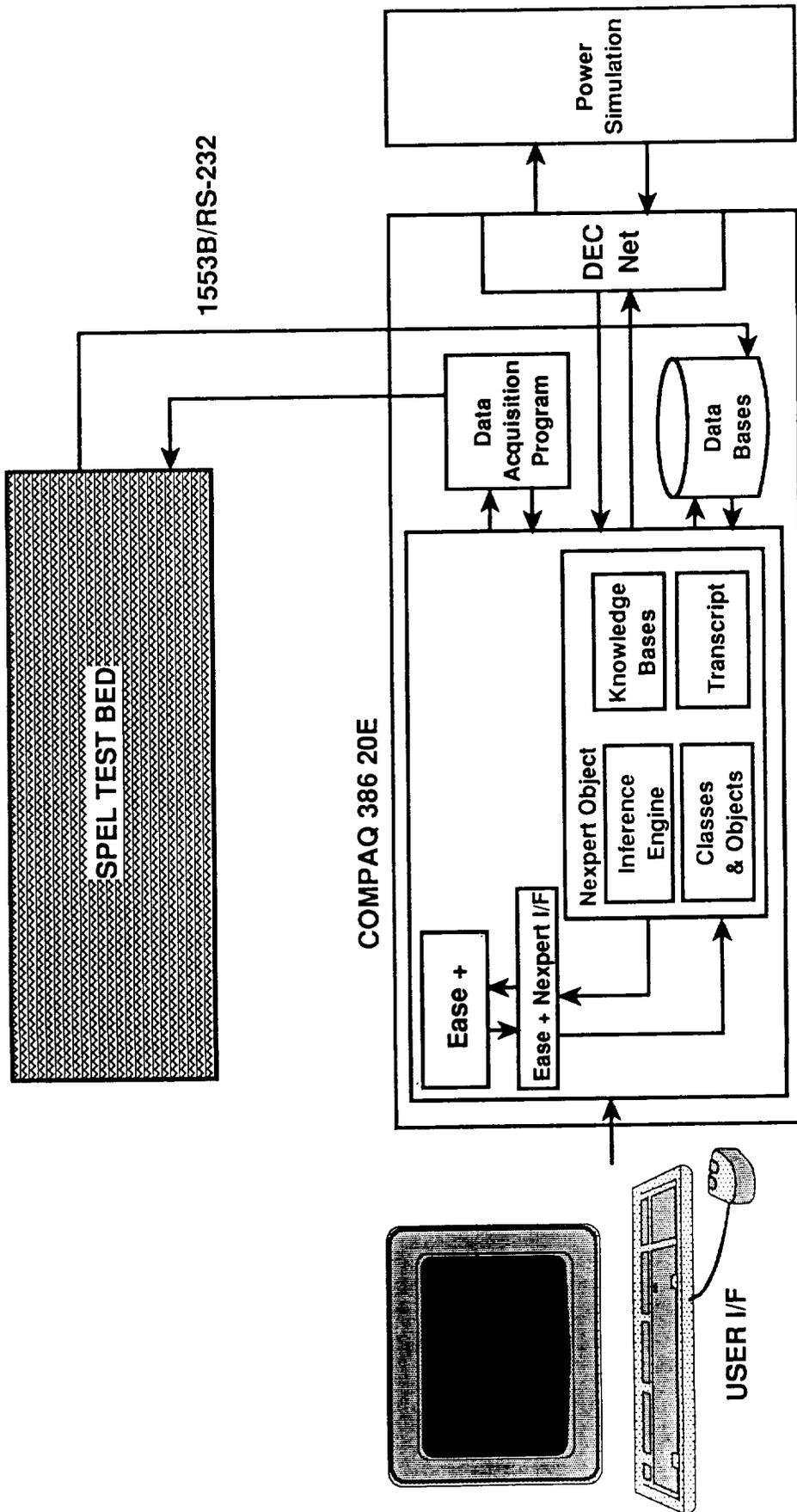


Figure 3. IPAC Environment.

is based on both functional and physical models. While similar research is being conducted at other facilities [6,8], IPAC is utilizing Rocketdyne's experienced power engineers and state-of-the-art facilities and technologies. These strengths are allowing IPAC to be developed and to evolve beyond expert system diagnostics to a total integrated power system environment.

Knowledgebase Structure and Development

To accommodate the growth in IPAC's knowledge, the structure and interactions of the DES knowledge base was changed. The DES used Ease+ only to access testbed measurements during execution, and then load the command knowledge-base. The command knowledge-base would then perform measurement validation and load appropriate knowledgebase(s) for various diagnoses and analyses (ie: communication failures, short circuits, degradation, etc.). This structure was not the most efficient for detecting some faults conditions on the testbed and for utilizing multiple knowledgebases.

IPAC's new structure alleviates these and other problems. First, the user is allowed to access three options for data during a session:

- Current testbed measurements
- Archived testbed measurements
- Simulation measurements

Data is continuously polled during a session. Depending on the option chosen, switchgear measurements will be accessed from the appropriate source and validation of these values will occur within the user interface software. This scheme is faster than using the knowledge-bases to check for correct values. While the data is being verified, critical conditions, such as short circuits, are being evaluated. This structure allows timely advisement to the user of critical problems. Once measurement validation is completed, appropriate knowledgebase(s) or programs will be loaded.

The knowledge-base(s) or numeric programs to be loaded are dependent on meta-rules within IPAC. These rules form a quasi-blackboard environment that accesses the modular programs. There are three classes of programs that can be accessed as shown in Figure 4: System Monitor, Fault Detection and Diagnostics, and System Simulation. The system monitor class contains trend analysis, component degradation, and numeric analysis programs. The fault detection and diagnostics system class contains expert systems that detect anomalies within the power system including the following:

- Communication failure detection
- Control power failure detection
- High Impedance detection
- Low Impedance detection

The system simulation module will be discussed below. Individual programs can be called from the quasi-blackboard. In the case of insufficient information, all programs will be run. Additionally, the quasi-blackboard environment can access archived data files and create new files for predictive simulations, access the SPEL testbed for testing and verification, and access the power system simulation.

When faults or possible problems are detected within the programs, advisory messages and reconfiguration suggestions are sent to the user, via the quasi-blackboard. Conflicting advice will be evaluated and adjusted by IPAC. The data is updated continuously throughout the user's session and the IPAC environment interacts with Ada control software to access the testbed.

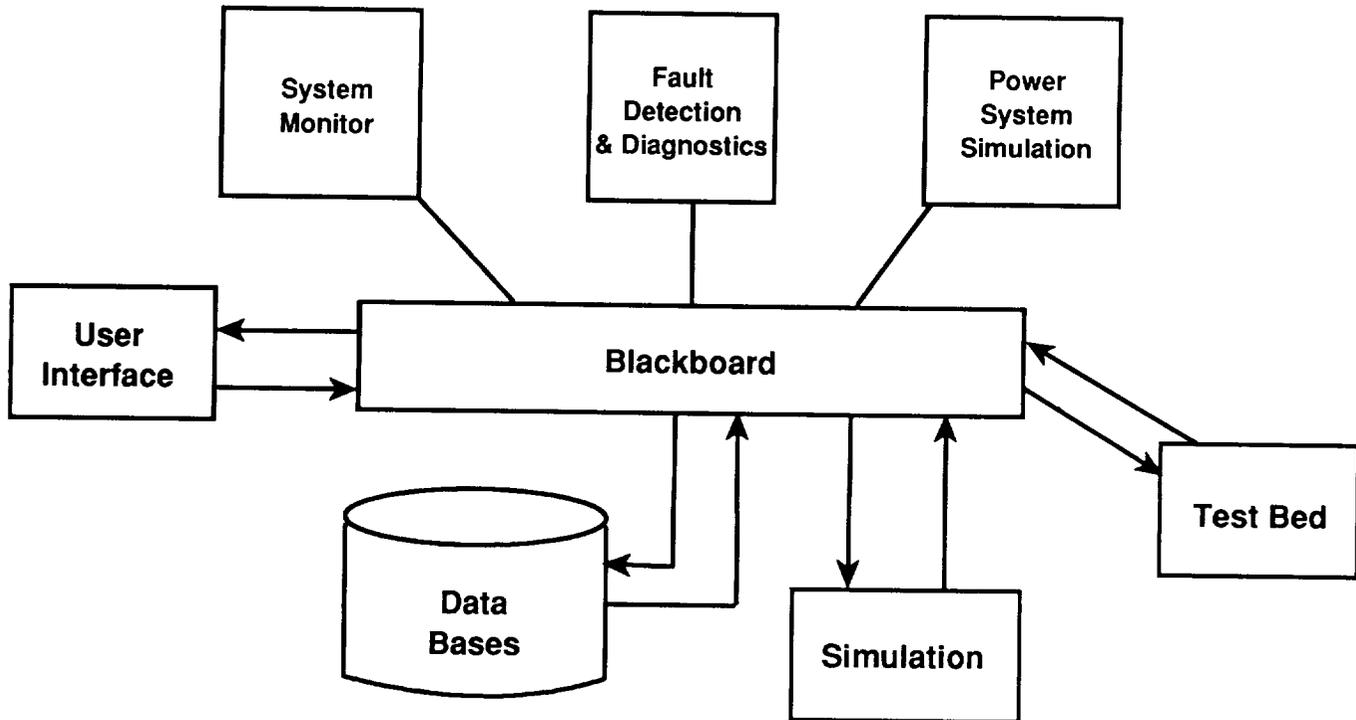


Figure 4. IPAC's Functional Environment.

Interaction with a Power Simulation

An option within IPAC is the ability to access a power system simulation. The IPAC interaction with a VAX-based power system simulation adds many capabilities to the overall program. Data from a simulation could be used as input to the diagnostic part of IPAC for monitoring and detection purposes. There has been research in the areas of power system simulation interaction with expert systems [1,5]. Additionally, interfacing with a UNIX environment, autonomous data reduction, and database protocol have been studied [9,10].

With these considerations in mind, the simulation interface to IPAC is being developed. As shown in Figure 3, a DECnet interface that will be used to access the simulation. Once the power system simulation is accessed, a power system scheduler and load flow can be executed. A template is available to specify the topology of the simulation including:

- Number of busses
- Bus connections
- Available power
- Number and type of loads

By employing the template and activating the simulation executive, a user could then receive power system information through databases. This information would include component measurements and failure modes that were detected. Power system scenarios on the simulator could be compared with data from the SPEL testbed and IPAC for evaluating predictive simulations. Work is continuing in this area.

Summary

The flexibility and adaptive diagnostic capabilities provided by an autonomous computer system, that combines classical control with artificial intelligence techniques, is being explored in the Rocketdyne SPEL. IPAC, the expert system portion of this hybrid environment, is being expanded as knowledge is gained on failure modes and hardware responses in the SPEL testbed. It is already emerging that the classical algorithms are superior in some categories of faults, while the heuristic rule base of IPAC appears better suited to others. As experimentation continues, both with real hardware and interactive simulations, the viability of this combined approach will be better determined. The results to date show considerable promise, and appear to offer a productive line of research.

References

1. Brady, M., G. E. Moody, and D. A. Scharnhorst, "A Symbolic Programming Approach to Intelligent Data Reduction," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, Washington, D. C., Vol. 1, August 1989, pp. 139-141.
2. Fesq, L. M. and A. Stephan, "On-Board Fault Management Using Modeling Techniques," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, Washington, D. C., August 1989, Vol. 1, pp. 225-230.
3. Gholdston, E. W., D. F. Janik, and G. Lane, "A Diagnostic Expert System for Space-Based Electrical Power Networks," Proceedings of the 23rd Intersociety Energy Conversion Engineering Conference, August 1988, Vol. 3, pp. 401-406.
4. Gholdston, E. W., D. F. Janik, and K. A. Newton, "A Hybrid Approach to Space Power Control Utilizing Expert Systems and Numerical Techniques," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, August 1989, Vol. 1, pp.
5. McKee, J. W., N. Whitehead, and L. Lollar, "Considerations in the Design of a Communication Network for an Autonomously Managed Power System," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, Washington, D. C., August 1989, Vol. 1, pp. 111-116.
6. Spier, R. J. and M. E. Liffing, "Real-Time Expert Systems for Advanced Power Control (A Status Update)," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, Washington, D. C., August 1989, Vol. 1, pp. 123-128.
7. Riedesel, J., "A Survey of Fault Diagnosis Technology," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, Washington, D. C., August 1989, Vol. 1, pp. 183-188.
8. Walls, B., "Exercise of the SSM/PMAD Breadboard," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, Washington, D. C., August 1989, Vol. 1, pp. 189-194.

9. Weeks, D. J. and S. A. Starks, "Advanced Automation Approaches for Space Power Systems," IEEE Computer Applications in Power, October 1989, pp. 13-17.
10. Wilhite, L. D, S. C. Lee, and L. F. Lollar, "Data Acquisition for a Real Time Fault Monitoring and Diagnosis Knowledge-Based System for Space Power System," Proceedings of the 24th Intersociety Energy Conversion Engineering Conference, Washington, D. C., August 1989, Vol. 1, pp. 117-121.



A Study on Diagnosability of Space Station ECLSS

S. Padalkar, W. Blokland, and J. Sztipanovits
Vanderbilt University, Nashville, TN.

ABSTRACT

This research demonstrates the use of the Multigraph Architecture (MGA) for studies on the Environment Control and Life Support System (ECLSS). The objective of this effort has been the following: (1) to create an updated set of models of the Potable Water Subsystem (PWS) by using the graphical model building tools of the Multigraph Programming Environment (MPE), (2) to derive a real-time alarm simulator from the models, and (3) to demonstrate the effects of sensor allocation on the diagnosability of the PWS. This work may serve as a preliminary study for the detailed analysis of the sensor allocation and diagnosability problems in the ECLSS.

PROBLEM STATEMENT

Real-time monitoring and diagnostics is a necessary component of critical and complex systems such as ECLSS. Their task is to provide an extensive fault detection capability combined with diagnostics of reasonable depth. The function of ECLSS requires the diagnostic system to operate continuously in a dynamically changing environment. Diagnostic hypotheses must be generated in an evolving fault scenario so as to allow corrective measures that can prevent the development of catastrophic failures.

Construction of real-time diagnostic systems is not straightforward. The diagnostic reasoning is subject to time constraints, has to indicate modeling errors, and must be robust enough to handle sensor failures. In dynamic systems where the presence of feedback loops are inevitable, the diagnostic system must apply temporal reasoning. These and similar requirements make the application of "associative" approaches - that associate patterns of observations (symptoms) with the underlying causes - unfeasible [1].

Model-based approaches have the potential of solving the challenging problems of real-time diagnostics. Core components of model-based diagnostic systems are: (1) well defined model of the system to be diagnosed, and (2) diagnostic reasoning algorithm, which interprets the observations in the context of the model. The purpose of the observations is to detect anomalies in the system behavior. Fault detection algorithms use the incoming data from sensors that are allocated in the plant and process them to check whether various operational constraints are satisfied.

Performance of a particular diagnostic system depends on the number and reliability of sensors providing input data for the fault detection system. Having a large number of reliable sensors "close" to the possible fault sources makes the diagnostic reasoning simple and the result accurate. The obvious limitation in improving the diagnostic performance by increasing the number of sensors is cost. Sensors are usually scarce resources that have to be carefully allocated. A realistic design approach can not be based on the unlimited availability of these resources. On the contrary, the question is how to allocate a limited number of sensors of limited reliability so as to achieve the best diagnostic resolution?

The objectives of this study have been to create models for the fault diagnostic system of PWS, to use the models for simulating real-time alarm sequences, and to demonstrate the relationship between sensor allocation and diagnostic resolution.

MULTIGRAPH ARCHITECTURE

Over the last five years Vanderbilt has developed a technology for the design and implementation of model-based real-time systems. The basic principles used in the construction of the Multigraph Architecture (MGA) are the followings:

- Model-based systems include: (1) multiple-aspect models of the system to be monitored, controlled, and diagnosed, (2) models of the components of the monitoring, control, and diagnostic system itself, and (3) models of their interaction.
- The models form a new level of abstraction in the system architecture, and are actively involved in the system operation.
- Driven by the external events that are received by the system, the models are continuously interpreted and reinterpreted. Interpretation of the models of the monitoring/control/diagnostic system components generates their actual implementation on a lower level. The lower level implementations determine the actual behavior of the system at a given time. Therefore, a model-based system can dynamically change its behavior if the state of the model changes on a higher level. This is one of the ultimate advantages of the model-based approach. It provides a very high-level of flexibility in a very simple manner.
- Most of the complexity of the model-based system is concentrated on the models. The rest of the system is a set of very generic, highly "reusable" procedural code providing the run-time support for the system. Due to this, the development technology of model-based systems can be supported by extremely efficient graphical model building, and automatic model verification tools.

The Multigraph Architecture (MGA) includes two main components, a graphic programming environment (Multigraph Programming Environment, MPE) [2] and a parallel execution environment (Multigraph Execution Environment, MEE) [3]. MPE facilitates building and maintaining multiple-aspect models of heterogeneous systems. The iconic graphic editors of MPE represent models in the form of graphic pictures and generate their symbolic representation in terms of specific declarative languages. MEE is a macro-dataflow model, which provides a unified environment for the execution of the functional components of model-based systems. Important feature of the Multigraph technology is that executable systems are automatically generated from the models, providing very high level software productivity.

MODELING TECHNIQUE

Any complex electro-mechanical system such as the ECLSS can be viewed from many different aspects. One such aspect is its function, another is its structure. A hierarchical decomposition of the functional aspect yields the Hierarchical Functional Model (HFM). Similarly, a hierarchical decomposition of the structural aspect yields the Hierarchical Component Model (HCM). The Hierarchical Fault Model (HFAM) is derived in the context of the HFM and the HCM.

Hierarchical Functional Model

The individual nodes in the HFM are referred to as processes. A process in the HFM can be thought of as an abstract entity that performs a specific function. It is possible to model certain viewpoints of every process. They are the structural viewpoint and the failure propagation viewpoint.

Structural Viewpoint

The structural viewpoint of a process represents information about its internal structure. The following items are acquired for each process:

- **Input/Output Process Variables,**
- **Process States,**
- **Alarms, and**
- **Subprocesses.**

Failure Propagation Viewpoint

While performing its function, a process may violate some of its functional constraints due to the presence of faults. When such a violation occurs, the process is said to contain a failure-mode. The presence of a failure-mode can be detected by an alarm derived from a sensor associated with the process. A process can exist in a number of different states. The following items are acquired for each process:

- **Failure-modes.**
- **Failure-mode Alarm associations.** Each failure-mode alarm association has associated with itself, the list of process states in which it is active.
- **A fault propagation graph (FPG).** The FPG of a process denotes a causal relationship between its failure-modes and the failure-modes of its subprocesses. Each causal link in the FPG originates from a failure-mode of a subprocess and propagates to a failure-mode of either a subprocess or the parent. A causal link is weighted by four parameters, the fault propagation probability, the minimum fault propagation time, the maximum fault propagation time, and the list of process states in which the link is active. An AND type of causal link is also permitted in the FPG. This link has as ancestors, more than one subprocess failure-mode, and as destination more than one subprocess failure-mode and/or parent failure-mode. In case of the AND type of causal link, the fault propagation probability implies the probability of occurrence of the destination failure-modes after all the ancestor failure-modes have occurred. Similarly, the minimum and maximum fault propagation times are the minimum and maximum times during which the destination failure-modes will occur after all ancestor failure-modes have occurred.

Hierarchical Component Model

A component in the HCM is an actual piece of hardware that can assist a variety of processes in performing their functions. The source of faults in a system are any of the components in the HCM. Each component upon becoming faulty, can exhibit a number of failed-states. The following items are acquired for each component:

- **Component failed-states and**
- **Subcomponents.**

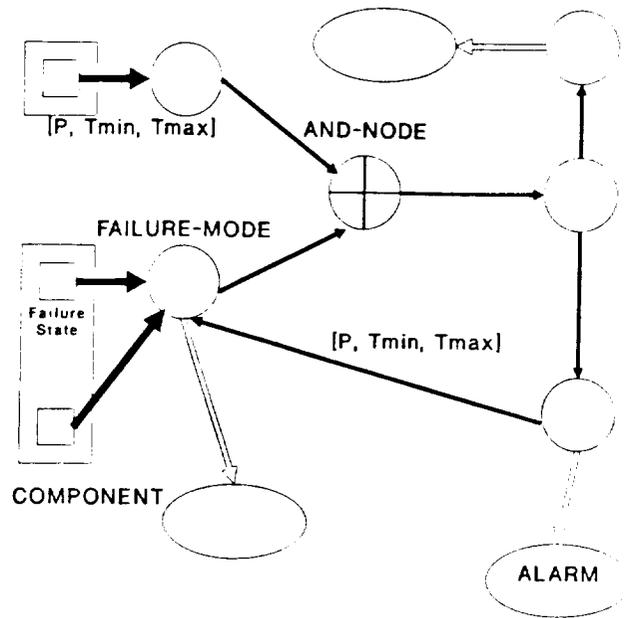


Figure 1: Process fault model

Function-Component Interactions

A failed-state in a component can lead to a number failure-modes being present in some of the processes in the HFM. The set of causal relationships between the failed-states of the components, and the failure-modes of the processes, is acquired. A causal link is weighted by three parameters, the fault propagation probability, the minimum fault propagation time, and the maximum fault propagation time.

Hierarchical Fault Model

The fault model of a process is the fault propagation viewpoint of the process model, and the set of causal links between the component failed-states and the process' failure-modes and failure-modes of all the process' existing subprocesses. The HFaM is the collection of all such process fault models. An example of a process fault model is shown in Figure 1.

REASONING TECHNIQUE

The occurrence of a fault in the system implies that a component or a set of components exhibit failed-states. The existence of failed-states leads to the existence of failure-modes in some processes. Among the set of existing failure-modes, those with associated alarms are detected by the ringing of those alarms. These ringing alarms start the diagnostic activity. The diagnostic reasoning technique selects the highest process in the HFM containing ringing alarms, and runs a Faulty Component Identification Algorithm (FCIA) on the fault model of the process' parent. The FCIA back-propagates along the ringing alarm failure-modes, and using structural and temporal constraints, identifies a set of possible fault source components [4]. The FCIA is guaranteed to produce a result in real-time because it possesses a polynomial time complexity. This time complexity is $O(n^3)$, where n is the number of existing failure-modes in the FPG of that process.

Certain factors affect the number of fault source candidates identified by the FCIA. A single fault case is where one component is responsible for the failures in the system. If a single fault case is identified by the

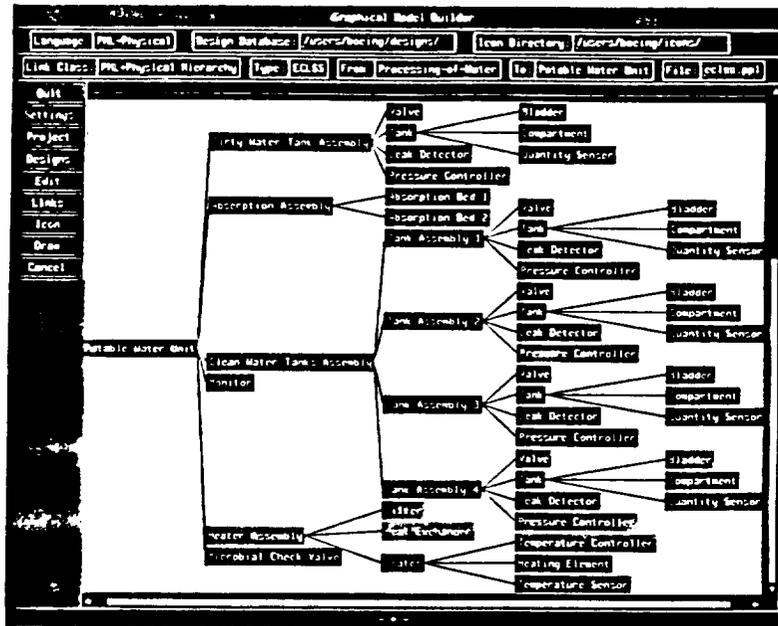


Figure 2: PWS Component Hierarchy

FCIA, the number of identified fault source candidates depends on the fault model. The primary factor is the number and allocation of alarms to failure-modes in the FPG. Since alarms are derived from sensors, the number of sensors allocated for diagnostic purposes affects the diagnostic result. If the number of sensors is low, then the number of identified fault source candidates can be high. On the other hand, if the the number of sensors are sufficient and are allocated to the right failure-modes in a FPG, the number of identified fault source candidates in a single fault case can be one. The other factor affecting the number of identified fault source candidates is the internal structure, i.e. the FPG and the component failure-mode associations, of the fault model of each process in the HFM.

PWS MODELS

The Potable Water System (PWS) of the ECLSS is decomposed from its structural aspect, resulting in a component hierarchy [5] shown in Figure 2. It is also decomposed in its functional aspect, resulting in a function hierarchy [5] shown in Figure 3.

SYSTEM DIAGNOSABILITY

Designers of modern industrial and space systems would like to use a lesser number of sensors for diagnostic purposes, in order to reduce costs. This is especially true in space systems because the total weight as well the total cost of sensors has to be reduced. Before they decide to eliminate a sensor, they would like to know the effects of its removal on the diagnostic result. Hence an approach that studies the diagnosability of a system given a particular sensor allocation is very useful to space system designers.

Some of the important terms in diagnosability studies are provided.

- **Sensor Allocation:** The number of sensors in the system, and the places in the system where they have been installed.

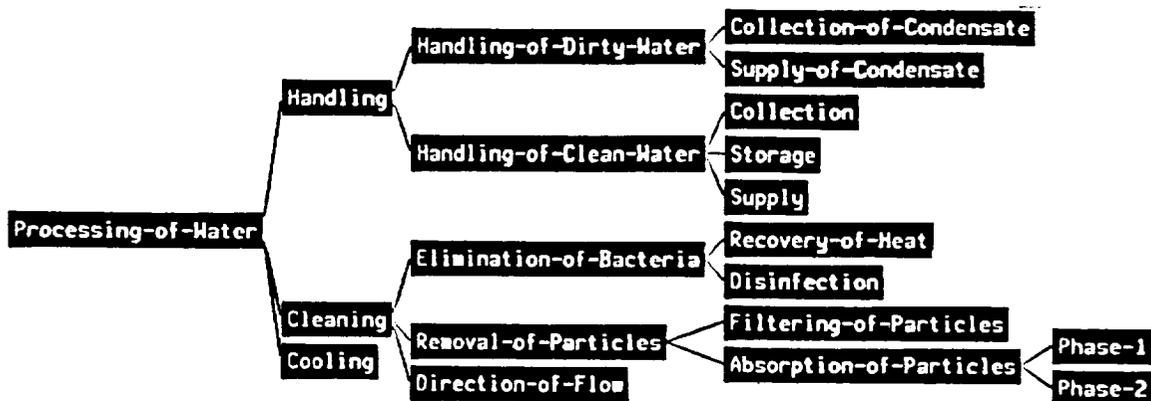


Figure 3: PWS Function Hierarchy

- **Diagnostic Result:** The set of fault source components diagnosed under no time constraints.
- **Time Constrained Diagnostic Result:** The set of fault source components diagnosed within a specific time constraint.
- **Single Component Diagnosability:** The diagnostic result when the single component in question is the fault source.
- **Multiple Component Diagnosability:** The diagnostic result when the specified multiple components are the set of fault source components.
- **Unique Diagnosability:** The diagnostic result is the same as the set of actual fault source components. In single fault cases the number of components in the diagnostic result and the actual set of fault source components is equal to one.

The single component diagnosability, multiple component diagnosability, and unique diagnosability definitions can be extended to include the case of time constrained diagnostic result.

Diagnosability Studies

A variety of studies can be performed on the system in order to determine its diagnosability. Given a particular sensor allocation, the diagnosability of components can be obtained. By eliminating a sensor from the allocation, the differences in the diagnosability of components can be determined. If the differences are minimal the designer has the option of eliminating that sensor. Finally, a study can be performed to find an optimal sensor allocation that provides unique diagnosability.

Simulation Method

The simulation method is used to demonstrate some aspects of the diagnosability studies. This method involved developing a fault simulator that simulates actual fault scenarios. The fault model of the system,

which includes the sensor allocation, serves as the internal database for the simulator. This internal database is automatically generated during the diagnostic runtime system generation phase by the diagnostic interpreter. The simulator is stand-alone program that can accept as input, a set of fault source components with their selected failed-states. The pattern of ensuing alarms is derived, and is simulated in real-time. This real-time alarm pattern serves as the input to the diagnostic process. A diagnostic result is generated on receipt of the simulated alarm pattern.

The diagnosability of any component can now be observed for any given sensor allocation. For a given sensor allocation, the fault scenario or the real-time alarm pattern with the component under question being the fault source is obtained. This real-time alarm pattern serves as the input for the diagnostic process. The generated diagnostic result is stored and compared with the diagnostic results from other sensor allocations. The effects of different sensor allocations on the diagnosability of a component can be studied in this manner.

Another use of such a simulator is to study the effects of adding or removing a sensor on the diagnostic process. Initially the designer can select a sensor of interest. The simulator determines all the components that are affected by the presence and absence of this sensor. Two fault simulations are generated for each of the affected components, one with the sensor being present and the other with the sensor being absent. The diagnosability of each component in both cases is determined on the basis of the two simulations. A statistical measure based on the diagnostic results, can then be used to determine whether or not the sensor should be retained in the system.

The simulation method is an effective first step in demonstrating the need for and effectiveness of diagnosability studies. It is quite effective in determining the diagnosability of a component. It sometimes proves useful in helping a designer decide whether to retain a sensor in the system or not. However, the process of finding the diagnostic result for all components before and after the removal of a particular sensor is very cumbersome and time consuming. The same process has to be repeated for any other sensor allocation. If the designer wants to determine an optimal sensor allocation that will achieve unique diagnosability in single fault cases, he/she has to simulate all possible sensor allocations before finding the answer. This process has an exponential time complexity because the number of possible sensor allocations can be $O(2^n)$, where n is the total number of failure-modes in the system. This complexity is clearly unacceptable for large-scale systems. An analytical method of a more manageable time complexity is definitely required for solving the problems of system diagnosability.

PWS DIAGNOSABILITY STUDY

An example that presents the effects on system diagnosability when a sensor is removed, is presented. The relevant portions of the fault model of the chosen process, Absorption-of-Particles is shown in Figure 4.

Initial Sensor Allocation

The sensor allocation for the failure-modes of the Absorption-of-Particles process and the failure-modes of its subprocesses follows:

- Sensor Allocation in Subprocess **Phase-1**.
sudden-bad-absorption: Sensor S0, Alarm A0. *slow-bad-absorption*: Sensor S1, Alarm A1. *reverse-absorption*: Sensor S2, Alarm A2.
- Sensor Allocation in Subprocess **Phase-2**.
sudden-bad-absorption: Sensor S3, Alarm A3. *slow-bad-absorption*: Sensor S4, Alarm A4. *reverse-absorption*: Sensor S5, Alarm A5.

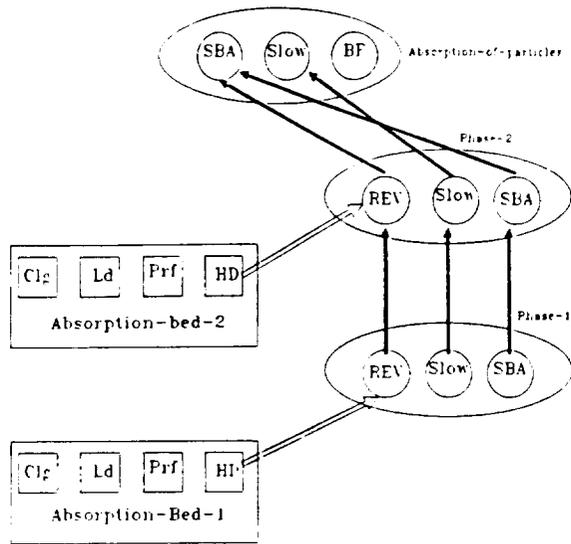


Figure 4: Absorption-of-Particles fault model

- **Sensor Allocation in Process Absorption-of-P articles.**

sudden-bad-absorption: Sensor S6, Alarm A6. *slow-bad-absorption*: Sensor S7, Alarm A7. *bad-flow*: Sensor S8, Alarm A8.

The single component diagnosability for the two associated components when only one failed-state is simulated at a time is:

- **Absorption Bed 1:**

Heat Damaged: Absorption Bed 1 Heat Damaged. *Loaded*: Absorption Bed 1 Loaded. *Perforated*: Absorption Bed 1 Perforated. *Clogged*: Absorption Bed 1 Clogged and Absorption Bed 2 Clogged.

- **Absorption Bed 2:**

Heat Damaged: Absorption Bed 2 Heat Damaged. *Loaded*: Absorption Bed 2 Loaded. *Perforated*: Absorption Bed 2 Perforated. *Clogged*: Absorption Bed 2 Clogged and Absorption Bed 1 Clogged.

The explanation of the diagnostic result is provided for the cases of absorption bed 1 and 2 being in the heat damaged failed state. If the absorption bed 1 is heat damaged, the alarms generated by the simulation in order are: A2, A5, A6. The alarm A2 is associated with failure-mode reverse-absorption in the phase-1 process. There exists a failure-propagation link from this failure-mode to the reverse-absorption failure-mode in the phase-2 process. Alarm A5 is associated with the reverse-absorption failure-mode in the phase-2 process. There exists a failure-propagation link from this failure-mode to the sudden-bad-absorption failure-mode in the Absorption-of-Particles process. Alarm A6 is associated with the sudden-bad-absorption failure-mode in the Absorption-of-Particles process. The FCIA decides that alarm A2 is the primary alarm since this alarm could have caused alarms A5 and A6. The ancestor components of the failure-mode associated with the primary alarm are the initial set fault source hypothesis. In this case it is absorption bed 1 in heat damaged failed state. After ascertaining the fact that if absorption bed 1 was heat damaged the times at

which alarms A2, A5, and A6 could have been generated are not in conflict with the actual generation times, the absorption bed 1 in heat damaged state is returned by the FCIA as the diagnostic result.

If absorption bed 2 is heat damaged, the alarms generated by the simulation in order are: A5, A6. The FCIA decides that alarm A5 is the primary alarm. The possible ancestor components of the failure-mode associated with this alarm are absorption bed 1 being heat damaged and absorption bed 2 being heat damaged. However, if absorption bed 1 was heat damaged, alarm A2 would have been generated. Since this did not happen, absorption bed 1 being heat damaged is removed from the list of identified fault source components. Therefore, the final diagnostic result is absorption bed 2 being heat damaged.

Final Sensor Allocation

The sensor S2 associated with alarm A2 and attached to failure-mode reverse-absorption in the phase-1 process is removed. The resultant sensor allocation follows:

- Sensor Allocation in Subprocess **Phase-1**.
sudden-bad-absorption: Sensor S0, Alarm A0. *slow-bad-absorption*: Sensor S1, Alarm A1. *reverse-absorption*: No sensor.
- Sensor Allocation in Subprocess **Phase-2**
sudden-bad-absorption: Sensor S3, Alarm A3. *slow-bad-absorption*: Sensor S4, Alarm A4. *reverse-absorption*: Sensor S5, Alarm A5.
- Sensor Allocation in Process **Absorption-of-P articles**.
sudden-bad-absorption: Sensor S6, Alarm A6. *slow-bad-absorption*: Sensor S7, Alarm A7. *bad-flow*: Sensor S8, Alarm A8.

The single component diagnosability for the two associated components is when only one failed-state is simulated at a time is:

- *Absorption Bed 1*:
Heat Damaged: Absorption Bed 1 Heat Damaged and Absorption Bed 2 Heat Damaged. *Loaded*: Absorption Bed 1 Loaded. *Perforated*: Absorption Bed 1 Perforated. *Clogged*: Absorption Bed 1 Clogged and Absorption Bed 2 Clogged.
- *Absorption Bed 2*:
Heat Damaged: Absorption Bed 2 Heat Damaged and Absorption Bed 1 Heat Damaged. *Loaded*: Absorption Bed 2 Loaded. *Perforated*: Absorption Bed 2 Perforated. *Clogged*: Absorption Bed 2 Clogged and Absorption Bed 1 Clogged.

A graphical tabulation of the diagnosability results in both cases of sensor allocation is shown in Figure 5.

The explanation of the diagnostic result is provided for the cases of absorption bed 1 and 2 being in the heat damaged failed state. The sensor S2 has been removed, therefore alarm A2 no longer exists, and the failure-mode reverse-absorption in the phase-1 process has no associated alarm. If either the absorption bed 1 or the absorption bed 2 was heat damaged, the alarms generated by the simulation in order are: A5, A6. The alarm A5 is diagnosed as the primary alarm. The ancestor components of this alarm are absorption bed 1 being heat damaged and absorption bed 2 being heat damaged. Since there is no alarm A2 in the fault model, the hypothesis absorption bed 1 being heat damaged has to be retained. Hence if either of the two components is the fault source, the diagnostic result contains both of them.

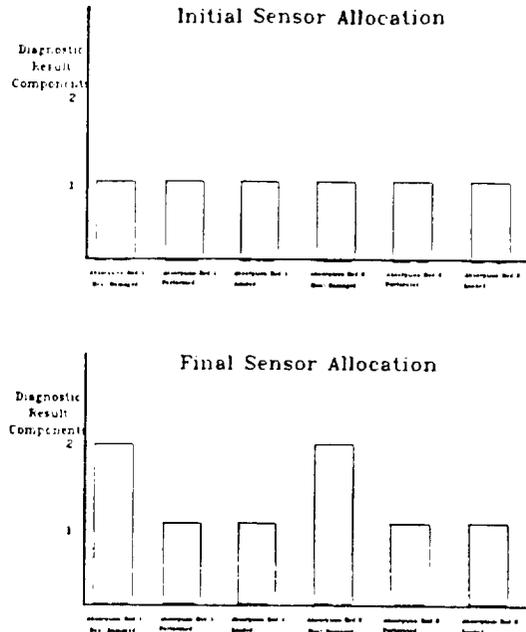


Figure 5: Diagnosability results

1 Acknowledgements

This work was supported by a grant from the BOEING Aerospace Co. Huntsville, AL. The authors would like to thank J. Ray Carnes of BOEING Computer Services for his insight, advice, and expertise. Charles Reeves, Roger Vonyouanne, and Art Brown provided the domain expertise for developing the various models.

References

- [1] T. Laffey, P. Cox, J. Schmidt, S. Kao, and J. Read, *Real-Time Knowledge-Based Systems*, A.I. Magazine, Spring 1988, pp 27-45.
- [2] G. Karasi, *Declarative Programming using Visual Tools*, Internal Report, Electrical Engineering, Vanderbilt University.
- [3] C. Biegl, *Design and Implementation of an Execution Environment for Knowledge Based systems*, Ph.D. Dissertation, Electrical Engineering, Vanderbilt University, August 1988.
- [4] S. Padalkar, G. Karsai, and J. Sztipanovits, *Graph-Based Real-Time Fault Diagnostics*, Proc. of the Fourth Conf. on A.I. for Space Applications, pp 115-123, Huntsville, AL, 1988.
- [5] W. Blokland, S. Padalkar and J. Sztipanovits, *Study and Approach to Artificial Intelligence Testing: Modeling the Potable Water System of the Environmental Control And Life Support System (ECLSS)*, Vanderbilt University, 1990.

Attitude Determination and Control System (ADCS) Maintenance and Diagnostic System (MDS)

A Maintenance and Diagnostic System for Space Station Freedom

David Toms and George D. Hadden
Honeywell Systems and Research Center
3660 Technology Drive
Minneapolis, Minnesota 55418

Jim Harrington
Honeywell Space Systems Operation
13350 US Highway 19 South
Clearwater, Florida 34624

Abstract

This paper describes the Maintenance and Diagnostic System (MDS) that is being developed at Honeywell* to enhance the Fault Detection Isolation and Recovery system (FDIR) for the Attitude Determination and Control System on Space Station Freedom. The MDS demonstrates ways that AI-based techniques can be used to improve the maintainability and safety of the Station by helping to resolve fault anomalies that cannot be fully determined by built-in-test, by providing predictive maintenance capabilities, and by providing expert maintenance assistance.

The MDS will address the problems associated with reasoning about dynamic, continuous information versus only about static data, the concerns of porting software based on AI techniques to embedded targets, and the difficulties associated with real-time response.

We have built an initial prototype of this MDS and are continuing development on it. The prototype executes on Sun and IBM PS/2 hardware and is implemented in the Common Lisp; further work will evaluate its functionality and develop mechanisms to port the code to Ada.

Introduction

The systems on Space Station Freedom (SSF) will require high levels of reliability, fault tolerance, and ease of maintainability. Our ability to satisfy these requirements can be enhanced by using sophisticated software techniques to further automate tasks. Many of the software techniques appropriate in this area grew out of, or are being developed by, research in artificial intelligence.

This automation is particularly useful for *remote* systems on SSF. Remote systems are defined as systems physically separated from the crew quarters. The Attitude Determination and Control System (ADCS) is one such remote system. All of the ADCS hardware, except for the Standard Data Processor (SDP), resides on a section of truss assembly approximately 25m from the habitable modules. The only means of repairing the ADCS hardware is for a crew member to perform extra-vehicular activity (EVA) or to use one of the SSF robot systems, such as the Flight Telerobotic Servicers (FTS) (not as dangerous as going EVA, but time-consuming and tedious).

Since the ADCS is a remote system, causing maintenance to be expensive, it becomes important to be able to reliably determine which ADCS element is faulty and to be able to predict when a failure might occur (or that a component is degrading). Accurate fault information is required to plan a maintenance action and optimize the time spent on the maintenance action.

* Work on the MDS is funded by NASA contract NAS 9-18200, through McDonnell Douglas Space Systems Company contract #87916007. The McDonnell Douglas Technical Monitor is Gerry Ridout.

The Maintenance and Diagnostic System (MDS) could provide additional information to the Fault Detection, Isolation, and Recovery (FDIR) system to aid in the fault isolation process. The MDS derives this additional information by reasoning about data in a global way; this is not possible with built in test (BIT) within an individual ADCS Orbital Replaceable Unit (ORU). The MDS software will collect, store, and analyze the health monitoring data from each active ADCS element to provide additional information for fault analysis. In addition to fault isolation, it is expected that the MDS will also be able to verify and flag ADCS BIT false alarms.

One of the goals of the MDS is to predict when a fault might occur. Fault prediction is a difficult problem to solve. However, most of the components within the ADCS are modifications of standard hardware, and a significant database of failure modes exists. The successful implementation of predictive capabilities in the MDS would provide significant benefits:

- Additional information is gained, leading to the identification of a faulty element (thus reducing the time and cost of fault isolation).
- Maintenance actions can be planned for a time when a crew member is “in the neighborhood” (thus reducing maintenance cost).
- Replacing or repairing the system before it malfunctions could prevent further damage to other systems as well as providing more reliable systems operation.
- Safety could be enhanced by predicting and preventing problems that would have occurred during a critical maneuver such as berthing.

Once a fault has been isolated, the MDS can be used as a maintenance aid, to instruct crew members on test and replacement procedures.

This paper discusses the overall architecture of the MDS, the general approach being used to develop each of its capabilities, and specific plans for the prototype currently under development.

ADCS/Space Station Freedom Overview and Maintenance Philosophy

The software for the ADCS will reside in the Guidance Navigation and Control (GN&C) SDP. Besides the software in the GN&C SDP, the ADCS also consists of six Control Moment Gyros (CMGs), three Star Trackers (STs), and three Inertial Sensor Assemblies (ISAs). The ISAs and STs are the sensors supporting the Attitude Determination Function (ADF); the CMGs are part of the

Attitude Control Function (ACF), along with the Reaction Control System (RCS). Functional representation of the ADCS within the GN&C SDP is portrayed in Figure 1.

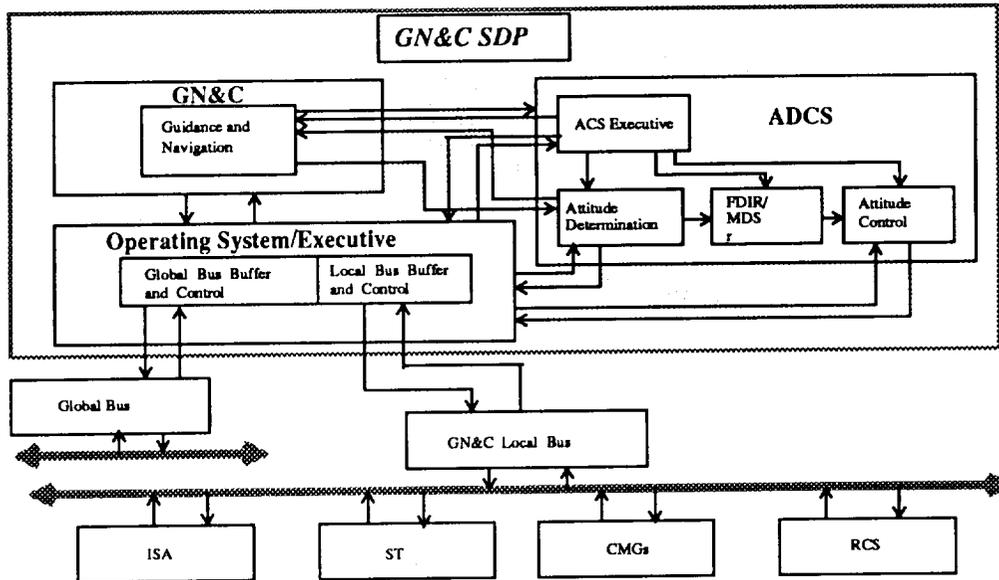


Figure 1. Functional Overview of MDS in ACDS

The minimum configuration of the ADF is the operation of one ISA and one ST; during docking, berthing, and other critical operations, additional ISAs and STs would be brought on line. The primary purpose for running in the minimum configuration is to conserve power.

The primary error checking parameter currently designed for the ADF, besides the BIT, is the chi-squared product (χ) formed from the primary Kalman filter used for attitude determination in the ADF. The BIT could indicate that both the ISA and ST are working within performance bounds; however, the χ could indicate significant error in the calculations based on the data returned from the ISA and ST. Without additional information, it is not possible, however, to determine which of the two elements is causing the problem.

Another problem that occurs in a Kalman filter is that one of the units (either the ST or the ISA) is degrading slowly. The Kalman filter can continue to add a bias to the ORU's data that will keep the system (and the χ) within established limits masking the degraded performance of the ORU. The coefficients of the χ can be monitored along with additional data generated by the Kalman filter to detect when this condition is arising.

ORUs are defined at a very high level: STs, ISAs, and CMGs are each considered an ORU. The CMG is a special case in that it has two additional ORUs mounted on it (the CMG electrical assemblies). By definition, no diagnosis or repair will be done below the ORU level. We do, however, expect to use the internal structure of the ORUs in our reasoning and not treat them as black boxes.

As one would expect, there are stringent requirements for mean time between failure (MTBF), error detection, and false alarms. BIT is required to detect 70% of the errors in the continuous mode and 95% in the commanded mode. It is required that each system must generate no more than one

false alarm every 3000hr. (This, of course, raises questions such as what exactly is considered a system, and whether or not an error in the BIT firmware counts as a failure or a false alarm.)

MDS Approach and Top Level Architecture

Our approach to both diagnostics and prognostics uses the concept of a *global view* of the ADCS. By this we mean the ability to view the state of all ORUs in the ADCS, monitor communications, correlate health status from multiple systems, and reason about this information. This global view of the system will give us the capability to address problems that BIT might not handle adequately, much as in the previously mentioned chi-square test.

The MDS is being designed to operate on and with a real-time control system, and will receive a regular “flow” of information from continuous BIT and discrete signals. It has to choose what information to reason about, handle changing information, and know when to look for and request additional information.

This real-time aspect of the MDS implies the need for it to handle real-time concerns such as speed, responsiveness, and timeliness. Further, this data has associated time-values, which complicates the data with temporal concerns such as degraded validity over time, etc.

The MDS needs access to this continuous information flow within the ADCS in order to maintain this global view. However, there are hardware constraints in SSF; memory and processing cycles available on the SDPs will be limited. This combination of requirements and constraints led us to partition the MDS architecture into two basic functional areas. The first is an *on-line* system, embedded within the SDP. The functions of the on-line MDS are intelligent data gathering, data compression, data transmission, and some reasoning. The other is an *off-line* reasoning system. The functions of the off-line MDS are diagnostics, prognostics, and maintenance aiding. In order to obtain the required computing power and mass storage capability it is likely that the off-line MDS would be on the ground. Figure 2 shows a general architectural view of the MDS system.

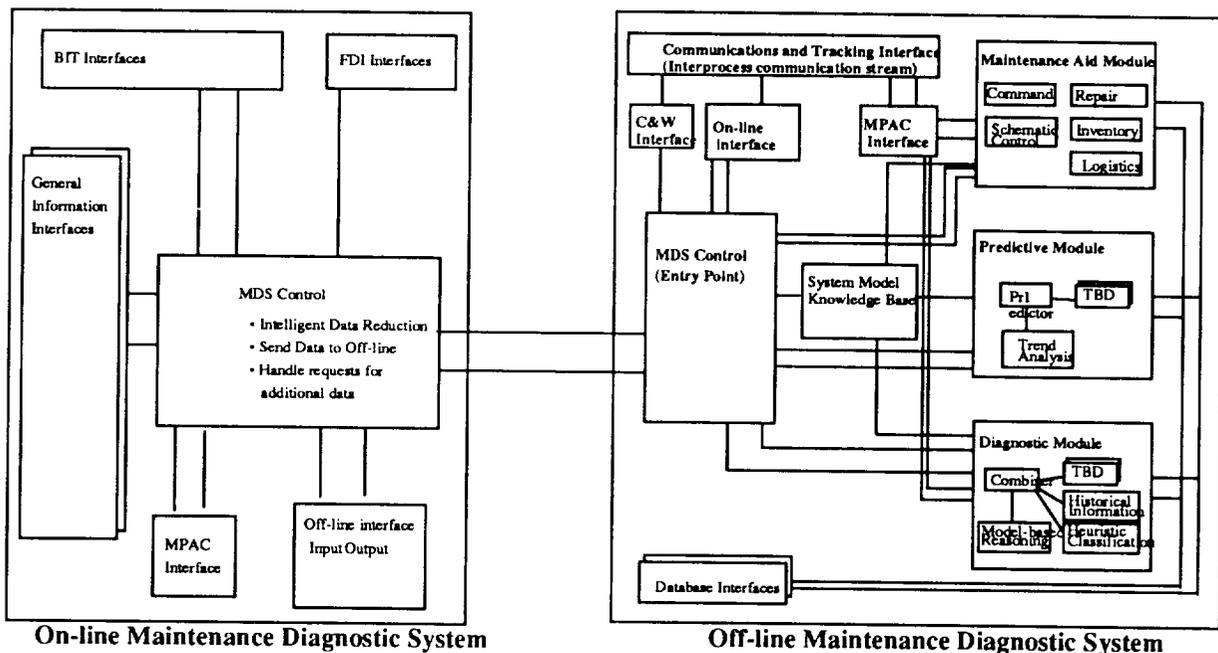


Figure 2. Basic On-line and Off-line Architecture

This partitioning provides the MDS with a natural mechanism to both handle data from a continuously operating real-time process and to reason in depth about the state of the system at any given time. Basically, the on-line MDS handles the real-time monitoring, screening and intelligently screening data generated by the ORUs. The current approach to this uses a dynamic thresholding approach, much as used in [Washington and Hayes-Roth, 1989]. The off-line MDS handles diagnostics, prognostics, and maintenance aiding, described in following sections.

Diagnostics

A diagnostic technique can be evaluated on its breadth and depth of coverage, its robustness and ability to degrade gracefully with ambiguous knowledge, its generality and adaptability to changing system configurations, its efficiency and ease of modification with the target system, and its ability to handle uncertainty. Possible diagnostic techniques include fault trees, explicitly coded fault models, heuristic classification, simulation, functional/causal modeling, and structural modeling.

All these techniques can weigh decisions based on various information sources, including historical information, FMEA (Failure Modes and Effects Analysis), and individual component reliability (predicted MTBF). Discussions of diagnostic approaches can be found in many places, including [Chu, 1988] and [Pau, 1986].

In general, heuristic classification, or “classical” expert systems, have many benefits. A key benefit is that they can capture “experiential” knowledge that is hard to come by and may be hard to characterize in other solutions. They provide an explicit separation of control logic and diagnostic knowledge. This makes them relatively maintainable, and often the diagnostic logic can be re-used. They can be easily extended, which might make them attractive given long-term extensibility goals. And, they provide viable solutions in situations where no model of the system exists.

There are, however, problems associated with the classical, shallow knowledge-based expert system diagnostic approach. First, they require a large amount of knowledge engineering up front to make them robust, because they require large amounts of explicit domain-specific information to solve diagnostic problems. Also, the requirement for explicit knowledge about fault situations makes these systems relatively “fragile” at the edges; if a situation isn’t at least partially foreseen, the system could abruptly fail rather than use existing information to degrade gracefully. This reliance on prior knowledge of a system’s explicit faults in given situations can cause problems, especially in newly designed systems, or any other system that might not have sufficient explicit fault knowledge immediately available.

Reasoning methods based on connectivity models overcome many of the problems with knowledge engineering and robustness, given that design information will be available with which the model can be built. The major drawback can be that connectivity models fail to characterize all types of fault models completely. This can result in overly ambiguous or incorrect answers in specific types of cases.

Functional qualitative models can alleviate these problems but have higher initial development costs and higher maintenance costs. They are also susceptible to the possibility of computational intractability due to the combinatorial explosive search. Another problem is ensuring the “correctness” of the model, particularly in the cases where good models of the process do not exist (e.g. in some of the ISA failure modes).

We believe that hybrid approaches based on multiple reasoning techniques are the most effective. A particular approach may rely on one dominant, controlling technique and draw on and weight the outputs of other techniques as needed. Hybrid approaches have been used with excellent success in diverse areas. This hybrid diagnostic approach, using heuristic and design-derived fault mode knowledge as it is available, can provide optimal diagnostic capability.

The MDS is being designed with a hybrid architecture, able to combine a variety of reasoning techniques. Using various kinds of knowledge is especially effective in a system that has been designed with both established, proven technologies and new design, such as ADCS. The ISAs, for example, are based on previous designs for which extensive performance information is available, the CMGs are based on a prototype design developed for Marshall Space Center, and the Star Trackers are based on an existing design.

Also, since this architecture can combine an arbitrary number of reasoning methods, it provides an excellent framework to handle extensions.

Prognostics

Earlier in this paper we discussed the reasons for using prognostic reasoning (also referred to as predictive maintenance). Briefly, these are: 1) faster fault isolation, 2) ability to plan maintenance, 3) reduction of the possibility of cascading faults, and 4) enhanced probability of operation (providing greater safety) during critical maneuvers.

Traditionally, predicting device failure has been notoriously difficult, for both mechanical and electronic devices (although mechanical devices are usually easier). The primary reason for this is that it is difficult to correlate the long-term behavior (fault signature) with a particular failure mode.

There are however a few exceptions which our system looks for explicitly. One example we have found is the Ring Laser Gyroscope (RLG) in the ISA. This device has a predictable relationship between input current and output power. There are situations in which monitoring the characteristics of this relationship will allow us to predict future problems with the lasers. Another example is the fault signatures of the CMG wheel bearings. We believe that particular types of vibration patterns might indicate upcoming problems.

The interesting thing from an artificial intelligence point of view is that there is no model for these effects. That is, we cannot evaluate these systems from first principles. Naturally, this makes a model-based approach to this problem relatively difficult. We are in the process of acquiring other predictive maintenance scenarios from experts in the ADCS, but our expectation is that these will be rare and dissimilar.

Our initial approach to prognostics, therefore, is somewhat "brute force." The on-line system has two major functions. The first function is to look specifically for BIT status signatures which might indicate future problems and make appropriate recommendations to the off-line system. The second thing is to send a minimal (but sufficient) set of device status data to the off-line system so that it can do more extensive trending analysis.

Maintenance aiding

Once a failure has been found, or if a problem is predicted, a correction may or may not be in order. The MDS has the context to support access to intelligent, on-line technical information for maintenance aiding, and the initial prototype demonstrates this capability.

One of the ways the MDS provides this access is to present a checklist to the user for subtasks that need to be done to perform the maintenance action. As these are accomplished, the user uses the mouse to check off a box. If there are ordering constraints among the subtasks, the MDS forces them to be met. In other words, the user can not move on to a new subtask unless the ones that must be done before it have been finished. In the initial prototype, the user can also ask for help, at which point a bitmap is displayed.

Another capability that we might add is preventive maintenance (PM) aiding. A PM procedure is usually complex enough that it is necessary to provide the user with more than a simple checklist. Normally, one finds that the procedure can vary quite a bit depending on what is found during the PM [Hadden]. At this point, as far as we know, no PM procedures have been specified for sub-systems in SSF, the philosophy being that they will not be necessary. If this continues to be the case, then obviously there will not be a need to provide PM aiding.

In the future, we will extend the initial prototype in many ways. First, we will expand the help options so that the user will be able to control how much help is presented and at what level. We will also allow the user to request explanations for what is being done, explanations both in the sense of "Why is this necessary, given the current goals?" and "How do I perform the requested action?" This latter extension might take the form of a finer-grained action list.

MDS Prototype

We are currently developing a prototype MDS to:

- Test our approaches in an environment "similar" to the fielded environment
- Implement an approach to fit the MDS within the ADCS architecture
- Provide a mechanism to get feedback from engineers on the program, and test diagnostic and prognostic components

The prototype MDS provides the same "look and feel" that a crew member would have on SSF, given the current NASA user interface definitions. These user interface definitions encompass the display, interactions with the crew, and, to some degree, control. Figure 4 shows the basic look of the current prototype.

We expect the final prototype to reside on two machines, one Sun workstation and one IBM PS/2, reflecting off-line and on-line MDS components, respectively, and to communicate directly with the real-time ADCS simulation systems (possibly on 386-class personal computers). Figure 5 shows this configuration.

We chose the IBM PS/2 to run the off-line software to remain consistent with the Software Support Environment (SSE) tools currently under development. The simulations that the prototype will communicate with are being developed to functionally represent the CMGs, STs, and ISAs.

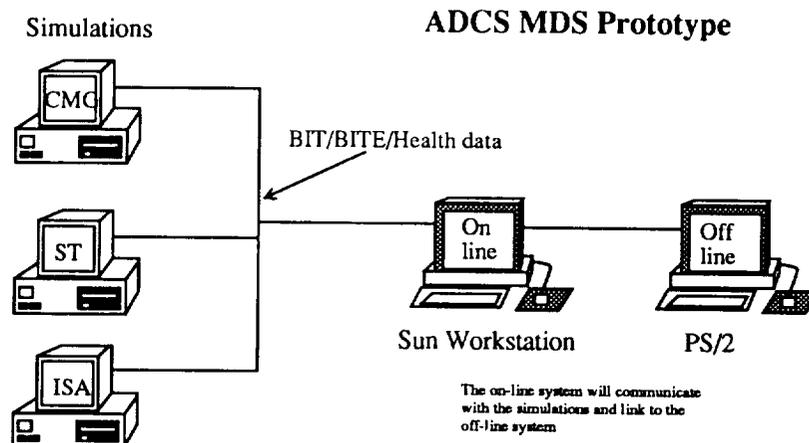


Figure 5. MDS Prototype

The initial demonstration prototype also resides on two machines, but does not yet communicate directly with simulations. Instead, we have simulated the ORUs (or rather the salient outputs of the ORUs) in software on the Sun workstation.

The prototype software for both our Sun and PS/2 environments is primarily implemented in Common Lisp and the Common Lisp Object System (CLOS), which provides us with many existing Lisp tools, the ability to use foreign functions (languages other than Lisp), and standard user interface calls.

This use of object-oriented software was partly driven by long-term goals for SSF for diverse and nearly unconstrained growth while in orbit, which imposes requirements for easy reconfiguration and transparency to continued technological upgrades.

Conclusions and Future Plans

Work on the prototype ADCS MDS has just reached the implementation stage. An infrastructure to simulate continuous ORU output data, an on-line implementation that intelligently screens data, a prognostics capability that acts on known predictive signatures in an individual way, and a maintenance aiding system are currently operational.

This MDS prototype is now being used to incrementally develop the diagnostic approaches and further prognostic capabilities.

Upcoming use and evaluations will determine the specific direction we take, but several areas will definitely be developed in the near future, including:

- Use of the MDS with the high-fidelity ORU simulators. This is a needed step to properly develop and evaluate the system.
- Development of a translation path from Common Lisp to Ada or the standard SSF expert system shell. This is needed to show compatibility with SSF embedded software standards.

In the future, the MDS replacement procedures may be useful in allowing an autonomous robot to perform the maintenance operations with only supervision (in contrast to tele-operation) by a crew member.

References

- [1] Murugesan, S., "Considerations in Development of Expert Systems for Real-Time Space Applications" *Fourth Conference on Artificial Intelligence for Space Applications*, November, 1988, pp 487-494
- [2] Hadden, George, "Mentor, An Expert System for Preventive Maintenance" Proceedings of the 1986 Annual Conference of the Instrument Society of America
- [3] Washington, R. and Hayes-Roth, B., "Input Data Management in Real-Time AI Systems" *Proceedings of the Eleventh Joint International Conference on Artificial Intelligence*, August 1989, pp250-255
- [4] Fink, P.K. and Lusth, J.C., "The Integrated Diagnostic Model - Towards a Second Generation Diagnostic Expert System" *Proceedings of the Air Force Workshop on Artificial Intelligence Applications for Integrated Diagnostics*, July 1986, pp188-197
- [5] Pau, L.F., "Survey of Expert Systems for Fault Detection, Test Generation and Maintenance" *Expert Systems*, April 1986, pp100-110
- [6] Chu, Sai-Cheong, "Approaches to Automatic Fault Diagnosis: A Critical Evaluation" *Proceedings of the AI in Armaments Workshop*, March 1988
- [7] McCown, P. and Lewy, D., "APU MAID - A Diagnostic Expert System Using Heuristic and Causal Reasoning" *AUTOTESTCON '87*, November 1987, pp371-375

KNOWLEDGE-BASED SYSTEMS AND NASA'S SOFTWARE SUPPORT ENVIRONMENT

Tim Dugan, PRC, Houston
Cora Carmody, PRC, Houston
Kent Lennington, LMSC, Houston
Bob Nelson, NASA, Reston

ABSTRACT

This paper describes a proposed role for knowledge-based systems within NASA's Software Support Environment (SSE). The SSE is chartered to support all software development for the Space Station Freedom Program (SSFP). This includes support for development of knowledge-based systems and the integration of these systems with conventional software systems. In addition to the support of development of knowledge-based systems, various software development functions provided by the SSE will utilize knowledge-based systems technology.

KEY WORDS

Ada, Development Tools, Knowledge-Based Systems, Software Engineering, Space Station

INTRODUCTION**NASA'S SOFTWARE SUPPORT ENVIRONMENT**

In order to provide a complete and consistent support environment for software development for the Space Station Freedom Program (SSFP), NASA initiated the Software Support Environment (SSE) project. The SSE is intended to support the life-cycle management of operational software (both ground and flight software) as well as the life-cycle management of the software for the SSE itself. The SSE software consists of both Commercial Off-The-Shelf (COTS) applications and custom software augmented by methods, procedures, standards, documentation, and training materials to support users of the environment. The SSE will evolve over the life cycle of the SSFP in order to be responsive to its user community's needs.

THE RELATIONSHIP OF KNOWLEDGE-BASED SYSTEMS TO THE SSE

The uses of knowledge-based systems technology within the SSE may be classified as either:

- support for the development of knowledge-based systems products and the integration and deployment of these products with conventional software systems, or
- support for SSE operations.

Figure 1 illustrates the relationship of knowledge-based systems technology to the SSE and SSFP.

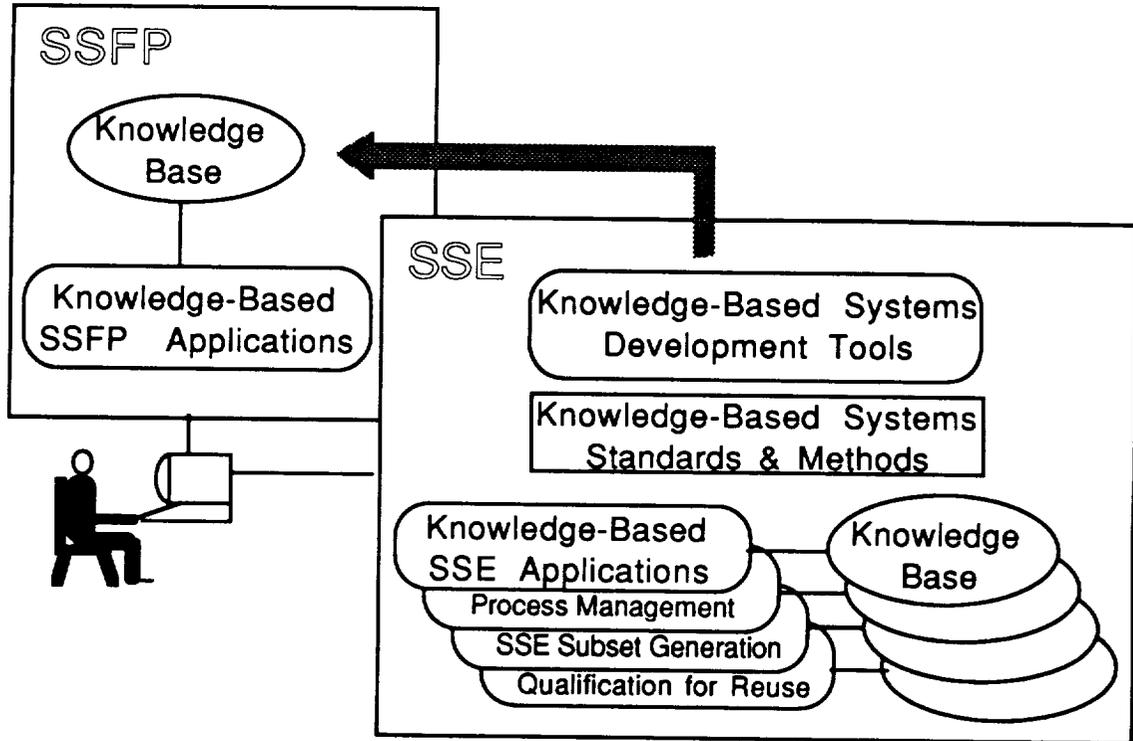


Figure 1. Two Uses of Knowledge-Based Systems Technology in the SSE

The remainder of this paper describes SSE support for and use of knowledge-based systems in more detail.

KNOWLEDGE-BASED SYSTEMS DEVELOPMENT

Increased feasibility of knowledge-based systems technology within an operational environment has produced a significant demand for automated development support. A large variety of commercial software systems are available that support the development and deployment of knowledge-based systems. In order to provide criteria for selecting existing tool sets or developing custom tools, an initial set of functional requirements for these tools has been developed. Based upon these requirements, a preliminary design of the knowledge-based support tools was also

produced. In order to supplement the knowledge-based systems development environment, the SSE will provide support for an expert systems development methodology. A description of the status of these efforts follows.

REQUIREMENTS ON THE SSE FOR A KNOWLEDGE-BASED SYSTEMS DEVELOPMENT ENVIRONMENT

The SSE contractor has defined a set of functional requirements for the SSE's knowledge-based systems development environment. These requirements are documented in detail in [LMSC-1] and summarized in [LMSC-4].

According to these requirements, the SSE will provide the necessary support to develop and deploy knowledge-based applications. This support includes capabilities for developers of applications to choose from a variety of knowledge representation schemes and reasoning strategies. The five broad categories of requirements include knowledge representation, reasoning strategies, external software integration, development, and delivery.

Knowledge Representation Requirements. The knowledge-based systems support tools will provide several different integrated knowledge representation schemes. One of these is production rules which are constructed from conditional patterns, priorities, and resulting actions. External procedures may be invoked from either the pattern or action part of a rule. A second form of knowledge representation is performed through object manipulations. Objects will support strong typing, multiple inheritance, uncertainty, truth maintenance, and external procedure calls.

Reasoning Strategies Requirements. The knowledge-based systems support tools will provide for reasoning using forward- and backward-chaining of rules. Reasoning strategies will include reasoning about uncertain data, hypothetical reasoning, constraint checking, and access to external procedures through rules and objects.

External Software Integration Requirements. The knowledge-based systems support tools will provide interfaces to external software systems. This includes the capability to execute functions attached to objects and rules which are implemented in other languages, particularly Ada. The knowledge-based systems support tools will include an interface to the knowledge base which may be accessed asynchronously from other languages.

Development Requirements. The knowledge-based systems support tools will provide an environment for developing, testing, debugging and validating knowledge-based applications in an

integrated environment. This environment includes an integrated knowledge base editor with incremental compilation capabilities. Debugging support includes the capability to trace the execution of rules within a knowledge-based system through the use of break points. Users will have the ability to interactively query a knowledge base and change its content. Additionally, users will have access to on-line help and explanation facilities. And, finally, the environment will provide the capability to save its current state in a file to be recreated at a later time.

Delivery Requirements. A user of the knowledge-based systems support tools will have the capability to deploy knowledge-based applications onto SSFP flight hardware and ground elements of the Space Station Information System. Knowledge-based systems may execute either as an interactive adviser or as an embedded subsystem of another application. Deployed versions of a knowledge-based application need not contain code for functions of the knowledge-based system support tools which are not required by a particular application. There will be no built-in restrictions on the size of a knowledge base and no restrictions on the number of users of the static portion of a knowledge base.

Note that deployment of knowledge-based applications onto SSFP flight hardware implies certain performance constraints. The exact nature of these requirements has not been investigated at this time.

THE PRELIMINARY DESIGN FOR KNOWLEDGE-BASED SYSTEMS SUPPORT TOOLS

A high-level preliminary design of the SSE software is presented in [LMSC-3]. This document provides a context for the design of the knowledge-based systems support environment which is described in more detail in [LMSC-1]. This document furthers the perspective that, in terms of an overall software system, a knowledge-based application is just another software component. However, the development of a knowledge-based system distinguishes it from traditional software systems in that the use of interactive development environments provide the capability to build successively more sophisticated prototypes of the desired system. Once a prototype of a knowledge-based system has reached a sufficiently mature form, it may be deployed in the form of an Ada package which may be embedded in a larger system.

Figure 2 depicts a Buhr-style diagram portraying high-level view of the knowledge-based systems support tool. The major components include a development interface which allows the user to interact with the environment to develop and deploy an application and an inference engine which supports object management, pattern matching, and execution management.

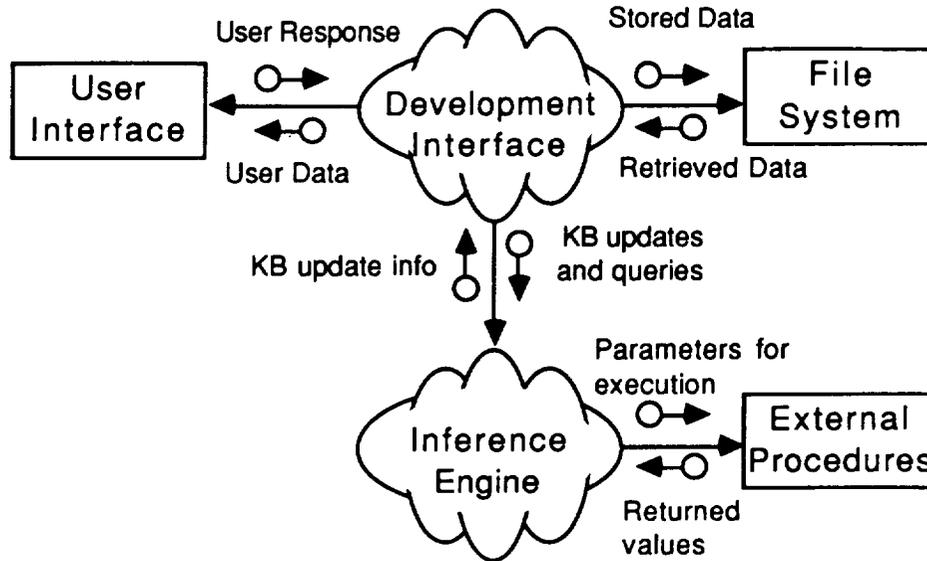


Figure 2. Knowledge-Based Tool Preliminary Design

METHODS SUPPORT FOR EXPERT SYSTEMS DEVELOPMENT

The SSE contractor is in the process of defining methodologies to be supported by the SSE as described in [LMSC-2]. Due to important differences in the life cycle of expert systems, the contractor has also prepared a supplementary document [LMSC-1] that describes the differences between the methodological support for traditional software systems and that needed for expert system development. The supplement emphasizes iterative knowledge acquisition, the use of prototypes, validation techniques, and user interfaces for expert systems.

IMPLEMENTATION OPTIONS

Some of the options for implementing SSE knowledge-based systems development tools that are under consideration include the purchase of an existing COTS product or the enhancement of a public domain product known as CLIPS. The final decision will depend in part on a comparison of the short-term benefits for developing or modifying a given product to meet the requirements compared to the long-term benefits of using that product. Adherence by the tool to any existing or emerging standard should play a role in the selection.

USE OF KNOWLEDGE-BASED SYSTEMS BY THE SSE

The SSE is itself a large, complex software system. As such, it provides many opportunities for utilizing knowledge-based

systems technology. Although not all areas where knowledge-based systems may be used have been identified, some of the candidate areas are described below.

SUBSET GENERATION

One important function of the final SSE system is the capability to automatically configure new Software Production Facilities (SPFs) and individual projects at SPFs. This capability is referred to as subset generation because each SPF contains some subset of the complete set of SSE capabilities. The subset generation capability is somewhat similar to DEC's XCON as described in [Waterman] and [Barker]. XCON provides for automated configuration of VAX computer systems. XCON performs these configurations in a fraction of the time that a technician would take to perform the same task. XCON is implemented in the language OPS5.

Much like XCON, the SSE subset generator will produce candidate configurations for SPFs and individual projects at SPFs based upon the needs at that site. Human users will then analyze the resulting SPFs to determine the validity of the configuration. Once it has been determined reasonable, the subset generator function will provide at least partial automated support in configuring the SPF.

SOFTWARE LIFE-CYCLE PROCESSES

A second area that is a candidate for use of knowledge-based technologies is the automation and control of software life-cycle processes. This goal evolved from a desire to coordinate unconnected tools into an integrated development environment such as that described in [Bisiani et al]. These goals were combined with the SSE goal of providing consistent life-cycle management for all SSFP software producing a desire to manage development processes through process programming. Some research indicates that a mix of procedural and rule-based approaches are best for describing such processes [Taylor]. To supplement this view, there is an emerging body of theory relating to process programming which is described in [Tully].

The management of life-cycle processes as they are applied to development products will require an interface between the knowledge-based application and the SSE project object base. One recommended approach outlined in [McKay] is to provide a standardized interface to the project object base through the use of an Information Resource Dictionary System (IRDS). The IRDS standard (ANSI X3H4) allows the development of a semantic model of a project object base in an entity-relationship form.

REUSABLE LIBRARY

A critical need for the SSE is to provide a useful library of reusable components. Unlike typical reusable libraries, the SSE reusable library will include not just reusable code but reusable objects from all phases of the software life cycle. The two primary uses of knowledge-based systems technology as it applies to reusable libraries are the qualification of components for inclusion in the reusable library and the search for reusable components based upon functional requirements.

CONCLUSION

SUMMARY

This SSE support for development of knowledge-based systems has been described and references for further information has been provided. Additionally, some of the planned uses of knowledge-based systems within the SSE have been described.

FUTURE DIRECTIONS

Future directions for support of knowledge-based systems are currently dependent on the recognition of new requirements based upon the needs of SSFP contractors. Future directions for the utilization of knowledge base technology within the SSE are also dependent on the determination of need as well as the identification of emerging uses of knowledge-based systems that support software engineering. Some examples include:

- Project management support as described in [Bimson]
- Performance tuning of the SSE System such as is described for UNIX® in [Samadi].

Work Pack Contractors may influence the directions of the SSE by contacting an SSE support representative or submitting an SSE Change Request.

®UNIX is a registered trademark of AT&T Bell Laboratories.

REFERENCES

- [Barker] V. Barker, D. O'Conner, "Expert Systems for Configuration at Digital: XCON and Beyond," *Communications of the ACM*, Vol. 32, No. 3, March 1989
- [Bisiani] R. Bisiani, F. Lecoual, V. Ambriola, "A Tools to Coordinate Tools," *IEEE Software*, Vol. 5, No. 6, November 1988.
- [Bimson] K. Bimson, L. Burris, "Assisting Managers in Project Definition: Foundations for Intelligent Software Management," *IEEE Expert*, Vol. 4, No. 2.
- [LMSC-1] LMSC Internal Document DC-301, Expert Systems Preliminary Design Documents, August 4, 1989.
- [LMSC-2] LMSC F255442, *SSE System Methods Manual*, Lockheed Missiles and Space Corporation, September 2, 1988
- [LMSC-3] LMSC F255458, *SSE Preliminary Design Document* Lockheed Missiles and Space Corporation, July 17, 1989
- [LMSC-4] LMSC F255472, *SSE Detailed Requirements Specification*, Lockheed Missiles and Space Corporation, July 17, 1989
- [McKay] C. McKay, "Adding Knowledge base technology to Object Management Systems: Addendum 5.4.7," Software Engineering Research Center/High Technologies Lab, University of Houston at Clear Lake, no date.
- [Samadi] B. Samadi, "TUNEX: A Knowledge-Based System for Performance Tuning of the UNIX Operating System," in *IEEE Transactions on Software Engineering*, Vol. 15, No. 7, July 1989.
- [Taylor] F. Taylor et al., "Foundations for the Arcadia Environment Architecture," ACM, 1988
- [Tully] C. Tully, Editor, "Representing and Enacting the Software Process," *Proceedings of the 4th International Software Process Workshop*, ACM SIGSOFT Software Engineering Notes, Vol. 14, No 4, June 1989.
- [Waterman] D. Waterman, *A Guide to Expert Systems*, Addison-Wesley, 1986

**Space Station Freedom ECLSS -
A Step Toward Autonomous Regenerative Life Support Systems**

Brandon S. Dewberry
National Aeronautics and Space Administration
Marshall Space Flight Center
Information and Electronic Systems Laboratory
Software and Data Management Division
Systems Software Branch EB42
MSFC, AL 35812

Abstract

The Environmental Control and Life Support System (ECLSS) is a Freedom Station distributed system with inherent applicability to extensive automation primarily due to its comparatively long control system latencies. These allow longer contemplation times in which to form a more intelligent control strategy and to prevent and diagnose faults. The regenerative nature of the Space Station Freedom ECLSS will contribute closed loop complexities never before encountered in life support systems.

A study to determine ECLSS automation approaches has been completed. The ECLSS baseline software and system processes could be augmented with more advanced fault management and regenerative control systems for a more autonomous evolutionary system, as well as serving as a firm foundation for future regenerative life support systems. Emerging advanced software technology and tools can be successfully applied to fault management, but a fully automated life support system will require research and development of regenerative control systems and models.

The baseline Environmental Control and Life Support System utilizes ground tests in development of batch chemical and microbial control processes. Long duration regenerative life support systems will require more active chemical and microbial feedback control systems which, in turn, will require advancements in regenerative life support models and tools. These models can be verified using ground and on orbit life support test and operational data, and used in the engineering analysis of proposed intelligent instrumentation feedback and flexible process control technologies for future autonomous regenerative life support systems, including the evolutionary Space Station Freedom ECLSS.

Introduction

When the baseline Space Station Freedom is completed in 1999, a milestone in regenerative life support systems for human exploration will be achieved. It will be the first complete on-orbit closure of drinking water, wash water, and oxygen loops which is a step along the way to independence from Earth's resources. Long duration interaction of these subsystems and humans in a micro-gravity environment will be a major achievement for the Environmental Control and Life Support System (ECLSS) designers and engineers.

A great deal of knowledge will be gained for development of advanced regenerative life support systems by the lessons learned in ground and on orbit operation of the baseline ECLSS. This knowledge and familiarization with the characteristics of regenerative life support systems will serve as a basis for advanced automation of these systems.

At present, regenerative life support knowledge is contained in the designs and operational data of previous life support systems and tests, the development documentation of the baseline ECLSS, and in the experiences of the systems engineers and medical experts involved. Many of the previously unknown complexities encountered in a long duration regenerative life support system will be manually dealt with in the baseline ECLSS. The underlying causes of these complexities must be understood and modelled in order to build more robust life support systems for long stays in space.

This objective of this report is to clarify approaches to automation of the Environmental Control and Life Support System. In doing so, the report progresses as follows:

- The Baseline ECLSS - an overview of life support system requirements and the Freedom Station's approach to meeting these requirements.
- Autonomous Regenerative Life Support System - methods of building upon the ECLSS for increased near and long term life support knowledge and automation.

Three automation areas will be discussed. The first two, advanced fault detection, isolation, and recovery (FDIR) and intelligent instrumentation are possible augmentations to the baseline ECLSS software with minimal hardware impacts. The third, advanced regenerative life support control systems, is an approach for using the baseline ECLSS as a test bed for development of more autonomous, long duration life support systems including that of the evolutionary Freedom Station.

The Baseline Environmental Control and Life Support System

Life Support Systems are required to provide the habitable environment for the crew and life sciences payloads. This environment includes water for drinking and washing, and atmospheric gasses. Previous life support systems have typically met these requirement by maintaining sufficient supplies of pressurized gases and fluids, though closed loop options have been investigated (5).

Process Description

The Temperature and Humidity Control, Water Recovery Management, and Air Revitalization Subsystems aboard the Space Station combine to meet the water and air supply requirements as in figure 1. These requirements are met by closing the air and water loops to an extent never before implemented in space. Even so, the control system is essentially open loop, a batch filtering process. Little chemical or microbial data is fed back into the control system for use in adjusting flexible processes for maximum efficiency.

The system is tested on the ground for sufficient cleaning and recycling set types and levels of fluids in the air and water, and is periodically verified on orbit using batch laboratory analysis procedures. This alone, the actual integration of these multiple interacting subsystems to meet specified requirements, will be a great achievement. Lessons learned in the on-orbit integration of these batch processing systems will be invaluable in determining micro-gravity interactions and recombinations of chemical and microbial constituents throughout the revitalization systems.

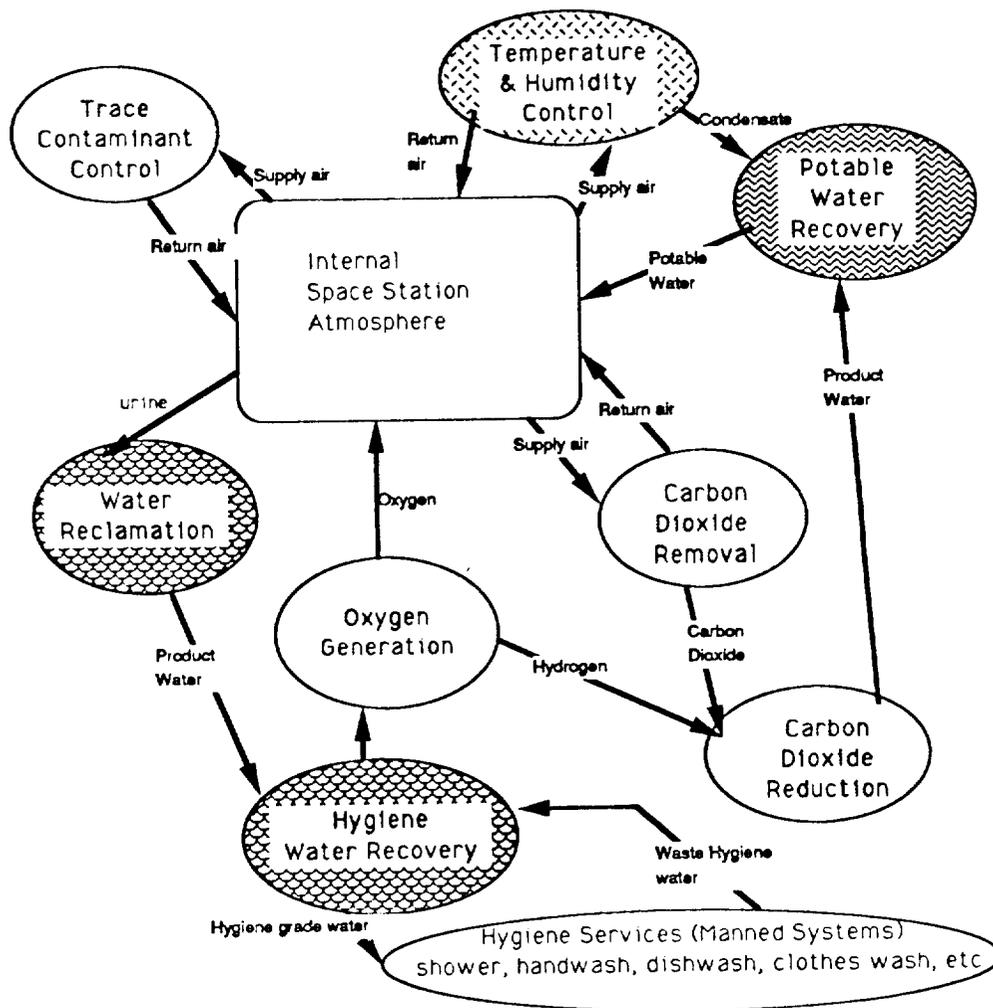


Figure 1 - The ECLSS Functional Interactions

Software Architecture

An overview of the approximate software architecture for the ECLSS is illustrated in figure 2. Two software processes which were determined prime candidates for automation are 2.3 Real-time & Off-line Subsystem FDIR (Fault Detection, Isolation, and Recovery), and 2.4 Component Performance and Trend Analysis. Both of these processes will contain parts initially in the ECLSS Ground Sustaining Engineering, with migration on board when flight data management resources permit.

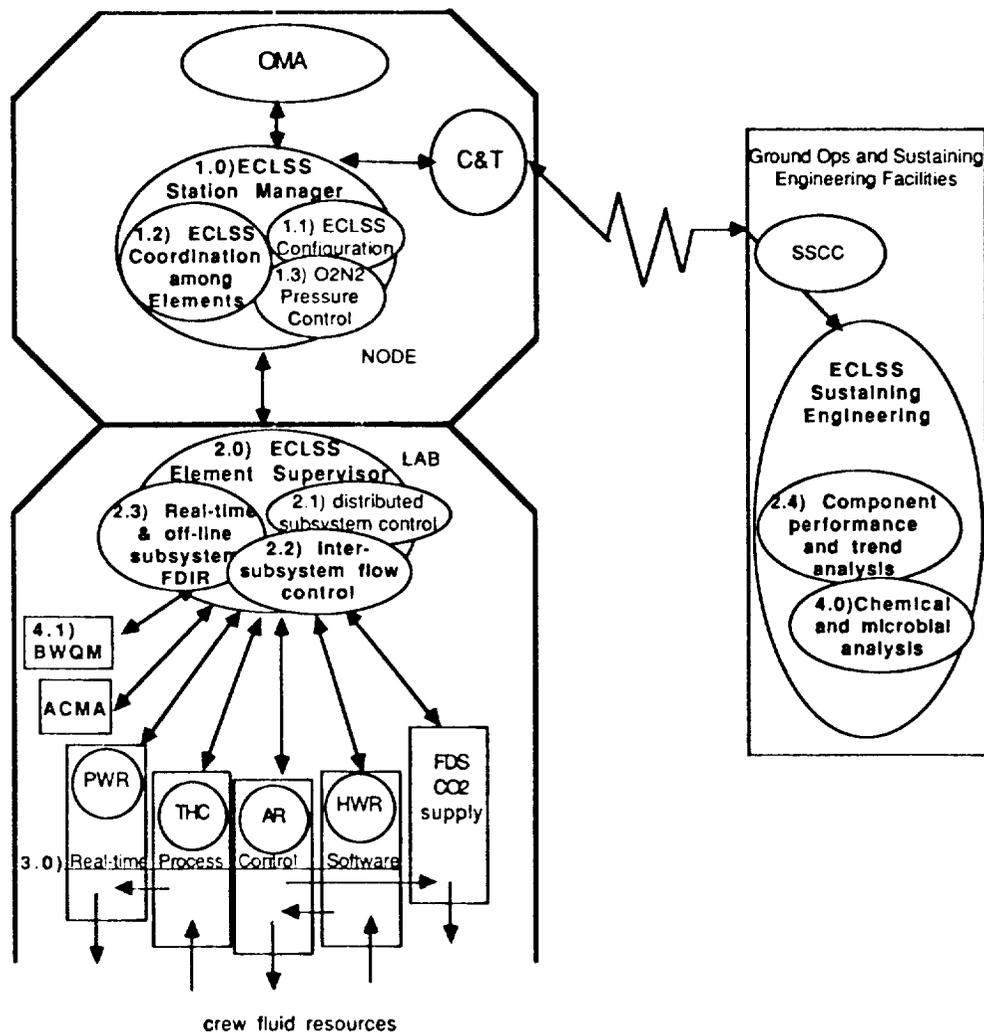


Figure 2 - ECLSS Software Architecture

Baseline Chemical and Microbial Monitoring and Control

Most of the active chemical and microbial monitoring and control of the ECLSS will be a batch manual process involving the crew, ground support labs, and life sciences equipment. There are a few exceptions, confirming that the addition of regeneration technologies introduces requirements for in line chemical and microbial instrumentation and control technologies in the life support system, previously more the domain of life sciences.

The potable recovery, urine pre-treatment, and hygiene recovery processes in the baseline system use some chemical and microbial feedback control. Trace organic carbon levels are monitored on their outputs, and if the measurement violates the required maximum, the water is not released downstream but is cleaned again by the systems' filters. Iodine and pH levels are also monitored at this point for adjustment of the assembly processing. This data will also be available for ground support monitoring and trend analysis. These process control water quality monitors do not provide distinction between organics or their individual levels (4). Also, the viral and inorganic constituents such as metals gathered from the THC slurper or the cabin air ducts are not determined or used by the control system.

More in depth analysis of the chemical constituents of the water is available by manual sampling of the output of the Water Recovery Management processes with on board mass spectroscopy in the Batch Water Quality Monitor (BWQM) and laboratory analysis. Also, extensive ground testing of water samples returned from the Space Station will be used to verify and support the ECLSS. Anomalies require manual replacement of filters or complete flushes and reinitialization of the system, little automatic adjustment of flexible process control subsystems, such as adjustment of chemical additive amounts, flexible filter sizes, or trace contaminant processes, is available.

Autonomous Regenerative Life Support System

In general, future autonomous regenerative life support systems, including the evolutionary ECLSS, will be required to supply water and air, within specific chemical and microbial limits, for extended durations without crew or ground support adjustment. The control system and plant will be intelligent and robust enough to autonomously withstand unexpected crew and payload anomalies. These requirements will be achieved with a minimal set of instrumentation and processing assemblies.

These requirements may be met by augmenting the baseline ECLSS with various technologies. Software hooks, and hardware scars in the baseline will be necessary to minimize the impact of integrating these technologies after Assembly Complete. Increased automation of the ECLSS is possible, but evolution to complete automation, defined as above but requiring some simple unit replacement occasionally, may not be feasible due to the degree of fundamental process adjustments and control strategies required. But the ECLSS can be used to dramatically increase the state-of-the-art in regenerative life support systems.

A block diagram illustrating the components of an Autonomous Regenerative Life Support System is provided in figure 3.

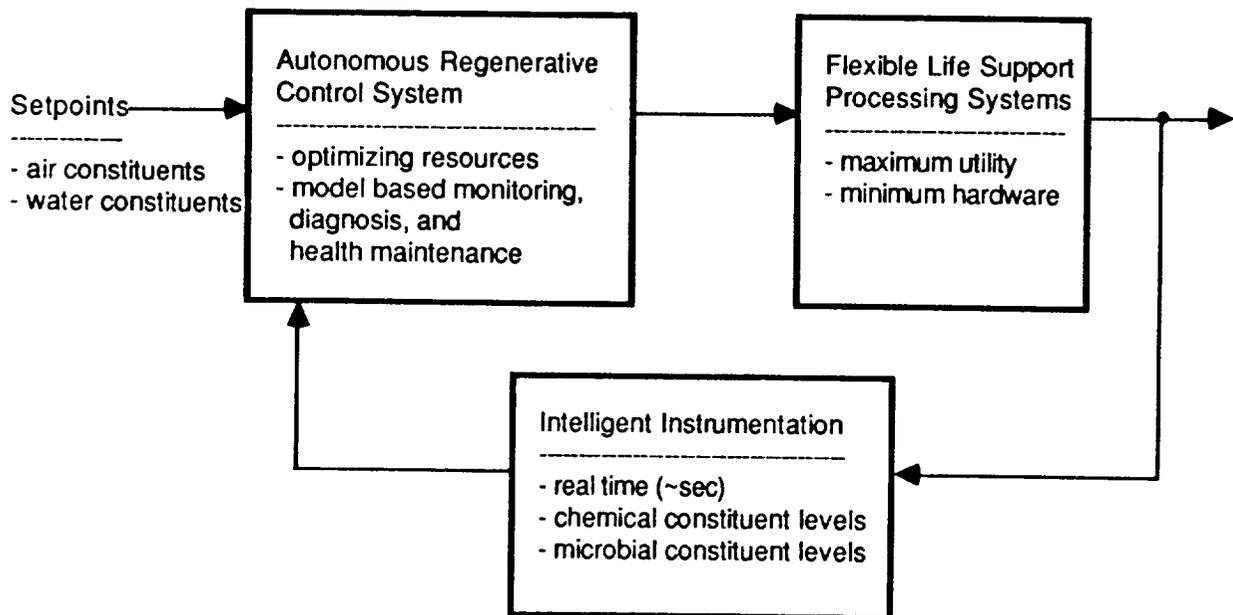


Figure 3 - Advanced ECLSS Block Diagram

The major areas for advanced technology application are:

- Automatic Fault Detection, Isolation, and Recovery (FDIR) and Health Maintenance
- Advanced Intelligent Instrumentation
- Flexible Life Support Processing Systems
- Regenerative Life Support Modelling and Analysis Systems

Automatic Fault Detection, Isolation, and Recovery (FDIR) and Health Maintenance

There are several advantages to beginning ECLSS automation with upgrades in the automatic fault detection, isolation, and recovery (FDIR) and health maintenance (failure prediction and prevention) processes. These processes are software oriented and theoretically, software is the most flexible part of the system and most amenable to upgrades.

FDIR and health maintenance processes require the implementation of emerging software technologies. These processes can be verified in the ground support environment and migrated to the flight ECLSS to increase the Station's flight autonomy. This approach to increasing ECLSS autonomy is described in (2) and (3) and will be the focus of the ECLSS Advanced Automation Project.

Structural and functional models of the ECLSS subsystem processes can be used to diagnose and isolate failures. Model based approaches to diagnosis are computationally intensive but perform autonomous, in-depth diagnosis of faults. The process control nature of the ECLSS allows the use of emerging model based reasoning tools in automating the system, while storing knowledge in component model form (7). The system also may be upgraded for automatic diagnosis of regeneration analysis with the future inclusion of chemical and microbial transfer equations.

Model based fault diagnosis using these models is analogous to the biomedical engineer who notices a trend in the chemical and microbial levels at a certain point and uses a mental model of the behavior of the upstream processes to determine the original anomaly. For instance a biological culture in the cabin air duct may cause chemical instabilities in the potable water system, which is downstream (through the THC unit) of this duct. The only way of isolating this fault is through knowledge (a model) of the chemical and microbial behavior of each component, along with knowledge of their structural interconnections.

Advanced Intelligent Instrumentation

Long duration life support systems will be required to monitor a large range of chemical and microbial constituents in real time. This information will be used in advanced feedback control, maximizing the revitalizing effectiveness while minimizing the use of life support processing and resources.

Minimization of system resources implies that chemicals, filters, catalytic gases, and whole subsystem processes can be bypassed or reduced. Maximum revitalizing effectiveness requires adjustment of the life support processes based on the range of chemical and microbial contaminants to be stabilized. The strategic placement of intelligent instrumentation which feeds back chemical and microbial constituents will allow these operations to take place.

Airborne microbial monitoring devices may be needed. The ECLS system uses a trace contaminant monitor which produces gas species data only. Cabin air contaminants and recombination in micro-gravity may need to be known by an advanced control system which could operate to control the contaminants.

Real time, in line chemical and microbial analysis instruments must be developed. This is a tough problem that may be solved by the successful combination of medical, life sciences instrumentation with advanced software technologies. Minimal Space Station Freedom augmentation implies a device which is the same size or smaller than the Process Control Water Quality Monitor.

A modelling system would be necessary in order to optimally design and place these monitors in the subsystem interconnections. The model would allow specific constituent levels to be available to the control system, the optimal placement analysis of feedback pick off points for system microbial stability and other considerations.

Flexible Life Support Processing Systems

Maximization of the range of chemical and microbial effectiveness with minimal system hardware will require process control subsystems which would be able to adjust their processes, for instance the filter type or size, based on the constituent levels of the input and the set point requirements of the output. The filter, or other component for fluid or gas cleanup, would only be used when needed, prolonging its useable life. Multiple, exchangeable component types in the same subsystem would allow a single subsystem to clean a broader range of fluids, minimizing the necessary hardware. The potable and hygiene loops may be able to be combined in this manner.

The amounts of chemicals added, such as urine pre-treatment biocides and iodine, could be adjusted based on intelligent instrumentation feedback. Minimization of these additives would require less supply, and may limit the chemical and microbial combinatorics of the downstream subsystems.

Regenerative Life Support Modelling and Analysis Systems

A high fidelity modelling system would be necessary for optimal placement of feedback instrumentation, engineering analysis of flexible life support processing requirements, and development of biological fault isolation techniques. If the pressure, temperature, chemical, and microbial transfer equations of each regenerative agent, such as a urine processing assembly, biological culture, or crew member, could be developed, the entire system's long duration chemical and microbial stability could also be analyzed.

Verification and upgrades of the models using ground and flight ECLSS configurations would provide an advanced engineering tool for autonomous regenerative life support system engineering. Evolutionary ECLSS, Lunar Base, and Mars Excursion Vehicle life support systems developers could then perform advanced, autonomous, and optimal control system analysis as well as long duration studies using this structured knowledge. Studies could indicate the relative behavior and stability of adding a two bed molecular sieve, CELSS greenhouse module, or Lunar Oxygen mining subsystem.

Gravity constants used throughout the transfer equations would predict changes in subsystem behaviors due to changes in gravity. Biological agents which do not combine due to weight differences in ground tests could produce unexpected results in space. Updating the model transfer equations will increase the fidelity and stored knowledge of variable gravity effects on biological agents in regenerative systems.

Conclusion

The Environmental Control and Life Support System aboard Space Station Freedom will be a step ahead in the implementation of regenerative life support systems. The interactions of its subsystems with each other and the crew will serve to greatly increase our knowledge in low gravity regeneration complexities. The Space Station can be used as a test bed for verification of chemical and microbial, variable gravity transfer models which will prove essential in long duration regenerative life support system engineering and autonomy analysis.

The fully automated regenerative life support system described cannot be built today. Quite a few steps must be taken, and research performed in order to develop systems which can autonomously remain stable for long durations. A first step is to build and deploy the Freedom Station. The actual hands-on knowledge generated from ground and flight tests will allow incremental builds upon the ECLSS toward automation and long term stability. Another step is the inclusion of Life Sciences medical technology in Life Support engineering. Life support systems which use regenerative techniques to meet their supply requirements will have to actively worry about and control microbial recombination, and insure biological stability.

Long term autonomous, robust, and stable regeneration of atmospheric resources require a proportional increase in control system activity and intelligence with the decreased size of the buffer of air and water resources. Stabilizing thimble sized atmospheres for human exploration will require a deeper understanding and active participation than maintaining the integrity of the vast resources of Earth.

References

1. Architectural Control Document - Environmental Control and Life Support System, SSP 30262, February 15, 1989.
2. Dewberry, B., "The Environmental Control and Life Support System Advanced Automation Project - Phase I Application Analysis", Proceedings of the Space Operations Automation and Robotics Conference, June, 1989.
3. Dewberry, B., "Automation of the Environmental Control and Life Support System", Proceedings of the Space Station Evolution Symposium, February 6-8, 1990.
4. Lukefahr, B. D., Rochowiak, D. M., Benson, B. L., Rogers, J. S., McKee, J. W., "ECLSS Advanced Automation Preliminary Requirements - Final Report", UAH Research Report Number 823, November, 1989.
5. Prince, Norman R., et. al., "Challenges in the Development of the Orbiter Atmospheric Revitalization Subsystem", NASA Conference Publication Space Shuttle Technical Conference, Volume 2342 Part 1, June 28-30, 1983, pages 465-479.
6. Reuter, J. L., Turner, L. D., and Humphries, W. R., "Preliminary Design of the Space Station Environmental Control and Life Support System", SAE Technical Paper Series 881031, July 1988, pages 1-10.
7. Scarl, E. A., Jamieson, J. R., and New, E., "Deriving Fault Location and Control from a Functional Model", Proceedings of the 3rd IEEE Symposium on Intelligent Control, August 1988.
8. Schunk, R. G. and Humphries, W. R., "Environmental Control and Life Support Testing at the Marshall Space Flight Center", SAE Technical Paper Series 871453, July 1987.

Simulation-Based Intelligent Robotic Agent for Space Station Freedom

Csaba A. Biegl, James F. Springfield, George E. Cook, *Kenneth R. Fernandez

Vanderbilt University, Dept. of Electrical Engineering,
Box 1824, Sta. B, Nashville, TN 37235

*NASA Marshall Space Flight Center, Huntsville, AL

Abstract

This paper describes a robot control package which utilizes on-line structural simulation of robot manipulators and objects in their workspace. The model-based controller is interfaced with a high-level agent-independent planner, which is responsible for the task-level planning of the robot's actions. Commands received from the agent-independent planner are refined and executed in the simulated workspace, and upon successful completion, they are transferred to the real manipulators.

1. Introduction

It is expected that robotic systems will play a crucial rule in the operation of Space Station Freedom. Various repetitive tasks associated with the maintenance of the Station are the most likely candidates for robotics and automation applications, since such applications would free up a considerable amount of crew-time for more useful activities, like scientific experiments, etc.. Controlling these robots will be a very complex task due to unusual operating conditions, like tight space, strong interdependence of various subsystems onboard the station, and the unavoidable presence of humans near the robot's working area. These requirements will necessitate the development of highly specialized robot control systems for space applications. The goal of the research described in this paper is to design and implement an autonomous robotic agent which can serve as a component of such control systems.

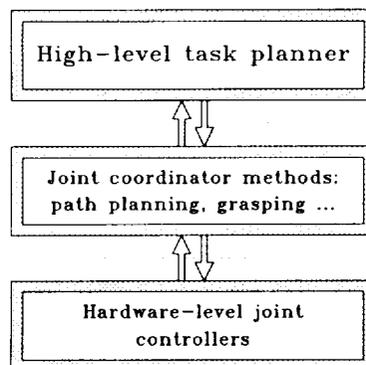


Figure 1. Multilayer Robot Control System Architecture

It is customary to distinguish between three layers in robot control systems, as seen in Figure 1. The bottom layer implements the low-level joint control systems. At this level the controllers' inputs are individual joint position and speed signals, and their output determines the voltage or hydraulic pressure applied to the joint actuators of the manipulator.

The middle layer of a hierarchical robot control system contains the modules which are necessary for the coordinated motion of the individual manipulator joints. Operations typically performed at this level include path synthesis, object grasping, and compliant motion.

The components of the highest layer implement the task planning functionalities in the system. Task planning means breaking down a complex goal into a sequence of simple actions suitable for input at the path synthesis level.

An agent-independent high-level task planner has been described in [3]. The goal of the work described in this paper was to provide a target environment for this task planner, which is capable of executing the planner's action sequences using either a graphics simulation environment or real robot hardware. The operation of this robotic agent is based on the structural, geometrical modeling of the robot manipulators and the objects in their workspace. The reason for this is that the techniques used to model solid objects provide the most natural way to describe the robots and the workspace objects. Solid modeling techniques might be used in different areas of the design and operation of robotics systems, like:

- *Design and testing of manipulators:* In such applications the purpose of the modeling is to study different approaches to fulfill the design specifications of the manipulator.
- *Robot action planning:* The modeling environment is used to build a representation of the robot and the objects in the workspace, and to create and validate action plans by simulating various actions in the model space.
- *On-line control of robot manipulators:* The modeling tools are integrated into a hierarchical robot control system, and the action plans generated and tested in the model space will be transmitted to the robot manipulators for execution.

2. Related Work -- A Robot Modeling Environment

The Intelligent Robotic Agent is based on a robot simulation and control environment described in [2]. The main requirements against a geometrical modeling toolkit used in robotics applications can be summarized as follows [1][4][7]:

- *A set of solid primitives* (like boxes, cylinders, cones, etc...) which can be used to construct the manipulator and world models.
- *A way to build structured objects* from the primitives or other previously defined structured types.
- *A set of methods to manipulate the models.* These include methods for accessing and animating objects, collision detection algorithms, and routines for forward and inverse kinematics calculations, graphics display, and possibly dynamics calculation.

These goals were satisfied by implementing a robot simulation library. The basis for this development was the ROBOSIM [5] robot simulation package. ROBOSIM in its original form is a command line oriented graphical robot modeling package. It provides commands to build the geometrical representation of robots and various objects and to perform simple operations on these.

To enhance the object manipulation capabilities of the original ROBOSIM package, a

simulation library has been built which is capable of operating on the internal geometrical model databases built up by ROBOSIM's command interpreter. This library includes the following components:

- *The animation module* provides the methods to "operate" the manipulators in the model space. Its methods include routines for driving the arms' joints, straight line motion, grasping, etc...
- *The inverse kinematics module* aids the animation by providing the inverse kinematics solutions for the robot arm(s) in the model space. By default it uses a numerical method, but if the analytical inverse equations are available, the default method can be replaced by these.
- *The collision detection module* is used by the animation module to check the validity of the steps during a movement. If the step would result in a collision, a report is sent to the animation module, describing the objects involved in the collision.
- *The robot interface driver module* is used to duplicate the actions taken in the simulation environment in the real robot's workspace. It generates calls to a low-level interface library based on the simulated joint movements. Error reporting is also possible in case the command fails in the real workspace.
- *The graphics display module* is used to display the workspace configuration and to follow the actions in the simulated environment.

The structure of the modeling environment composed of the above modules and the original ROBOSIM interpreter can be seen in Figure 2.

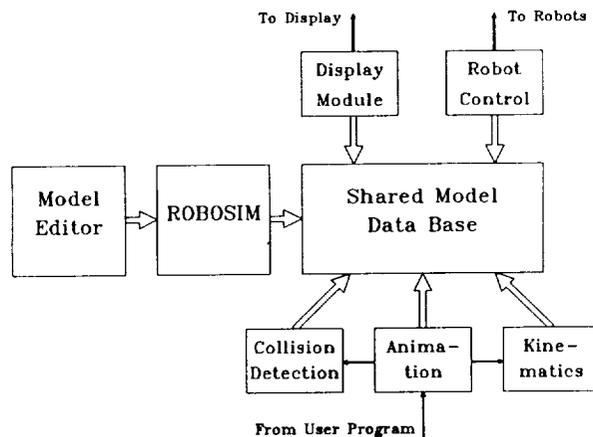


Figure 2. Components of the Robot Modeling Environment

3. The Intelligent Robotic Agent

The purpose of the Intelligent Robotic Agent is to provide a user-friendly way to use the tools of the Robot Modeling Environment described above. The agent was designed to be able to receive robot action sequences either from a high-level symbolic task planner or

directly from the user during an interactive session. The main requirements against the agent are summarized below:

- *Symbolic Planner/User Interface Protocol:* Regardless of the operating mode of the agent (used by a task planner or by an interactive user), the same command interface is being used. This command language provides symbolic manipulator and object identifiers for a more user-friendly programming environment. The interface protocol also includes extensive error reporting mechanisms.

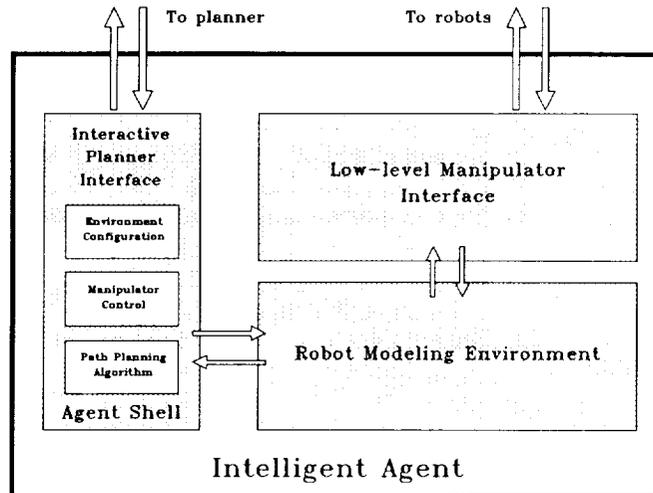


Figure 3. Intelligent Agent System Architecture

- *Environment Configuration Interface:* In order to ensure the consistency of the world models with the task planner, the agent includes a set of commands forming a configuration interface which is used by the planner (or by the user in an interactive session) to build the geometrical representation of the workspace. The agent features its own modeling language for creating elementary objects, building complex objects from these, and to perform various transformations. In view of the considerable effort already invested in creating models of various objects and robots in the ROBOSIM language, the possibility of using existing ROBOSIM models is also provided.
- *Agent Operation Interface:* A set of commands is provided to operate the robot manipulator models in the agent's data base. These include commands for various types of motion (joint interpolated, straight line, etc.), path planning with collision avoidance, and for grasping and releasing objects.
- *Graphics Display:* Probably one of the most useful features of any robot simulation package is the possibility to view the manipulators in their workspace as they perform various actions, in a safe, simulated environment. The agent includes a graphics display feature which is usable during both the environment configuration and the agent operation phases.
- *Interface to Robot Controllers:* The agent can operate in either of two modes. In the first mode the actions received from the command interface are performed only in the geometrical modeling environment. In this case the execution can be viewed on a graphics display, and the agent will report any error conditions encountered during the execution (collisions, joint violations, etc.). In the second mode the successful execu-

tion of the above steps is followed by sending commands to a real robot hardware interfaced to the agent to duplicate the actions taken in the simulated environment in the real workspace too.

The above goals were satisfied by incorporating the services of the modeling environment into an interactive shell, and by creating a low-level interface package between the simulation library's robot interface driver module and the controller of the manipulator used in the system. The architecture of the Intelligent Agent can be seen in Figure 3.

The agent shell's task is to interpret the commands sent by the planner. It has a built-in symbol table for storing the various objects in the workspace and the attributes associated with these. After parsing the commands, the shell will invoke the appropriate functions of the robot simulation library with an argument structure built using the information stored in the symbol table. Any results or error codes returned by the library call are translated into the corresponding message format and sent back to the planner, or printed to the user during an interactive session.

One important design consideration associated with the agent was the minimization of the numerical calculations required from the task planner. To accomplish this goal various attributes are associated with each object. These attributes include the object's current position, predefined grasping points, and predefined joint parameter vectors for the robot manipulator objects. These attributes are linked to the object, i.e. any transformation will update their values. This means, that for example the task planner does not have to compute numerical coordinates to move a robot manipulator to an object's grasping point, but it can use the symbolic grasping point attribute whose value is always linked to the object's current position.

The agent shell also includes a path planning algorithm, which is used to synthesize a collision-free path between two locations in the workspace. This algorithm relies on the collision detection feature of the simulation library and utilizes a simple search strategy to find an alternate route in case a collision is detected during a motion segment. First the endpoint of the failed motion segment is checked to see if there is any collision when the manipulator reaches that point. If this is the case then the search fails, since the motion segment can not be completed regardless of the route taken. Otherwise the search algorithm will subdivide the failed motion segment by inserting an intermediate point selected using a set of heuristic rules, and recursively repeat the above process. There may be situations when this method will fail to find a path even if such a path exists, but the algorithm described above has been observed to find a solution in the majority of the situations examined so far.

The agent shell also supports the parallel operation of robot manipulators. An extended command line format makes it possible to enter commands for every manipulator in the system at the same time, and these commands will execute parallelly.

4. An Application Example

The Intelligent Agent has been used as an interface between a robotic task planner and a PUMA 560 manipulator. The architecture of the whole system can be seen in Figure 4. The low-level robot interface between the agent shell and the PUMA manipulator's controller has been implemented in a distributed fashion, partly on the workstation running the agent shell, and partly on an IBM PC AT. The reason for this solution was that parts of this interface package have to meet certain real-time requirements.

The configuration described above has been used to execute task plans for repairing electronic equipment. The operations included setting various switches and dials on the front panels of the racks housing the equipment, and replacing faulty circuit boards. A graphics display of the simulated workspace of this application can be seen in Figure 5.

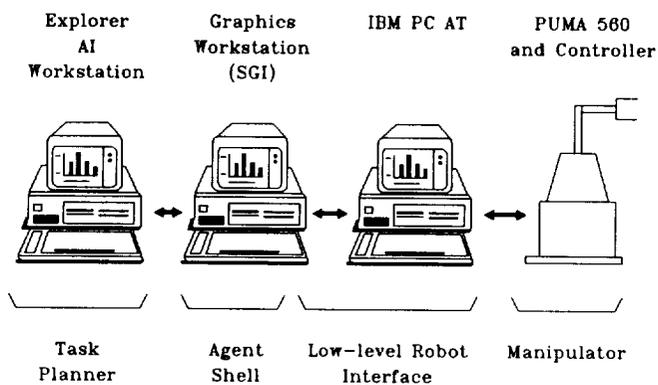


Figure 4. System Configuration for Intelligent Agent Application Example

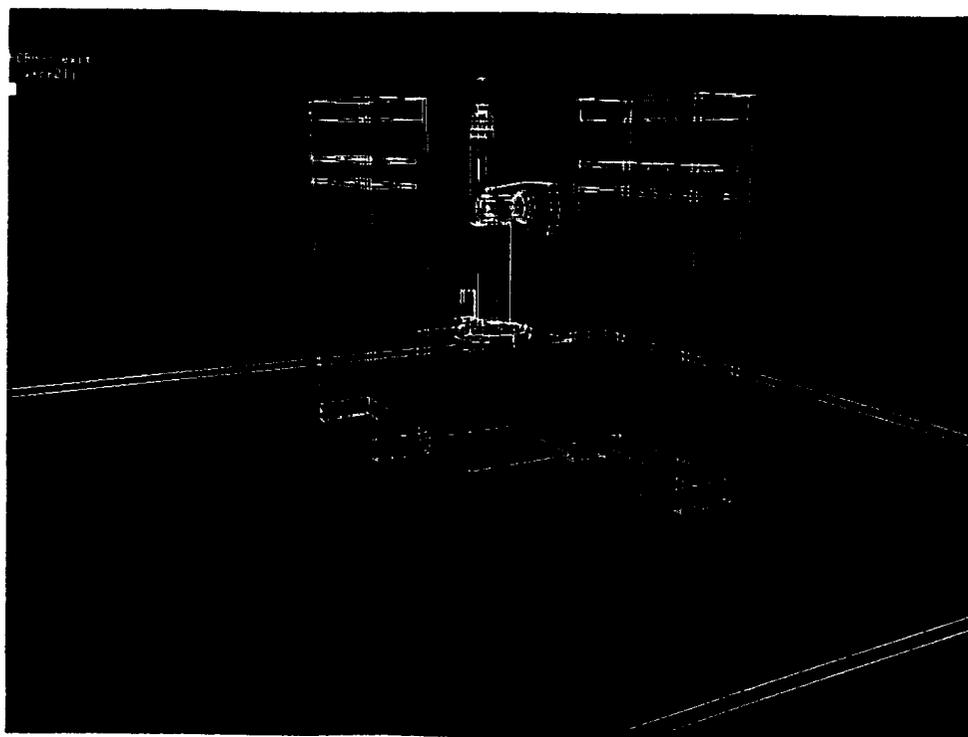


Figure 5. Workspace Configuration for Intelligent Agent Application

5. Summary and Future Research

An intelligent agent has been developed for the execution of robotic task plans generated by a high-level planner. The agent uses a well defined protocol for receiving environment configuration and manipulator commands from the planner and to send back status reports. After translation, the agent uses the services of a robot simulation and control environment to carry out the high-level commands of the planner. This configuration has been used to examine the possibilities of using robots for repairing electronic equipment.

The advantage of this approach is its flexibility. It is easy to adapt the system to different workspace configurations or to different robot manipulators by changing the environment configuration commands sent to the agent. The well-defined planner-agent interface simplifies the development and verification of new task planners, because the task plans can be tested for validity by the agent, and the user can also see the effects of the planner's output on the graphics display.

The most important area of planned future research is the upgrading of the low-level robot manipulator interface. The current interface uses the PUMA manipulator's command language (VAL II [8]) to execute the robot movement commands, but a new version is currently being developed which will bypass this command language and use the PUMA controller's "real-time path control" feature to achieve an even better quality of joint control. This new version will be built using the so called Multigraph Architecture [6], which provides a dynamically configurable distributed macro-dataflow computational environment. This will allow the selection of the joint control scheme and its parameters from the agent shell, resulting in a highly optimized controller performance for every application.

6. Acknowledgements

William S. Davis and Ray Carnes (Boeing Aerospace Company, Huntsville, AL), and Professor Janos Sztipanovits (Vanderbilt University, Nashville, TN) provided valuable technical information and criticism during the development of this work. Some of the equipment necessary for the work was provided by the Hewlett Packard Company and the Boeing Aerospace Company.

7. REFERENCES

1. B. Bhanu, C. C. Ho: "CAD-Based 3D Object Representation for Robot Vision", *Computer*, Vol. 20, No. 8, Aug. 1987, pp. 19-35
2. C. A. Biegl, J. Springfield, et. al.: "Adaptive Control of a Dual-Arm Robot Manipulator Using On-Line Graphical Simulation", in *Proceedings of the ISA Forth Annual Workshop on Robotics and Expert Systems*, (August 1989, Palo Alto, CA) vol. 4, pp. 253-259.
3. W. S. Davis: "Robotic Task Planning: Independent of Agents but Dependent on Time", in *Proceedings of the 1989 IEEE Conference on Robotics and Automation*, (May 1989, Scottsdale, AZ) vol. 2, pp. 690-695.
4. E. Dombre, P. Borrel, A. Liegeois: "A CAD System for Programming and Simulating Robots' Actions", in *Computing Techniques for Robots*, ed I. Aleksander, Chapman and Hall, New York, 1985, pp. 222-247.

5. K. R. Fernandez: "Robotic Simulation and a Method for Jacobian Control of a Redundant Mechanism with Imbedded Constraints", Ph.D. thesis, Vanderbilt University, Spring 1988
6. G. Karsai, et. al.: "Knowledge-Based Approach to Real-Time Supervisory Control", *Proc. of the 1988 American Control Conference*, Atlanta, GA, pp. 620-626, 1988.
7. C. Mirolo, E. Pagello: "A Solid Modeling System for Robot Action Planning", *IEEE Computer Graphics & Applications*, January 1989, pp. 55-69
8. Unimation Inc.: "User's Guide to VAL II", Danbury, CT 1986

Graphical Explanation in an Expert System for Space Station Freedom Rack Integration

F.G. Craig, D.E. Cutts, & T.R. Fennel
Boeing Computer Services M/S JA-74
Huntsville Artificial Intelligence Center

B. Purves
Boeing Aerospace M/S JA-62
499 Boeing Blvd.
Huntsville, AL 35824-6402

COPYRIGHT 1990 THE BOEING COMPANY
PUBLISHED WITH PERMISSION

Abstract

This paper focuses on the rationale and methodology used to incorporate graphics into explanations provided by an expert system for Space Station Freedom rack integration. The rack integration task is typical of a class of constraint satisfaction problems for large programs where expertise from several areas is required. Graphically oriented approaches are used to explain the conclusions made by the system, the knowledge base content, and even at more abstract levels the control strategies employed by the system. The implemented architecture combines hypermedia and inference engine capabilities. The advantages of this architecture include: closer integration of user interface, explanation system, and knowledge base; the ability to embed links to deeper knowledge underlying the compiled knowledge used in the knowledge base; and allowing for more direct control of explanation depth and duration by the user. The graphical techniques employed range from simple static presentation of schematics to dynamic creation of a series of pictures presented "motion picture" style. User models control the type, amount, and order of information presented.

Introduction

The Space Station Freedom (SSF) Program is a complex task requiring the integrated skills of thousands of people. There are many examples within the program of tasks which require the cooperation and participation of several organizations to make critical decisions. As automated expert systems are developed to aid in these decisions and to capture the knowledge from several areas, we should be able to ask them for explanation/justification of their results as we would human experts. The task of rack integration is exemplary of tasks for which justification is required. The racks aboard SSF provide a common element around which design, operational, manufacturing, and logistics decisions are made. The basic task is to decide where racks of a given type should be located aboard SSF. There are several types of constraints which influence the final decision, ranging from operational (such as noisy racks should not be located near crew sleeping quarters) to physical constraints dependent upon other design decisions (such as the general rule that data management system racks, although shielded, should not be unnecessarily located next to potential sources of electromagnetic interference).

The expert system to aid in the integration of this task is documented in detail in an accompanying paper at this conference [1]. This paper will focus on the benefits, methodology, and some of the issues researched for making such systems more usable and complete in the area of explanation.

Explanation

One of the earliest claims of expert system developers was that the resulting systems could "explain" their actions. These claims were often effectively backed up by the textual presentation of traces of rule firings which could explain "how" the system had made a decision.[2] Additionally, systems could answer "why" the system was asking for information by presenting as explanation an English text description of the rule which required the information.[3] However, complete explanation requires addressing the problems of what, how, when and to whom knowledge is to be communicated. In the past, most expert systems have typically relied on textual presentation. Notable exceptions to this include the STEAMER [4] system which used an underlying simulation model with incorporated graphics and the General Electric DELTA expert system for diagnosing diesel electric locomotive failures which incorporated video storage as part of the system[5].

Wick and Slagle [6] suggest that explanation capabilities could be greatly enhanced by the introduction of supplementary knowledge and by allowing variations of queries over time. For example, the user could ask not only "Why do you want to know this now?", but could also ask "Why would you ever ask me for this information?". Similarly the user could ask not only "How did you know?" , but also "How could you find out?". To answer these questions the system must keep extended histories, or traces, of actions taken by the expert system and based on dependencies be able to generate responses of a forward looking nature.

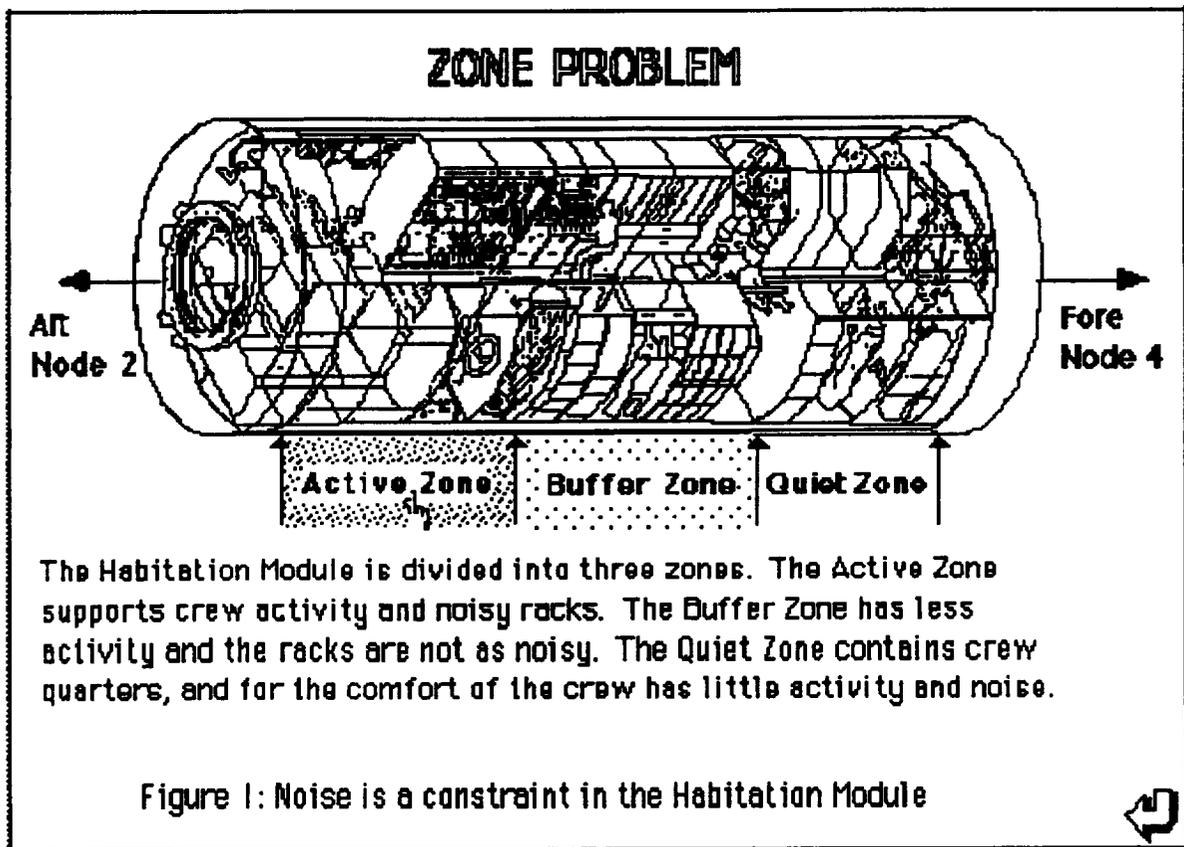
Chandrasekaran, Tanner, and Josephson [7] emphasize that explanation should be provided not only at the low levels (exemplified by presenting the conditions associated with a single specific rule) but that high-level explanation of overall system goals should also be available. Their suggestions are supported by work on automatic generation of textual explanations through specialized grammars [8] . An underlying truth here is that humans tend to be much better at explaining their actions because they are able to convey both their abstract goals and detailed information -- but with the significance of the details "slanted" towards satisfying the stated goals. Therefore, the grammar used by humans during explanation goes beyond that used for simply explaining system details.

Most explanations are presented to a single individual, or at least to a group with focused attention in a common setting. An additional level of complexity is added to the problem of explanation when we introduce the need for models of the user so that the information presented will be both understandable and timely. Related work [9] in the rapidly expanding field of intelligent tutoring systems demonstrates repeatedly that it is the **communication of knowledge** (not just data) that is important and that the presenter of knowledge must make allowances for student abilities. For example an expert system developed as an engineering aid may be used repeatedly by individual engineers who are experts in the domain. However; when explaining the actions of the system (which have led to specific decisions) during a formal review, the experts must be able to integrate background information, current focused information, and their overall goals into explanations at a level their audience will understand. (And insistence on understanding is something formal review boards are well known for!) The point is that the same explanations given by the system to the expert during its normal use will not suffice as explanations given to a broader audience. The task of trying to model even the typical user (in an effort to know what to present and how to present it) is often not straightforward.

Rationale for Incorporating Graphics

An ancient Chinese proverb states "It is better to see a thing once than to read about it one hundred times." The wisdom of this statement has been proven repeatedly by people who while trying to explain their actions to others resort to the use of a graphic for clarification. For the rack integration task we developed guidelines which dictated that even the quick sketches of an expert should be included as part of the documentation for any rules developed as a result of a knowledge engineering session. Therefore, perhaps the best rationale for incorporating graphics is simply to mimic reliance upon them as humans do.

A sequence of pictures is often very effective at presenting information as it changes over time, and in many situations an appreciable amount of information can be conveyed by a single picture. For example, figure 1 graphically illustrates the noise constraint mentioned previously for the Habitation module of the Space Station. From the figure it becomes obvious that "noisy" racks should be located at the aft end of the module, with a buffer zone located between them and those at the opposite quiet end. Closer scrutiny of more detailed drawings reveals that most of the subsystem related racks such as the Air Revitalization System (ARS) and Thermal Control System (TCS) are located at the noisy end of the module because of the mechanically oriented nature of those racks. However, the galley/wardroom racks are also at the noisy end, indicating that noise associated with the use of a rack is also enough reason to isolate it from the crew sleeping quarters.



Modern portable computers, optical discs, and graphics software make it possible to quickly and easily capture and integrate graphic material. The architecture chosen for our research combines database, hypermedia and inference engine capabilities. The

specific software packages used in the initial effort are Microsoft Excel, Neuron Data's Nexpert Object, and Apple Computer's Hypercard. The hardware platform chosen was a Macintosh II with 8 MB of memory.

Additionally, the recent emergence of very affordable hypermedia systems was also a major contributor in our decision to incorporate graphics. By using figures which have been scanned in, and then adding "buttons" or links to additional information we can allow for perusal of a tremendous amount of information at a level dynamically controlled by the user. It is important to realize that the links created for explanations tend to be more specific than those created simply for an informational stack -- at least at the beginning of the explanation. However, as the user traverses links away from the starting point the bounds on what type of information is presented is left up to the system developers. For the rack integration expert system we "flavored" the entire network of information by relating to a hierarchy of interface, control, constraints, and state information (the layered approach used here is typical of constraint satisfaction problems we have dealt with in the past and is documented in detail in [1]).

In the following three sections we present our research applied to the three areas of explaining knowledge base content, strategies, and decisions. Chandrasekarn, et al, [7] provide details regarding this three pronged approach for explanation from introspection of knowledge and inference.

Explaining Knowledge Base Content

For our research purposes we have pursued providing explanation of knowledge base content at all levels. Starting with the lowest level, the underlying database represents basic facts about the problem (such as the number of possible locations for racks in each module) or about the current state of the world as the knowledge base knows it (engineers often start their analysis from a baseline configuration of rack assignments and attempt perturbations). For the database we provide information on the data sources, last update, units of measure, and validity intervals.

At the next highest level, an object hierarchy is provided and the object definitions are all linked to conceptual definitions. Graphics depicting component and subcomponent details are used where appropriate. Information provided about each object class include its importance in the rack integration task and how it is used in the problem solving process. Each object attribute is similarly treated with the addition that each object attribute is also flagged to indicate whether its value is simply read in from the database or can be changed by the problem dynamics. The idea of assigning values of LABDATA to data that typically requires no explanation other than source was suggested by Davis, et al in [10]. Where attributes can have multiple values, the meaning of the multiple values is explained, along with expected consequences on the problem solving process. For example, the "RACK" class which represents the rack objects has an attribute "noise_level_environment_required". The values for this attribute are "sensitive", "not very sensitive", or "not sensitive at all". The effect is that racks which are "sensitive" to noise can only be located in the quiet zone of the Habitation module (noise is not a concern in the Laboratory or Logistics modules).

The constraint rules form the third level of the knowledge base and serve to emphasize that in a rule based system oriented towards explanation the rules themselves should be thought of as objects. A graphical depiction of the constraint hierarchy is presented using only keyword phrases. Additionally each rule is captured in hypertext form, so that the user can select any rule from the keyword hierarchy, then any part of the rule can be selected to explain the contents in more detail. Rule attributes include static English text which restates the rule, the rule originator, last update, a list of pointers to any related "cases" or "tests" from which the rule was derived, the relation to other rules, an understandable English text prompt used in conjunction with the rule when requesting information, and a graphical representation of the rule where possible.

Although our current system does not use confidence factors, it is interesting to note that the confidence factors themselves convey knowledge that should be explained.[10] A confidence factor of unity indicates that a "shallow" explanation may suffice since the rule is most likely definitional in nature, while confidence factors not equal to unity represent the application of judgement and the relevant ranking of its importance and therefore requires more explanation.

Explaining the Knowledge Based System Strategy

The control rules at the fourth level of the knowledge base are also represented in a graphic hierarchy. At this level the source for the rules becomes critical as these are the rules which control the order for checking the constraints at the next lower level. These rules explicitly determine which constraints are checked under varying circumstances. Not all constraints are checked for the varying types of racks. For example constraints associated with zoning restrictions based on the type of science are only checked for racks in the Laboratory module. The strategies implemented intentionally mimic those used by experts from various areas within the rack integration domain. For example, the strategy for checking the constraints associated with moving a Laboratory payload rack were derived from knowledge engineering sessions with a payload integration specialist. Because payloads are typically unique, they have widely varying utility requirements. This is exactly one of the areas checked first and is responsible for most problems with integrating payload racks. Justification of this strategy is supported with a graphic depicting the low percentage of common interface plates in the Laboratory module due to payload unique requirements.

Graphically representing generic tasks such as "hierarchical classification" or "plan selection and refinement" has proven to be a very difficult task. Current efforts are focusing on the use of simple conceptual sketches or icons presented in a cyclic manner to emphasize the ongoing and dynamic nature of such tasks.

Explaining Knowledge Based System Decisions

The ideal situation here is to employ any material a person may use, the point being to represent the "bottom line" as clearly as possible. For example, the rule hierarchies presented to explain the knowledge base content can be enhanced by highlighting information (the computers equivalent of pointing) used in the decision process. For the rack placement expert system we incorporated the ability to highlight a single keyword representing a rule or group of rules while presenting results from an analysis (see figure 2). This often served as sufficient explanation for the domain experts, while links from the keyword hierarchy provided the "back pocket" type of information (previously shown in figure 1) needed for justification to other audiences.

Following the example set by [11], we have attempted to anticipate what are most likely to be the more difficult areas involved in making the decisions and have provided even more depth and tutorial type of information for explanation of decisions in some areas. For example, the routing of utilities required by a rack is an area where many of the verification test cases showed that the human experts had the hardest time explaining their actions. For this reason, the assumptions and formula used for calculating weight, volume, and length information for utilities are all well documented and incorporated in explaining decisions affected by routing criteria.

SOFT CONSTRAINTS

Moving rack HMF EXERCISE WITH WINDOW from HS17 to location HP4 may violate the following soft constraints.

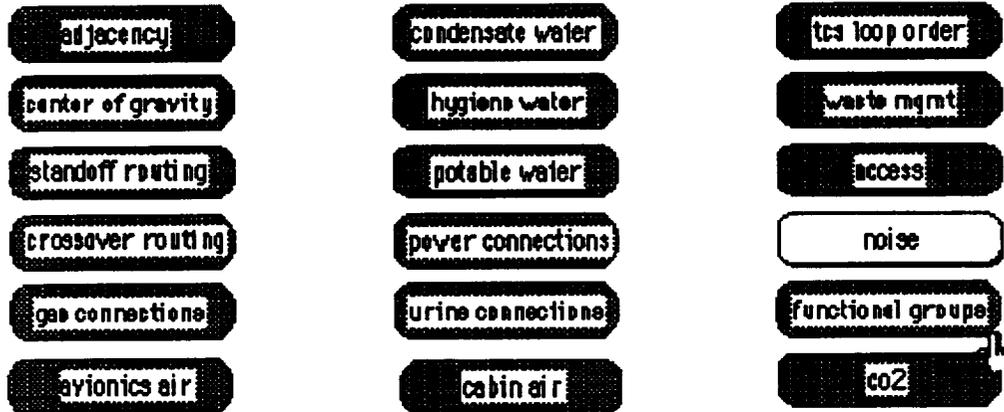


Figure 2: The noise constraint is readily identified

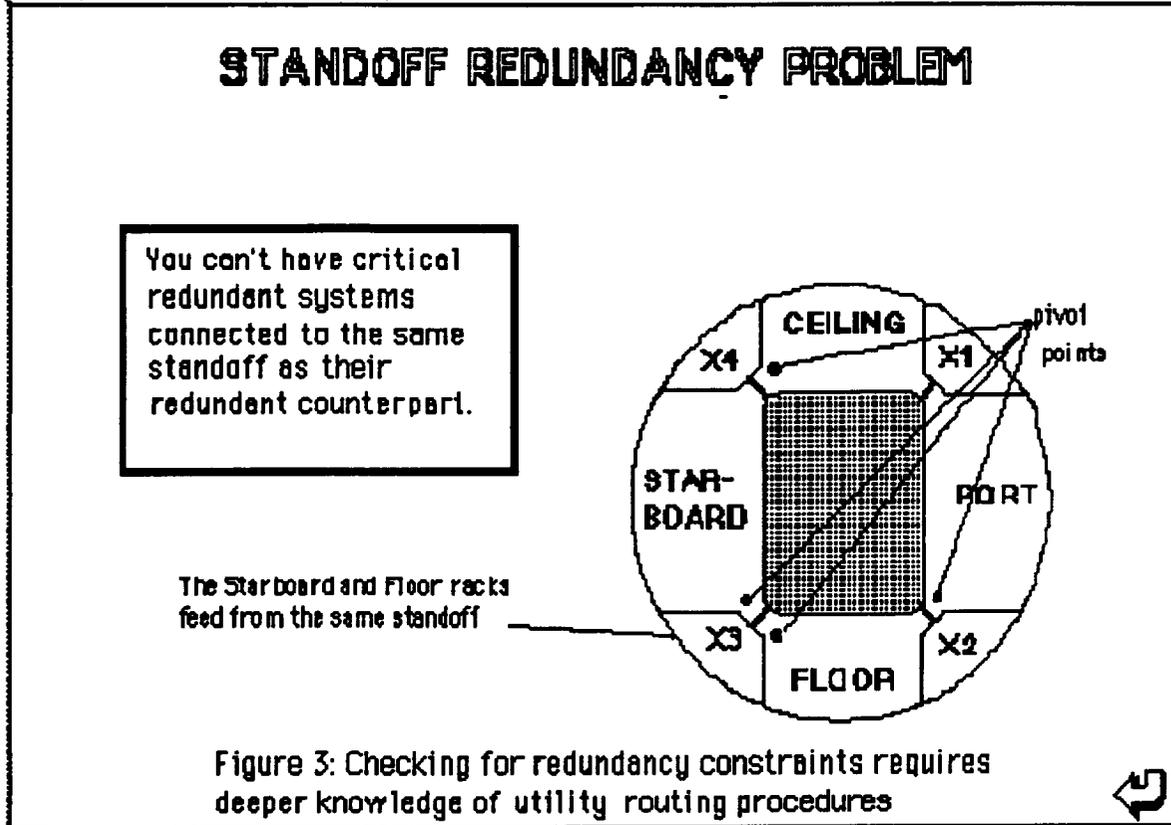


It is for use in explaining decisions that we are developing user models to control the first level of explanation presented to the user. The interface presents only a CAN or CAN NOT decision regarding placement of a rack in a given area and a brief explanation of WHY NOT if the placement was disallowed. Whenever the user asks for further explanation, a "novice" user is presented with a more detailed explanation of the type of problems encountered. An "experienced" user is linked directly to the constraint keyword hierarchy. At the present time, the explanation information presented is mostly static -- prepared beforehand. One of our areas of interest in extending the system is in dynamic creation of explanation objects which would change with the circumstances associated with the knowledge base and with the user. We have made a first step in this direction with the constraint keyword highlighting mechanism mentioned above.

Capturing the "Link" between Compiled and Deep Knowledge

An admission on our part and hopefully a lesson for others is that our first pass at using graphics to explain the knowledge base content was woefully inadequate. It was only when a new member joined our team who was totally unfamiliar with the SSF program that we came to realize this fact. Without knowing it we had been unintentionally "compiling out" knowledge by not representing what we had come to believe was "common sense". For example, we had neglected to document the reasoning behind not allowing racks requiring windows to be placed on the wall facing forward in the SSF orbit. These walls are more subject to meteor hits than the other walls and since windows are regarded as built in safety hazards anyway, they should be located where they are not likely to get hit. Obvious. Right. Another example is where different

walls (the Starboard and Floor walls) use the same physical area for routing of utilities. This imposes an additional level of constraints to be checked to satisfy the requirement for separation of redundant systems as illustrated in figure 3.



It is the high level and abstract knowledge (such as originally intended use, goals, or even current events such as budgetary constraints) that is often compiled out of the final version of a knowledge base. As a result, explanations associated with expert system will most likely be later questioned regarding completeness, accuracy, or accountability -- and the true explanations may not be available. For the rack integration expert system we have used graphically oriented techniques to document the source, intent, and actual meaning of the knowledge in the knowledge base. We've found that the most difficult part of this is indeed deciding how to graphically represent the higher level goals and in many cases we use simple English text statements as they seem most appropriate. The more abstract problem solving goals (such as the control rules) are depicted using process flow diagrams. A fairly simple mapping allows for capturing the link between the control rules and the constraint rules.

Future Directions

The Apollo program provides proof that much of the data, information, and knowledge associated with large aerospace programs can be lost to later generations. One of the goals of the Space Station Freedom (SSF) program is to ensure that not only is basic data and information available for future access, but also that knowledge available now is also captured for later use by the program. However, while documentation for data or computer programs often have very specific standards imposed upon them, the standards for documentation associated with captured knowledge is still in the formative stages [12]. One of our research goals is to investigate ways of testing how to document

captured knowledge. It is fairly easy to understand that just as comment statements form an important part of computer program documentation, explanation capabilities can be used to determine how well a knowledge based system is documented.

We also recognize a need to expand the explanations of why a rack WAS allowed in a given location, not just WHY NOT. The current approach uses the how capabilities of the expert system shell to graphically demonstrate that the control rules were invoked and which constraints were checked.

It has been suggested [13] that links to conceptually faithful simulations can provide for a form of continuous explanations and could thereby represent a deeper knowledge of the domain. We would like to pursue this area by providing links to an application written for simulating the effects of different routing strategies.

Construction of an appropriate grammar for describing the relationships among objects and rules within the domain and specialized for use in explanations is being considered for future research. The grammar definition would help ensure future applications would find the embodied knowledge was machine intelligible and could be used to limit the scope of explanations which must be generated.

We would like to continue to investigate the use of expert systems as intelligent tutors. Conceptual definitions of objects and rule hierarchies are used extensively in explanations, and serve as excellent starting places for those using the system as a tutor. These hierarchies can be used for quickly identifying areas of interest to different users.

Summary

This research has focused on incorporation of graphics into explanations for a knowledge based system. The test domain chosen was that of rack integration for the Space Station Freedom. This test domain is typical of a class of constraint satisfaction problems and demonstrates that configuration tasks are particularly amenable to effective use of graphics in explanations. Components of explanation include explaining knowledge base content, strategy, and decisions.

By emphasizing explanation as a major system goal the systems can benefit: by being more readily received in the end user environment; by also serving as a beginning platform for instruction; by providing links to the deeper knowledge underlying that which would normally be compiled out of the knowledge base; and by providing for smoother integration of interface, knowledge base, and data which helps ensure they will continue to be used.

References

- 1) Craig, F., Cutts, D, and Fennel, T. , A Knowledge-Based approach to Configuration Layout, Justification and Documentation, 5th Conference on Artificial Intelligence for Space Applications, Huntsville, AL, May 1990.
- 2) Pople, H.E., The Formation of Composite Hypotheses in Diagnostic Problem Solving, in *Proc. Fifth IJCAI*, Morgan Kaufmann, Los Altos, Calif., 1977, pp. 1030-1037.
- 3) Clancey, W.J., The Epistemology of a Rule-Based Expert system -- A Framework for Explanation, *Artificial Intelligence*, May 1983, pp. 215-251.
- 4) Hollan, J.D.; Hutchins, E.L.; and Weitzman, L. (1984) STEAMER: an interactive inspectable simulation-based training system, *AI Magazine*, vol. 5, no. 2, pp. 15-27.
- 5) Bonissone, P.P. and Johnson, H.E., Expert system for diesel electric locomotive repair, *Knowledge-based Systems Report*, General Electric Co., Schenectady, N.Y., 1983.
- 6) Wick, M.R. and Slagle, J.R., An Explanation Facility for Today's Expert Systems, *IEEE Expert*, Spring 1989, pp. 26-36.
- 7) Chandrasekaran, B.; Tanner, M.C.; and Josephson, J.R., Explaining Control Strategies in Problem Solving, *IEEE Expert*, Spring 1989, pp. 9-24.
- 8) Bridges, S., and Johannes, J.D., Integration of Knowledge Sources for Explanation Production, *1989 ACM*
- 9) Wenger, E., *Artificial Intelligence and Tutoring Systems* , Morgan Kaufmann Publishers, Inc., 1987.
- 10) Davis, R. and Buchanan, B., Production Rules as a Representation for a Knowledge-based Consultation Program, *Readings in Knowledge Representation*, Brachman and Levesque eds, where, when, pages?.
- 11) Brown, J.S.; Burton, R.R.; and de Kleer, J., Peagogical, Natural Language and Knowledge Engineering Techniques in Sophie I, II, and III, in *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, eds., Academic Press, London, UK, 1982, pp 227-282.
- 12) Carnes, J.R.; Fennel, T.R.; Rice, R.; Trammel, M., Automation and Robotics Plan, *Space Station Freedom Data Requirement SE08*, draft 1988
- 13) Weld, D.S. (1983) Explaining complex engineered devices. *BBN Report 5489*. Bolt Beranek and Newman Inc., Cambridge, Massachusetts.

Space Station Advanced Automation*

Donald Woods
MDSSC-SSD
16055 Space Center
Houston, TX 77062
(713)280-1500
DWoods@NASAMAIL.NASA.GOV

In the development of a safe, productive and maintainable space station, Automation and Robotics (A&R) has been identified as an enabling technology which will allow efficient operation at a reasonable cost. The Space Station Freedom's (SSF) systems are very complex, and interdependent. The usage of Advanced Automation (AA) will help restructure, and integrate system status so that station and ground personnel can operate more efficiently. To use AA technology for the augmentation of system management functions requires a development model which consists of well defined phases: evaluation, development, integration, and maintenance. The evaluation phase will consider system management functions against traditional solutions, implementation techniques and requirements; the end result of this phase should be a well developed concept along with a feasibility analysis. In the development phase the AA system will be developed in accordance with a traditional Life Cycle Model (LCM) modified for Knowledge Based Systems (KBS) applications. A way by which both knowledge bases and reasoning techniques can be reused to control costs is explained. During the integration phase the KBS software must be integrated with conventional software, and verified and validated. The Verification and Validation (V&V) techniques applicable to these KBS are based on the ideas of consistency, minimal competency, and graph theory. The maintenance phase will be aided by having well designed and documented KBS software.

Introduction

The development of complex space systems is a costly endeavor; however, the operation of these systems is where the majority of the cost will occur. For example in the National Space Transportation System (NSTS), approximately 50% of the current program cost went into the design; however, by the end of the program lifecycle operations will have accounted for 95% of the total program cost. The important thing to remember while designing the system is that the ease of operations and maintenance will be the long term life cycle cost drivers.

Building the space station will be more difficult than the shuttle. For example, it will exist in at least 20 different configurations during the assembly phase, and at each of these minor milestones, it must meet different requirements with different resources while still maintaining safety. The shuttle was built on the ground by one major contractor while the station will be built in space by 4 NASA work packages, and 3 international partners.

All of these factors taken together point out that the space station will be one of the most complex engineered systems ever taken into space. The Systems Engineering and Integration (SE&I) in the program are of utmost importance. A&R has been recognized by congress, NASA, and the contractors as being an enabling technology for designing a safe, efficient, reliable, and maintainable station. However, KBS have no well defined standards for evaluation, development, and integration. Possible technical approaches to be employed in each of these tasks will be explained. Without well defined engineering approaches few engineers will be willing to give this technology a chance in a budget, and safety conscious environment.

* Work funded by NASA contract NAS 9-18200, the technical monitor was Jon D. Erickson.

Motivation

The operation of the shuttle requires approximately 5,000 people to support a 10 day mission every 90 days; the space station will be onorbit for 30 years, hopefully this won't require 45,000 people to support. The operation of complex systems requires a lot of manpower. The traditional approach to dealing with faults has been to develop malfunction procedures based on Failure Modes and Effects Analysis Critical Item List (FMEA/CIL). This approach has been fairly successful in dealing with common failures, with a few exceptions. For example out of 9 inflight Shuttle Remote Manipulator System (SRMS) failures none were covered in approximately 200 FMEA/CIL sheets. We are able to deal with the failures we anticipate; however, the unexpected failures resulting from faulty instruments, and unexpected causality are very hard to isolate. Currently in the NSTS program these faults are isolated by ground controllers who examine telemetry data, check system schematics, execute simulations in the mission evaluation room, and basically do very difficult and thorough analysis in real time.

These demands upon the controllers have caused them to investigate, develop and use a KBS. The first KBS in the Mission Control Center (MCC) is the Integrated Communications Officer (INCO). It has been so well accepted by the controllers that at least four more KBS consoles are being planned: Guidance, Navigation, and Control (GN&C), On-orbit propulsion, Electrical Power, and Life Support [1]. INCO has produced major cost savings (\$480K/Year) [2]. The personnel at Kennedy Space Center (KSC) have also identified A&R as being one of the most important new technologies which should be developed for SSF [3]. It is clear that the personnel involved in the operation of space systems consider automation very important, and even necessary.

Development Model

In the automation development model (Figure 1) several distinct phases are identified which will insure that safe, productive, and reliable automation applications are put into the SSFP [4]. Steps 1 to 3 help define what is clearly needed in terms of requirements, and the various changes that constantly occur in major programs. The goals and objectives (step 4) are used to develop criteria for candidate selection. The approaches of the subcontractors in meeting these requirements is feed into the A&R management function where a decision is made along with the customer on whether or not a function is worth evaluating for eventual inclusion into the baselined system. A detailed assessment of the support capabilities available in the program will consider the maturity and availability of KBS tools in the SSE, analysis of the Master Measurement and Command List (MMCL) for sufficient sensors and actuators to allow automated operation, inter- and intra system dependencies, and what KBS technology is applicable to the problem.

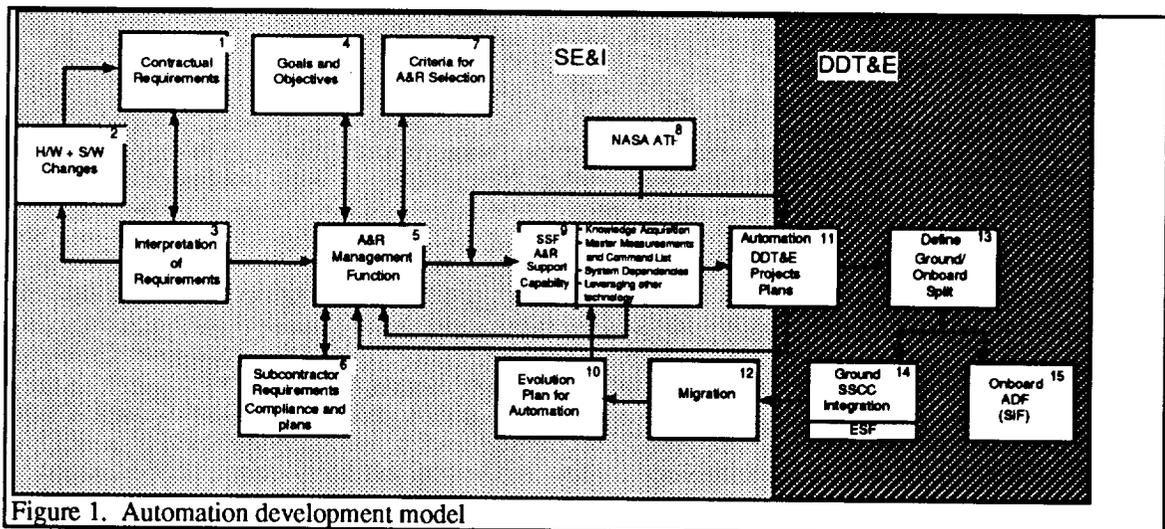


Figure 1. Automation development model

During Phase B a thorough evaluation of the Space Station's (SS) functions identified 22 candidates which could be automated. Four different approaches were considered for doing these candidate evaluations: linear weighting, fuzzy set method, Weight Outranking Method (WOM), and Multi-Attribute Utility Theory (MAUT) [5-6]. The last two were used. MAUT requires a greater degree of definition of the

system, and each candidate evaluation is independent of other candidates; however, it is generally more accurate if criteria and relative importance are well defined. WOM causes each candidate to be relative to each other candidate, and with a large number of candidates ranking can be difficult. If any further evaluations must be performed MAUT would probably be the more useful measure considering the level of design detail at present. This sort of analysis can also be used to determine the most applicable KBS or traditional technique for implementing a given function. In many cases it will probably be a combination of both. Based on this work there are two broad categories for which automation is being considered for performing SS functions: Planning and Operations. Planning entails resource, logistics, and maintenance management and scheduling. Operations involves: monitoring and analysis, Caution and Warning (C&W) filtering, Fault Detection Isolation and Recovery (FDIR), Testing, Fault Tolerance and Redundancy Management (FT/RM), and Command and Control (C&C). This analysis is also justified as part of the required logistics activities; for example in MIL-STD-1388-1A, the following tasks are identified: 301, 302, 303. Task 301 defines the requirements; task 302 identifies possible technologies for implementing these operational requirements; and task 303 performs the trade-study to identify the most promising implementation technique.

The next step is the development of a Design, Develop, Test and Evaluation (DDT&E) plan. This is where the focus of activities moves from SE&I to the responsible development organization (usually avionics/software). The main thing that most KBS should emphasize is that they should augment, not replace existing systems, and be initially used in an advisory mode. When used in an advisory mode the system can be evaluated with little risk.

Results of the DDT&E prototyping efforts will be used to assess the feasibility of onboard implementation. Recommendations relative to placement of functions on the ground and onboard will be documented. If the function is designed to go onboard then it will enter full scale development with the Avionics Development Facility (ADF) as its target. For that part of the application which is designated as ground software, an analysis will be conducted to determine hooks and scars, and a migration plan will be developed that is consistent with program guidelines.

Life Cycle Model

The waterfall life cycle model is the most commonly used methodology for development of software (DOD-2167, and the Software Management and Assurance Program (SMAP) 3.0). In the past KBS have been developed using "rapid prototyping", "iterative" or the "add rules till it works" development model. This method does not lend itself to requirements traceability, verification, validation, maintainability or reliability. This "iterative model" should not be confused with the spiral approach which is driven by risk reduction, rather than being "code driven". In a situation such as the space station program where the hardware and software are being developed simultaneously it should be possible to develop KBS using the flexible life cycle model allowed by SMAP, version 4.3. The most important thing this model allows is a period of controlled prototyping to define requirements. This pre-requirements "prototyping phase" is controlled, meaning that goals are defined a priori, and the results are documented in the appropriate documents. Four sub-phases will be repeated until the Analysis sub-phase fails to identify new tasks or problems requiring further prototype development. The four subphases are: analysis, knowledge acquisition, design and implement, and test and evaluation (figure 2).

During the analysis phase Software developers will determine the scope of the problem and the form of the probable solution(s). In later iterations, this analysis will be partially provided by testing from the previous pass. The product of this sub-phase will be a set of tasks which the prototype is required to perform at the end of the iteration. The knowledge acquisition phase will collect the knowledge that is required to carry out the tasks. A description of this knowledge will be the product of this sub-phase. In the design and implement sub-phase software developers will select algorithms, search routines, paradigms, and knowledge representations that are necessary to produce solutions from the knowledge acquired in the previous step. The series of design decisions and the final design solution are the product of this sub-phase. During the test and evaluation sub-phase software developers will design tests to verify that the prototype system is behaving as desired. The product of this sub-phase will include descriptions of the tests, and interpretations of the test results. If the prototype system is performing suitably, then its current functionality will be transmitted to the appropriate phases of the SMAP development model.

All other SMAP mandates and guidelines will be observed. Specifically, Verification and Validation of expert system application software will be incorporated into all phases of software development. Also, expert system application software will conform to the SMAP standards and practices with respect to: application software integration, simulations, testing, delivery, interfaces, and acceptance.

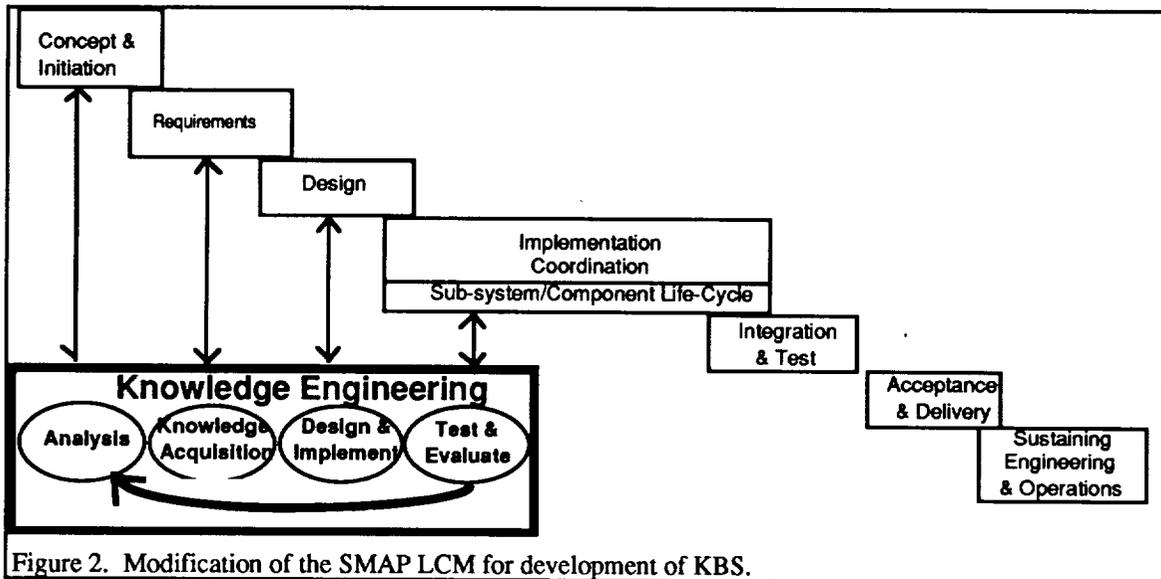
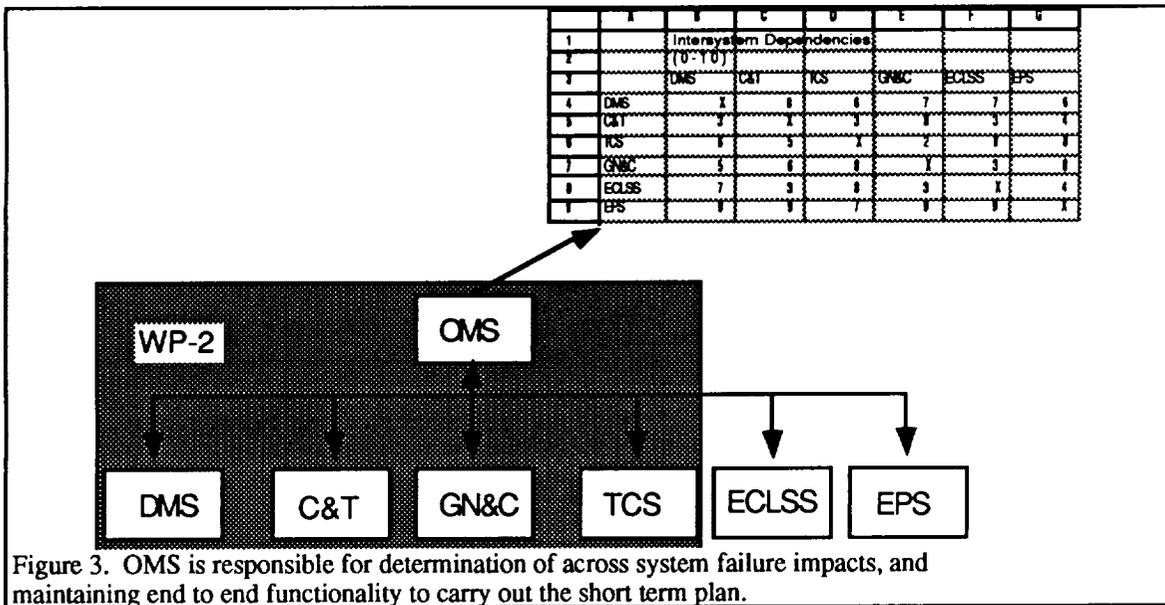


Figure 2. Modification of the SMAP LCM for development of KBS.

Given that SMAP 4.3 defines an acceptable life cycle model, the next major capability which needs to be in place is an acceptable KBS tool which will work within the capabilities and constraints in the SSFP: SSE, Ada, and SMAP. In the past most KBS products have been developed on symbolic processing computers using LISP. While at one time serious consideration was given to developing space qualified symbolic processors; this effort has been suspended. With the availability of KBS tools in ADA (CLIPS, ART and TIRS) this has become somewhat a moot issue; however, the integration of these tools into the SSF Data Management System (DMS) architecture is an open issue.

Systems Engineering

The space station has highly coupled and interdependent systems. Realizing that no one system can or should have to maintain a world model, the concept of the Operations Management System (OMS) was developed. The OMS consists of an onboard portion, the Operations Management Application (OMA), and a ground based portion Operations Management Ground Application (OMGA). The basic function of the OMS is to carry out the Operational Short Term Plan (OSTP) taking into consideration the stations current state (resources, and constraints). The OMS is also responsible for station wide FDIR. While each system is required to do internal diagnosis, and may for time or safety critical functions reconfigure autonomously, the OMS has to be responsible for determination of across system impacts, and final determination of the best reconfiguration. In the operation of the shuttle once one failure occurs, the immediate question is what is going to fail next; the OMS should help with that analysis. Figure 3 shows the relationship between the OMA and the systems. The OMS may also have to deal with multiple automated systems using KBS techniques. If that ends up being the case then some type of fusion will be needed; one popular KBS technique is that of a blackboard [7]. Blackboards provide a global data base for multiple expert systems to access asynchronously, and allow cooperative problem solving. The theory of systems [8] should be used in the development of automated systems in a much more methodical fashion, for example decomposition, criticality analysis, information requirements analysis, and function and requirements allocation tools should be used to help define well engineered automated systems [9]. How systems theory can be applied to fault diagnosis is covered in [39].



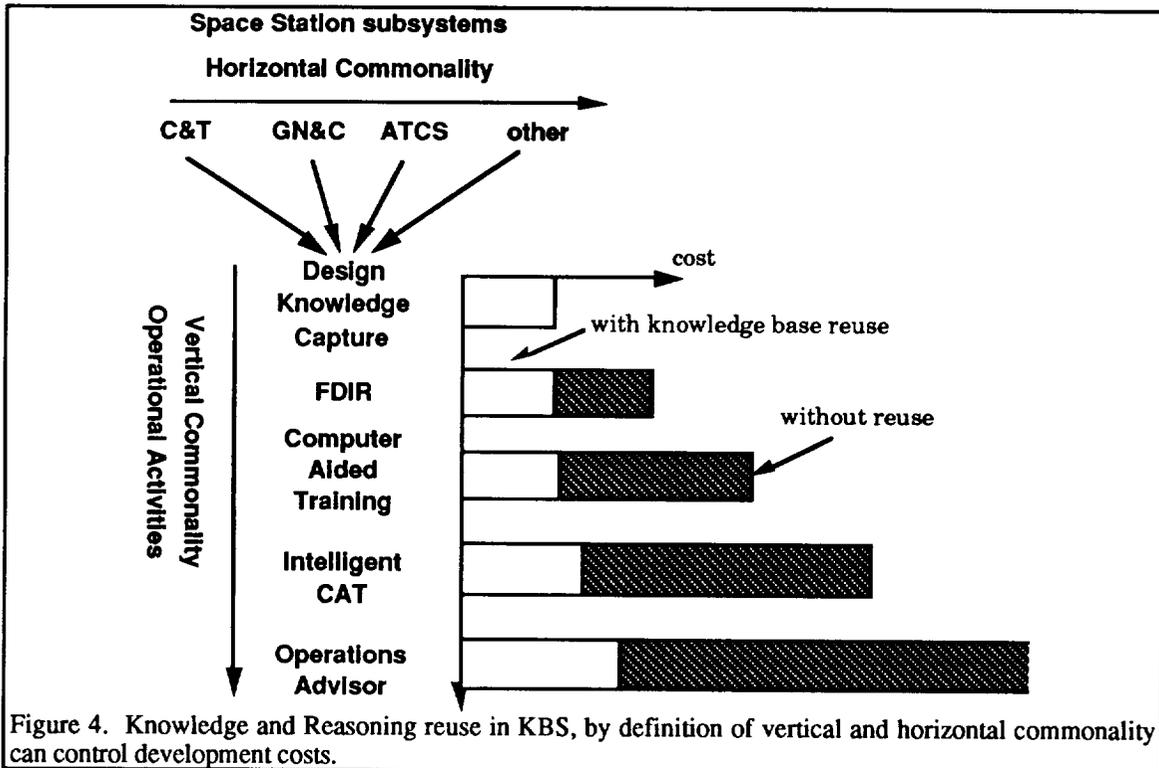
Software Engineering

Just as object oriented programming principals (i.e. software engineering) have popularized the ideas of abstraction, objects, maintainability, and reusability, it appears that KBS S/W can also be engineered to have the same desirable characteristics. There are two sorts of KBS reuse: knowledge reuse, and reasoning (or inference) reuse (see Figure 4). Once the knowledge has been acquired in a useful form it can be used for a variety of applications such as Fault Detection Isolation and Recovery (FDIR), Intelligent Computer Aided Training (ICAT), Planning and Scheduling (P&S), and operations advisors. Once a way of doing any of these applications is determined, it is possible to use the same inference mechanisms in a variety of systems. Model Based Reasoning (MBR) has popularized this idea for FDIR, for example fluid, heat and electrical flows are all similar and obey the same basic physical laws: Kirchoff's current and voltage laws (i.e. conservation of energy). So once you know how to deal with a open or closed resistor by extension you know how to deal with a stuck open or closed valve in a fluid system. MBR also brings a host of other useful tools to the table, such as constraint propagation, and suspension, and qualitative modeling [10-19, 41-42]. The biggest open issue in MBR is what level of modeling is best suited for a particular problem. The simulations being developed fit into 4 categories: 1.) highly accurate mathematical model with real dynamics, 2.) very accurate model, but without all of the coupling between elements, 3.) table driven interface checking, 4.) very low fidelity. One of the big arguments with model based reasoning is that it is computationally excessive, for example the Thermal Expert System (TEXSYS), which implemented DeKleer's algorithm [18] for dealing with multiple faults, required 155,908 Source Lines Of Code (SLOC), and a variety of computers (symbolic and traditional) to perform the monitoring and control of the thermal testbed. While what TEXSYS accomplished was impressive (FDIR of 7 system level faults, and 10 component level faults), it should be considered as only having scratched the surface of what MBR can accomplish.

In almost all cases the most useful knowledge representation for physical systems is one based on structure and behavior. Such a model is much more flexible, and multipurpose than rules, decision trees, or a frame based system. Qualitative modeling can view components at a variety of resolutions, for example in electronics: a diode can be considered from the following viewpoints: as a binary switch, or as a linear approximation, a nonlinear approximation, in terms of electromagnetic fields (solving Maxwell's equations), going to the atomic level and considering quantum mechanics, or considering relativistic effects (quantum electrodynamics). In most cases the first two models mentioned are sufficient. Qualitative modeling also allows one to view systems (connected components) at different levels of detail, for example an operational amplifier can be considered in terms of inputs, and outputs voltages, and amperages, or as a differential amplifier. One is a result of the physical makeup of the device, the second interpretation results from the fact that it is an engineered device designed to perform a function.

Automation Integration

Besides architectural compatibility and interfaces the major integration efforts for KBS must be in the area of Verification and Validation (V&V). Traditional S/W V&V is largely a heuristic art that involves path checking and test generation. The two main methods are the black box and glass box methodologies. The black box is the interface checking functional requirements and limit checking verification, while the glass box is more concerned with how the functions are accomplished, exercising all branches, and code walk throughs. The importance of modularizing a program or rules can not be over emphasized, without it one is faced with a combinatorial explosion of possible paths.



Current research gives many clear directions on how to do V&V of rule based systems [20-26]. Rushby [25] develops the idea of "minimal competency" which states: while an optimum solution may be hard to define or verify, it is possible to define quite sharply what the minimum level of acceptable behavior is. Nguyen's CHECK program [22] checks the consistency of the knowledge base in the following fashions: redundancy, conflict, subsumption, unnecessary if rules, circularity, unreferenced and illegal attribute values, unreachable conclusions and dead end goals or if conditions. Stachowitz [21] has extended these conditions to deal with certainty factors [23]. The only difficulty is the integration of these techniques into a tool which is usable in the SSFP. One of the biggest stumbling blocks in the way of using rule based systems is that these V&V concerns are all trying to address the basic problem that rule based systems are not bounded in time and space computationally because of the way the underlying data structures are setup (Rete or Treat networks).

Since decision trees are derived from data, the data must be accurate; however, generating Ada code from a decision tree is straightforward, and therefore the V&V should be easily accomplished using Ada V&V methods. Model Based Reasoning can be done in procedural languages, so ADA should provide no problem however the lack of tools does. The issue of V&V for MBR systems is basically to make sure the simulation used is accurate. This simulation can be considered the knowledge base. The reasoning techniques which interface with these simulation knowledge bases should be verified and validated separately (just as rules and their interpreter in expert systems are validated separately).

The other issue involved in the integration of KBS is interfacing with Ada procedural code. Since there are a variety of Ada KBS tools (CLIPS, ART, and TIRS) this should not be a major problem.

Diagnostics

Currently most of the space station's systems have baselined Built In Test/Built In Test Equipment (BIT/BITE) as being the solution to performing FDIR. The main advantage of this technique since it is performed in hardware, is its quickness. While this is a popular technique, the shortcomings of this approach are fairly well known: false alarms, can not duplicates, retest okay, and failure to diagnose. In fact the unreliability of BIT/BITE has prompted the generation of measures of how poorly it performs [27], as noted "the experience with automatic detection and isolation systems, in the form of BIT, has not lived up to expectations. False alarm, false isolation, and failure to diagnose errors are reported as a result of system diagnostic inadequacies. With the existence of such errors, the evaluation of the operational capability of BIT/BITE becomes a real challenge."

While BIT/BITE is not useless, it should definitely not be the only technique available for FDIR. A more intelligent approach which considers the overall systems state will help remove false alarms by verifying that each Orbitally Replaceable Unit (ORU) is receiving its necessary inputs with no noise or bad signals. It is extremely important that the inputs and outputs of each ORU are well instrumented to help with eventual automated diagnosis. Failure to diagnose errors in BIT/BITE can be solved by considering long term trends such as sensor failures and calibration errors (this is a 30 year not 10 day mission).

It would appear that using BIT/BITE by itself won't solve our problems. We could create a "fault dictionary" by using a simulation and a list of the kinds of faults anticipated. This can be considered an automated FMEA/CIL search table. This results in a list of fault/symptom pairs, we can then invert this list in a variety of ways: Bit string register level comparisons [28], machine learning [29], or even a rule based expert system [30]. The problem with this approach when used on any reasonably complex system (a space station for example) is that the creator of the diagnostics must settle on a small, fixed class of expected faults so that acceptable fault coverage (statistically likely), isolation, and computing efficiency can be realized. A fault is predefined, while it actually should be "anything other than the intended behavior" [12]. Again it should be emphasized here that this sort of diagnosis is good as a second layer on top of BIT/BITE to help with some of the ambiguities, and inconsistencies that arise, but should not be the only answer.

Perhaps decision trees [31-33] could help with writing down an efficient sequence of tests and conclusions to guide a diagnosis. Decision trees are useful in other aspects as well; for example, they can help flag useless and redundant tests, and depending on the algorithm used to generate them usually give the most efficient sequencing of tests available based on information theory (entropy), but even this heuristic can be in error by as much as 30 percent from the optimal solution [32]. Again the point is that they are a way of writing down an answer which is already known.

Rule based expert systems provide an efficient computational mechanism [34] for using the knowledge of expert troubleshooters. This approach is more intuitive, and perhaps more in tune with the actual operation of the system than "fault dictionaries"; however, there is a strong system dependence (a new set of rules is needed for every system), and minor changes in the system (upgrades) often render an entire knowledge base obsolete, and the time needed to acquire the knowledge makes it hard to deliver a rule based expert system at the time of delivery of the actual system (it hasn't been operated enough for technicians to figure out how to diagnose it). A more subtle failing in rule based systems is a lack of clarity, a rule saying "IF voltage1=100 and voltage3 = 0 THEN resistor1 = open" gives no clue as to how the components are connected or where these measurements are taken. This approach does not lend itself to maintainability (without a schematic the rule is gibberish). Another failing of rule based systems is that often since there are no requirements or specifications, failures often go unnoticed since there is no idea of what the "correct" behavior should be. Many rule based "FDIR systems" do nothing more than sensor verification; however, once a failure of a sensor is identified they are often incapable of performing any useful diagnosis due to a lack of complete information [19].

The approach which is suggested here would combine elements of the previous approaches, but would use as their backup Model Based Reasoning (MBR). Research in MBR has developed many

mathematically sound methods for doing diagnosis, and has made significant progress in dealing with some of the more difficult fault diagnosis problems: multiple faults, fault masking, unexpected causality (dripping pipes for example or a solder bridge), unanticipated failure modes, and faulty instruments. None of which can typically be dealt with in other FDIR systems. The research in this area has developed many sharply focused techniques [10-19].

The initial efforts in Fault Tolerance Redundancy Management (FT/RM) have used Digraph Matrix Analysis (DMA) to capture connectivity or causality among components and to help develop and verify the design of redundancy management algorithms and fault tolerant systems [35]. DMA can also help in the determination of overall system reliability when used in conjunction with Failure Modes and Effects Analysis Critical Item Lists (FMEA/CIL) work sheets. DMA will expose points of failure that have system wide or safety critical consequences. The DMA tool uses the Warren algorithm [36] for transitive closure to compute reachability. This reachability analysis is typically the first step taken when using a MBR system.

Another tool being used in this effort is the Systems Testability Analyzer (STA) which is part of the Integrated Diagnostic Support System (IDSS) which has been developed in Ada by Harris Corporation for the NAVY. It uses a systems design to determine testability and ambiguity groups, and help improve fault detection probability. The testability analysis flags areas where ambiguity groups arise (you can't figure out which component is faulted), and suggests either further test in operations, or where to put additional instrumentation during design [37]. The Adaptive Diagnostic System (ADS) utilizes an opportunistic approach to doing FDIR. It starts with the simplest model, i.e. compiled, lookup table knowledge, and proceeds through several intermediate stages until it does a full simulation. Two of the intermediate knowledge levels are: analytical and logical. The analytical approach uses dependency models, fault signatures, and BIT information to detect and isolate faults. The empirical approach uses production rules, to help mask BIT/BITE false alarms. ADS can also update its own statistical parameters to reorganize its search more effectively dependent on real world behavior of the system. The algorithm used to generate the decision trees in STA are based on Artificial Intelligence (AI) search techniques and information theory [31]. Ways in which MBR can extend these tools are being investigated.

Planning

Another area where automated systems could save a lot of manpower, is in planning and scheduling the resources, manpower, and experiments on the SS. Planning and scheduling has received a lot of attention in the literature [43-49]. This problem is basically intractable in the following ways: job shop scheduling is NP-Hard, creating an optimal schedule is exponentially related to the size of a given problem, determining if an arbitrary plan is feasible is NP-Complete, and determining an admissible first step is NP-Hard [43-44]. The way in which constraints are represented can have a major impact on how well solutions can be determined [45]. A Computer Assisted Scheduling System (COMPASS) has been developed in Ada, using XWindows for the operator interface, which has many features desirable for NASA programs such as: discrete and continuous resources, returnable and consumable/produced resources, and can reason about state dependent activities including both boolean and real valued state variables. This system allows interactive, mixed initiative scheduling in several different contexts, multi-level scheduling, multi-interval scheduling, and multi-agent scheduling [46].

Current Status and Future Plans

Honeywell continues in the development of the Maintenance Diagnostic System (MDS) which will augment the existing FDIR system in the Attitude Determination and Control System (ADCS) by isolating faults, aiding in preventive maintenance, and maintenance instruction [38]. A port from Sun workstations to PS/2s has also been accomplished. The predictive maintenance aspect takes advantage of empirical relationships, such as the fact that in the lasers in the ISA ring gyros have a fault signature that can be recognized 2-3 months in advance. It turns out that if one plots the lasing power against the input current there is a characteristic curve. However, when a laser starts to go bad its performance strays from this curve in a fairly predictable fashion. It is this sort of trend analysis that KBS are ideally suited for.

As part of MDSSC's participation in the Advanced Automation Methodology Project (AAMP), which is defining engineering methodologies and standards for developing KBS in the SSFP, the recovery

part of an FDIR system for the Space to Ground Communication link is being developed. With the possible movement of the FDIR functions to the ground at Permanently Manned Configuration (PMC) this may be one of the most important links for the space station. This project is using existing tools already developed at Johnson Space Center (and possibly General Electric), and augmenting them by adding the recovery procedure generation function. An IR&D for doing ICAT on the C&T system is being initiated this year.

An effort to enhance the Active Thermal Control System (ATCS) simulation with an external control architecture is being initiated. This is an outgrowth of the Thermal Expert System (TEXSYS) demonstration project. This task order will initially involve MDSSC Thermal personnel in developing the simulation and interfaces, and LMSC in developing the control algorithms, and identifying interesting (not easy to identify) fault modes. This system will be designed with an interface so that an external control system can be implemented (either human, or some automated system).

The DMS contractor (IBM) has been asked to perform a common space station expert system services trade study. This task would involve polling the space station community to provide inputs for a white paper on common expert system services (emphasis on FDIR). They have also been asked to develop a DMS system management FDIR function using KBS techniques.

QMR and DXPlain are two medical diagnosis expert systems which are being considered in a trade study being undertaken by Crew Health Care System (CHeCS) personnel to pick a diagnosis system for the station. A medical diagnosis expert system is important because there will not be a physician immediately available all the time on the ground, and because there may be life threatening conditions when ground support is unavailable (C&T or the Tracking Data Relay Satellite System failed).

Conclusions

This paper has described a technical approach which should allow the integration of KBS into the SSFP. It is important to remember that KBS are actually "people amplifiers" and should be used to augment both the human and machines capabilities [40]. The current status of several SSFP KBS projects was briefly reviewed. The efficient usage of KBS technology in the SSFP should make the operation of the station much easier and cost effective.

References

- [1] Ferguson, Mary "Machine Intelligence and Robotics at Johnson Space Center" JSC-23518, September 1989.
- [2] Erickson, J. D., "Intelligent Systems in NASA Space Missions" to be published: Proceedings international conference on supercomputing in Nuclear Applications, Mito City, Ibaraki, Japan, March 1990.
- [3] MDSSC-KSC "KSC Lessons Learned Applicability to Space Station Freedom Support of Exploration Scenarios" Onorbit Assembly/Service Task Definition Study.
- [4] MDSSC-SSD "Automation and Robotics Plan" MDC-H4115, October 1989
- [5] Flaherty, D.R. "Automation and Robotics Plan" (DR-17), MDC H2036A, June 1986
- [6] Wood, R.M., McKee, J.W., Kuck, G.A., Flaherty, D.R. "High-Leverage Advanced Development Projects to meet Space Station Requirements" MDC H1479, October 1985
- [7] Nii, P. H., "Blackboard Systems" The handbook of Artificial Intelligence Volume IV, Barr, A., Cohen, P. R., and Feigenbaum, E. A. (eds.), Addison Wesley Publishing Co. Inc., New York, NY, 1989.
- [8] Klir, G. J., "Architecture of Systems Problem Solving" Plenum Press, New York, NY, 1985.
- [9] Smith, K. M., and O'Neil, G., "Application of General Systems Theory for the Definition and design of Advanced Avionics Systems" Tutorial at Digital Avionics Systems Conference, San Jose, California, October 17-20, 1988.
- [10] Bobrow, D.G., "Qualitative Reasoning about Physical Systems" MIT Press, 1985.
- [11] Davis, R., "Form and Content in Model Based Reasoning" presented at Workshop on Model Based Reasoning AAAI/JCAI-89, Detroit, MI, August 1989.
- [12] Davis, R., and Hamscher, W., "Model Based Reasoning: Troubleshooting" MIT AI Memo No. 1059.
- [13] Davis, R., "Robustness and Transparency in Intelligent Systems" Symposium on Human Factors, National Academy of Sciences, Washington, DC, January 29-30, 1987.
- [14] Reiter, R., "A Theory of Diagnosis from First Principals" Artificial Intelligence, 32, 57-95, 1987.
- [15] Raiman, O. "Order of Magnitude Reasoning" AAAI-86, Philadelphia, Pennsylvania. San Mateo: Morgan Kaufmann Publishers.
- [16] Malin, J. T., Basham, B. D., and Harris, R. A., "Use of Qualitative Models in Discrete Event Simulation for Analysis of Malfunctions in Continuous Processing Systems" chapter in "Artificial Intelligence in Process Engineering (M. Mavrovouniotis, ed.), Academic Press, 1989.

- [17] Forbus, Kenneth "Qualitative Physics: Past Present, and Future" AAAI-88 Tutorial, pp. 239-296.
- [18] de Kleer, J., and Williams, B. C., "Diagnosing Multiple Faults" *Artificial Intelligence*, 32, pp. 97-130, 1987.
- [19] Fulton, S. L., and Pepe, C. O. "An introduction to Model-Based Reasoning" *AI Expert*, pp. 48-55, January 1990.
- [20] Bellman, C. L., and Walter, D. O., "Analyzing and Correcting Knowledge Based Systems requires Explicit Models" AAAI-88 Verification and Validation Workshop
- [21] Stachowitz, R. A., "Validation of Knowledge Based Systems" Second AIAA/NASA/USAF Symposium on Automation, Robotics, and Advanced Computing for the National Space Program, March 1987.
- [22] Nguyen, T. A., Perkins, W. A., Laffey, T.J., and Pecora, D. "Knowledge Base Verification" *AI Magazine*, 8(2), pp. 65-79, Summer 1987.
- [23] Stachowitz, R. A., et. al. "Building validation tools for knowledge based systems" first annual workshop on Space Operations, Automation and Robotics, pp. 207-216, NASA conference Publication 2491, Houston, TX, August 1987.
- [24] Culbert, C., Riley, G., and Savely, R.T., "Approaches to the Verification of Rule Based Expert Systems" SOAR 1987
- [25] Rushby, J., "Quality Measures and Assurance for AI Software" NASA contractor report 4187, 1988.
- [26] Landauer, C., "Principles of RuleBase Correctness" AAAI/IJCAI-89 Workshop on V&V.
- [27] Aly, N. A., Aly, A. A. "Measures of Testability for Automatic Diagnostic Systems" *IEEE Transactions on Reliability*, 37(5), pp. 531-538, December 1988.
- [28] Feagin, T. "Real-Time Diagnostic Expert Systems using Bit Strings" To be published.
- [29] Pearce, D., "Induction of Fault Diagnosis Systems from Qualitative Models", AAAI-88, pp. 353-359.
- [30] Gilmore, J. P., and Gingher, K. "A Survey of Diagnostic Expert Systems" *Applications of Artificial Intelligence V, Proceedings of SPIE - The international society for Optical Engineering*, Vol, 786, pp. 2-11, 1987.
- [31] Pattipati, K. R., Deckert, J. C., and Alexandris, M. G. "Time-Efficient Sequencer of Tests (TEST)" *IEEE 1985 Autotestcon proceedings*, pp. 49-62, 1985.
- [32] Schumacher, H., and Sevcik, K.C. "The synthetic Approach to Decision Table Conversion", *Communications of the ACM*, 19(6), pp. 343-351, 1976.
- [33] Jackson, A. H. "Machine Learning" *Expert Systems*, 5(2), PP. 132-150, May 1988.
- [34] Forgy, C. L., and Shepard, S. J. "Rete: a fast match algorithm" *AI Expert*, pp. 34-40, January 1987.
- [35] Sacks, I. J., "Digraph Matrix Analysis" *IEEE Transactions on Reliability*, Vol. R-34, No. 5, December 1985.
- [36] Warren, H. S., "A Modification of Warshall's Algorithm for the Transitive Closure of Binary Relations" *Communications of the ACM*, 18(4), April, 1975.
- [37] Rosenberg, B. J., "The Navy Integrated Diagnostic Support System - System Overview, Architecture, and Interfaces", Harris Corporation, 6801 Jericho Turnpike, Syosset, New York 11791.
- [38] Toms, D., Hadden, G., and Harrington, J. "Attitude Determination and Control System Maintenance Diagnostic System" These proceedings
- [39] Narayan, H., and Viswanadham, N., "A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems" *IEEE Systems, Man and Cybernetics*, 17(2), pp. 274-288, March/April 1987.
- [40] Woods, D.D., "Cognitive Technologies: The Design of Joint Human-Machine Cognitive Systems" *The AI Magazine* pp. 86-91, 1987.
- [41] Clancey, W. J., "Viewing Knowledge Bases as Qualitative Models" *IEEE Expert* Summer 1989, pp. 9-23.
- [42] Clancey, W. J., "Heuristic Classification" *Artificial Intelligence*, 27, pp. 289-350, 1985.
- [43] Fox, B.R., "Minimally Ordered Sets" to appear in *Progress in Robotics and Intelligent Systems*, C. Y. Ho and G. Zobrist (eds.), Ablex Publishing, Norwood, New Jersey, 1989
- [44] Ullman, J.D., "NP-Complete Scheduling Problems" *Journal of Computer and System Sciences*, 10, pp. 384-393, 1975.
- [45] Fox, B.R. "Representational Adequacy in Temporal Reasoning" Submitted to the First International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Canada, May 15-18, 1989.
- [46] Fox, B.R. "Mixed Initiative Scheduling" to be published.
- [47] Fox, M.S., and Sadeh, N. "Preference Propagation in Temporal/Capacity Constraint Graphs", CMU-CS-88-193.
- [48] Allen, J.F. "Maintaining knowledge about temporal intervals" *Comm. of the ACM*, 26(11), pp. 832-843, 1983.

A Development Framework for Artificial Intelligence Based Distributed Operations Support Systems

Richard M. Adler and Bruce H. Cottman

Symbiotics, Incorporated
875 Main Street
Cambridge, Ma 02139
(617) 876-3633

Abstract

Advanced automation is required to reduce costly human operations support requirements for complex space-based and ground control systems. Existing knowledge-based technologies have been used successfully to automate individual operations tasks. Considerably less progress has been made in integrating and coordinating *multiple* operations applications for *unified* intelligent support systems. To fill this gap, we are constructing SOCIAL, a tool set for developing Distributed Artificial Intelligence (DAI) systems. SOCIAL consists of three primary language-based components defining: models of interprocess communication across heterogeneous platforms; models for interprocess coordination, concurrency control, and fault management; and for accessing heterogeneous information resources. DAI application subsystems, either new or existing, will access these distributed services non-intrusively, via high-level message-based protocols. SOCIAL will reduce the complexity of distributed communications, control, and integration, enabling developers to concentrate on the design and functionality of the target DAI system itself.

Introduction

Operational support of complex space-related systems currently entails expensive manpower requirements. Human labor costs are particularly high in manned space systems such as the Space Shuttle and the planned Space Station: in these remote settings, scarce manpower that is dedicated to operational support cannot be allocated to primary mission objectives. The economic viability of increasingly advanced space systems hinges on significant increases in operational support automation [Ba88].

Standard engineering formalisms such as control theory and operations research can be used to automate simple control, monitoring, and scheduling tasks. However, such methods do not generalize readily to non-routine contexts: assessing and responding to system failures; revising plans in the face of unforeseen conditions; and similarly difficult cognitive tasks. Over the last several decades, artificial intelligence (AI) researchers have addressed these problems by developing symbolic modeling and automated reasoning techniques. These methods offer superior flexibility and generality for modeling human analytic and decision-making processes and for solving combinatorially complex problems.

Expert systems, model-based reasoning, and other knowledge-based tools and methods have been applied to automate tasks including fault detection and diagnosis, planning and scheduling, data analysis, and information storage and retrieval. Several important prototypes systems developed in recent years are being extended and validated in field tests, in preparation for integration into existing operational support systems for complex networks [Ad89b, Br89, Ba88, Mu89, Ru88].

Integrating and coordinating *multiple* knowledge-based applications related to a common domain are critical problems that have received little attention until recently [Ad89a]. Existing intelligent applications for operations support rely on system-specific interfaces to users, data feeds, databases, and conventional automation software. These "standalone" systems also lack access and control facilities for working together cooperatively on clearly related operations tasks, such as intelligent diagnosis and error-tracking. As increasing numbers of intelligent support tools are deployed together in common domains, the need for effective tools for integrating such systems into a unified cooperative framework will become critical.

This paper describes SOCIAL, a development framework for distributed systems that is intended to fill the technology gap. SOCIAL consists of three primary language-based tools: MetaCourier supplies functionality for interprocess communication and control access across heterogeneous platforms; MetaAgents defines control models for interprocess organization, data replication, concurrency management, and fault

detection and recovery; *MetaViews* defines a uniform data model for accessing and controlling persistent information stores such as data and knowledge bases. New and existing application elements access these distributed services non-intrusively, via high-level message-based protocols. *SOCIAL* thereby reduces the complexity of distributed communications, control, and integration, enabling developers to concentrate on the design and functionality of the target system itself.

The next two sections of the paper define the central system integration issues that *SOCIAL* addresses and review related research. Next, *SOCIAL*'s architecture and user model are described and illustrated with a hypothetical operations support example. The remaining sections outline the design and functionality of *SOCIAL*'s primary language-based subsystems.

Integrating and Coordinating Heterogeneous Intelligent Systems

Several basic problems arise in integrating and coordinating multiple knowledge-based systems related to a common domain. First, different activities within a domain such as operational support generally depend on different kinds of knowledge, skills, tools, and methodologies. Knowledge-based automated assistants tend to require correspondingly diverse representation, reasoning, and internal control models. Integrating such applications thus requires methods for reconciling or accommodating heterogeneous internal architectures.

Second, different tasks within a given domain, while distinctive in many respects, frequently display important commonalities. For example, network operators and managers share background information and expertise concerning configuration procedures, although their respective depth and application of such knowledge may differ. A framework for integrating multiple intelligent applications in a given domain must facilitate sharing of knowledge resources, including symbolic models of domain structures, behavior, and operational expertise. Other resources of common utility across applications include interfaces to: users; databases; target system data feeds and command/control effecters; and conventional software for data analysis, performance monitoring, and (low-level) automated process control and safing systems.

Third, the integration strategy must be non-intrusive. Existing "standalone" knowledge-based and conventional programs and data resources represent significant investments in capital equipment, software development, and safety (i.e., from prior validation and verification). It would be prohibitively expensive to discard such resources or to re-engineer them extensively.

Fourth, applications and resources are generally distributed across heterogeneous software and hardware platforms connected by one or more (local area) networks. A generalized communications capability is needed for data exchange and control access across intelligent applications. Moreover, this functional capability should be accessible through a modular, high-level interface: minimizing the visibility of the mechanics of distributed communication fosters maintainability of application code and accessibility for developers unversed in exotic communication protocols.

The final and perhaps most critical problem is establishing cooperation between knowledge-based applications once they are integrated into a unified framework. Coordination presupposes that applications somehow know about one another, their respective capabilities, activities, intentions, and needs. In addition, coordination also presupposes control and communications models for exchanging requests, commands, suggestions, beliefs, and other information. Again, to facilitate maintainability and extensibility, it is important that application models and interprocess control mechanisms be partitioned from one another *and* from distributed communication functionality.

Related Work

Distributed Artificial Intelligence (DAI) deals with the solution of complex problems by networks of autonomous, cooperating computational processes [Hu87]. These processes, often called agents, can be distributed physically across computational resources and logically across an organizational structure. Typically, cooperation is mediated by message-passing communication between agents.

DAI research to date, has focused almost exclusively on domains in which single organizations of agents cooperate to solve *single* complex tasks [Bo88], including data fusion [Le83] and speech understanding [Bs87]. These "single problem" DAI research efforts have concentrated on developing complex *local* control structures for coordinating a network of homogeneous agents to converge to globally consistent problem solutions. For example, intelligent schedulers prioritize local agent tasks for execution according to heuristics or metrics

that gauge probable global problem-solving effectiveness [Le83]. More complex planners create, order, and filter agent tasks adaptively, based on hierarchies of local and global problem-solving goals [Ha86].

Single problem DAI architectures, while suggestive, are not directly applicable to the integration problems described above. DAI research has generally assumed: a single logical organization of homogeneous complex agents, such as distributed blackboards; correspondingly uniform models for intra- and inter-agent communication and control; and homogeneous software and hardware platforms [Hu87,Bo88,Ja89]. All three assumptions are violated in the complex DAI environments of interest here.

Recently, DAI research has broadened to consider domains such as operations support and battle management, which encompass collections of related problems of varying complexity. While requisite problem-solving skills, knowledge and data resources may overlap considerably, the solutions to problems in these domains may be independent or only weakly dependent upon one another. These characteristics favor coarser-grained, more loosely-coupled DAI architectures, comprised of individual agents and organizations of agents that focus on particular problems or problem sets disjoint from one another. A useful human analogy is a legal or medical practice of consultants with different areas of specialized expertise.

Fine-grained scheduling and planning of inter-organization activities tend not to be critical issues in these domains because agent organizations only depend weakly on one another. Instead, the critical design issues are: (a) to integrate agents and agent organizations bounded by different knowledge representation, reasoning, control, and communication models; and (b) to access and integrate existing conventional software and data resources.

Initial "multiple problem" DAI applications have failed to address all of the issues raised in the last section in a generalizable manner. For example, KB-BATMAN integrates three intelligent decision aids for a military tactical command. However, the subsystems only communicate indirectly, through pairwise interactions with a shared relational data base and in a fixed, predefined control pattern [Nu88].

OPERA assists in operations support for NASA's Space Shuttle Launch Processing System [He87,Ad89b]. Its hierarchical blackboard architecture successfully integrates and coordinates heterogeneous expert systems, which share external interfaces and knowledge bases [Ad89c]. However, OPERA applications are all co-resident (i.e., physically *non*-distributed). Knowledge bases are restricted to a common representational model. OPERA also lacks generalized tools for handling errors and accessing data feeds or databases.

Several DAI development tools support integration of intelligent applications with heterogeneous organizational models. ABE and AGORA provide predefined models for inter-organizational control (e.g., dataflow, blackboard, transaction-based) [Bs87,Ha88]. ABE also supplies a high-level graphic editor and an interface to a commercial relational database management system. AGORA uses a shared-memory communication model, while ABE uses message-passing. Both tools employ virtual machine models that map onto particular platforms and network communication services (e.g., MACH, Chaosnet). MACE [Ga86], a message-based DAI testbed incorporates an elegant declarative language for modeling agents' roles, skills, goals, and acquaintances. However, MACE offers limited tools for coordinating multiple agent organizations and lacks support for heterogeneous processing platforms.

Architecture of the SOCIAL DAI Development Framework

SOCIAL is a generalized framework for developing both single and multiple problem DAI applications. Its architecture, shown in Figure 1, consists of a layered, partitioned set of system building blocks and development interfaces.

Developers use the high-level Application Interface to access predefined object classes, called *Types*. Each Type represents a different, generic DAI control skeleton for intelligent agents or agent organizations. Organization Types are skeletons for agents whose logical functions are to coordinate a collection of agents (i.e., organizational members), and to manage their communications with outside agents and organizations.

DAI systems are constructed by instantiating (or specializing and instantiating) suitable agent Types and embedding application elements within those "wrapper" objects. Application elements access the distributed services of its embedding Type instance through discrete high-level message-based *Protocols*. A given DAI system can integrate multiple heterogeneous agents and agent organization Types.

Agent Types are structured as an inheritance hierarchy of object classes, whose initial subclasses are

shown in Figure 2. Discrete application elements (e.g., knowledge sources), are embedded in basic Receptionist agent skeletons. Specialized subclasses of the Receptionist, called Gateways, are instantiated for embedding protected knowledge or data bases. The Manager Type is the root Agent Organization class. Manager subclasses include variant blackboard architectures and other organizational models such as have been developed in single problem DAI research. These Types are described further in the MetaAgents section of this paper.

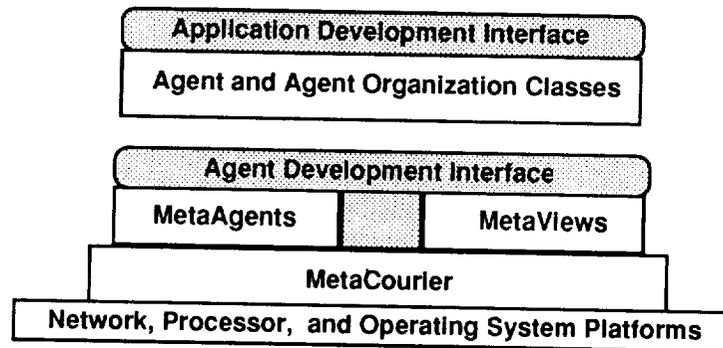


Figure .1: SOCIAL Architecture

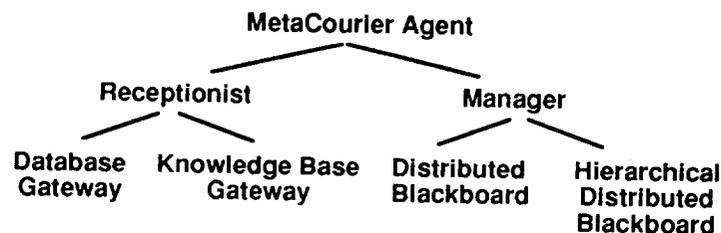


Figure .2: Agent Types in SOCIAL's Application Library

Each Type is comprised of other kinds of objects called *Models*, which define different aspects of distributed behavior. Models are accessed through a separate Agent Development Interface. At present, SOCIAL describes three types of Models, which are represented in terms of compilable object-oriented languages. The MetaCourier language, SOCIAL's basic substrate, defines a class of Models for distributed communications. MetaAgents defines a class of intra- and inter-process control Models for agent and agent organizations. The MetaViews language defines a class of Models for accessing different models of data and knowledge. Both languages exploit MetaCourier's distributed communication services.

In effect, developers use the Application Interface to access a *library* of predefined DAI building blocks. Most of these objects can be customized by setting mode switches that override default services such as error-handling behavior. Applications may sometimes require service options or new behaviors not provided by the library of existing agent Types. In these situations, the dedicated languages comprising the Agent Development Interface can be used to extend the library by defining specialized Models and combining them to create new agent Type subclasses.

Operations Support using SOCIAL

A hypothetical example of a DAI operations support system based on SOCIAL is illustrated in Figure 3. The target domain is a distributed ground control network such as a launch processing system, consisting of user consoles, computers, data links, ground support equipment, and embedded sensors. Sensor monitor programs would be realized as Receptionist agents, with asynchronous or synchronous communication Models, depending on individual polling requirements. A relational database for tracking problems would be integrated using a Gateway agent. A Blackboard-based data fusion Manager Agent would coordinate sensor polling, measurement interpretation, and anomaly detection. A diagnoser agent would generate and test

fault hypotheses and issue recovery suggestions to an Executive Manager Agent. Operations users would view ongoing activities and issue queries or commands to the Executive and database Gateway agents.

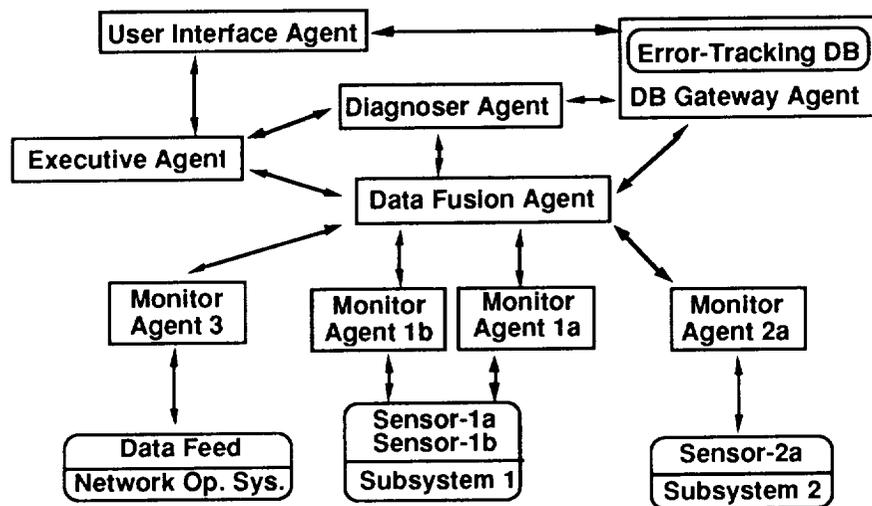


Figure .3: Hypothetical DAI System for Operation Support based on SOCIAL

The remaining sections describe SOCIAL's underlying languages and Models, which enable Agent Types to provide distributed services for integrating and coordinating DAI application elements.

MetaCourier: A Language for Distributed Communication

Advanced operational support architectures for space and ground control systems will have to integrate emerging hardware and software technologies with existing applications (both conventional and intelligent), interfaces, languages, and hardware platforms. Cost and reliability concerns dictate an integration strategy that minimizes intrusive modifications to existing system elements and allows them to be maintained and enhanced independently. Moreover, this strategy should maximize portability, to enable migration of system components to newer, high performance processor platforms. Technology transfer and management risks are also minimized, by reducing adjustments to training and operational procedures, and standdowns for system replacement and validation.

MetaCourier is a high-level object-oriented language for distributed communication that is designed to achieve these system integration objectives [Pa88]. The leading alternative communication model, based on the Remote Procedure Call (RPC) facility, is asymmetric and pairwise-restricted: an active *client* process invokes one (and only one) passive *server* process, which responds as required. In contrast, MetaCourier services provide fully peer to peer transparent communication between distributed applications.

The MetaCourier language defines four major object classes, Agents, Environments, Hosts, and Messages. *Agents* are intelligent, self-contained, autonomous processes. *Host* object attributes characterize the structure of network nodes: their processors, operating systems, peripherals, network types and physical addresses. *Environments* depict software dependencies for Agents, such as language compilers, and editors. Environments can be specialized to enhance communication performance for particular data types (e.g., sparse arrays), by defining custom encoding and decoding methods.

A MetaCourier *Message* defines the specific distributed communication behavior used by an Agent when it executes in an Environment. Both asynchronous and synchronous message-passing Models are available. An application Agent communicates with another by formulating a Message using the relevant Model protocol, for example:

```

Asynchronous: (Tell :agent sensor-monitor :sys Symb1 "(poll measurement-Z)")
Synchronous: (Tell-and-Block :agent user-interface :sys Hac2
              "(trigger-alarm sensor-1 window-2)")
  
```

MetaCourier handles message routing, transmission, and delivery services transparently to the source and target agents' associated applications. Conceptually, the Agents' associated Hosts and Environments act as filters that manage processing and network dependencies in the communication process (cf Figure 4). Distributed control is achieved in a DAI application when Agents autonomously invoke other Agents. Concurrency is realized when multiple Agents are invoked simultaneously (across multiple Hosts).

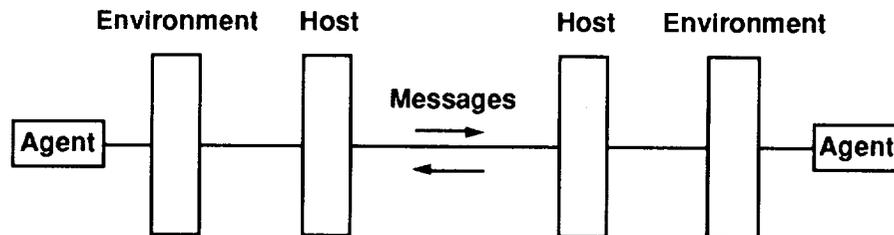


Figure 4: Operational Model of MetaCourier Communication Process

The openness of MetaCourier's communications architecture distinguishes SOCIAL from other DAI development frameworks, such as ABE, AGORA, MACE, ERASMUS [Ja88], and AF [Gr87].

Status: MetaCourier can be used as a standalone development language. It is currently available for: ANSI C and Common LISP programming languages; MS-DOS, UNIX, VMS, Macintosh Multifinder, and Lisp Machine operating systems; PCs, Macintoshes, Lisp Machines, VAX, Sun, and HP workstations. It currently utilizes TCP/IP on Ethernet and Appletalk protocols, but is extensible to other OSI-compatible network protocol suites.

MetaAgents: A Language for Agent Control and Coordination

The MetaCourier language offers a high-level interface that conceals the complexity of interprocess communication in distributed heterogeneous computing environments. Additional development capabilities are needed for internal process control, peer to peer inter-process coordination, and other distributed control services. MetaAgents is an object-oriented language for defining control Models to address these requirements.

The basic kernel MetaAgents Model protocols provide the equivalent of a traditional operating system's executive process control operations: agent creation, duplication, migration, and deletion. These protocols provide development-level options for specifying how to control Type inheritance behavior across distributed environments. The creation and copy protocols are critical because they allow new Agents to be defined dynamically at runtime.

MetaAgents Models support high-level message and concurrency management services. MetaCourier makes minimal assumptions about the ordering behavior of the low-level network protocols for message delivery, providing protocols to enforce simple message ordering schemes such as First In First Out (FIFO) delivery at particular nodes. MetaAgents Models define polices that use such guaranteed orderings to satisfy synchronization requirements of particular DAI applications [Pe89].

For example, MetaAgents supports an "atomic" broadcast protocol, which guarantees a globally invariant ordering of message delivery across all networks nodes. Atomic broadcast requires multiple phases of message exchanges; it should therefore only be used selectively, in situations where partial orderings of many-to-many agent interactions are insufficient and where lower performance can be tolerated. Atomic broadcasts are useful for maintaining consistency in transaction-oriented applications, such as where multiple agents send messages that operate on distributed replicated data.

MetaAgents defines other complex communication Models [Bi89] using a Group-based conversation abstraction: protocols are defined for agents to join a Group, to converse with other Group members via directed messages or broadcasts, and to depart the Group and the conversation. For example, a reliable Group broadcast protocol propagates information from one agent to other Group members such that all operational agents receive this information despite failures in the system.

Groups and broadcasts are very useful for replicating data for concurrency and fault management. Bottlenecks caused by centralized control can be alleviated by distributing task elements among agents that operate concurrently on replicated data and control state information. Similarly, data replicated along time-critical control paths can help to compensate for communication delay latencies in distributed networks (due to packet loss and node load variances) that lead to violations of real-time processing constraints. Replicated data can also be used to maintain redundant copies of critical state information to facilitate recovery control strategies for fault tolerant behavior in distributed systems. Group protocols also ensure orderly reintegration of agents into DAI applications when dropped network links are recovered.

The following sections describe basic SOCIAL agent Types to illustrate the roles of MetaAgents Models.

Receptionists and Gateways

The Receptionist is the root or kernel MetaAgents Type for single agents. It specifies basic communication services through MetaCourier or more complex MetaAgents protocols and Group protocols. A Receptionist agent is responsible for serializing concurrent requests, for scheduling access to its embedded application, and for detecting and recovering from possible error states that the application might enter. Receptionists manage the control transactions that implement fault tolerant behavior; agents departing from a Group due to failures of nodes or network links and agents rejoining a task processing conversation following network recovery. Receptionists can also be designed to manage security functions, for restricting access to specific application elements.

Databases and application programs are often constructed using commercial development tools such as DBMSs and AI shells. SOCIAL simplifies the design of Receptionists in such cases by abstracting the application-independent aspects of tools' control and data interfaces into specialized, predefined Receptionist subTypes called Gateways. Gateway agents supply predefined interface protocols for formulating queries or commands, concealing variations of syntax across comparable tools. Accessing a resource or program through a Gateway reduces to defining the application-specific aspects of the interface, in particular, formulating queries or commands whose arguments reference particular objects or attributes. Gateways for AI development shells must provide bidirectional interfaces for control as well as for data, so that intelligent applications can initiate queries or commands to other agents in the context of their own environments.

Manager

The distributed services provided by Receptionists enable application agents to interact through a "loosely-coupled" model of cooperation. More sophisticated control is often needed to coordinate a set of agents working together on one or several closely related tasks. The MetaAgents Manager and associated subTypes provide the requisite organizational control functionality.

A Manager regulates all communication between the agents within an organization via directed and broadcast protocols, providing a shared memory and a locus for centralized oversight and control. The Manager agent also mediates communication between external agents and organizational members, such as requests for data or services. To accomplish these various routing functions, the Manager maintains a "database" describing member agents and their relationships. Managers can be replicated to avoid processing bottlenecks and single point failures, although this entails additional control and performance overhead.

Specialized Manager subTypes will realize specific tightly-coupled distributed control frameworks, such as blackboard architectures [Ni86,Ja89]. The Manager Type does not restrict membership based on agent Type. This means that organizations can be arbitrarily complex. In particular, SOCIAL supports hierarchical organizations, in which a Manager coordinates other Managers. Thus, SOCIAL's library of organization Types can incorporate or subsume popular single problem DAI architectures, as well as hierarchical (multiple problem) frameworks such as OPERA. More important, SOCIAL permits different elements of a complex DAI system to be implemented using *different* agent and agent organization Types. MetaCourier provides the substrate or "backplane" of distributed communication services that enables high-level integration and coordination. Developers can exploit SOCIAL's support of heterogeneity to implement application elements using the most appropriate strategies for control and cooperation.

Status: The MetaAgents language design specification has been finished. Kernel process control protocols have been implemented. Initial control Models, Gateways, and agent organization Types will be completed by mid-1990.

MetaViews: A Language for Accessing Heterogeneous Data Resources

SOCIAL's Gateway agent Type facilitates non-intrusive integration of databases and knowledge-based systems implemented using standard, commercial DBMSs and AI shells. Gateway interfaces and services for distributed communication and process control derive from MetaCourier and MetaAgents Models. Additional services are required for formulating and processing queries and commands. MetaViews will address this problem through interface Models that are specific to particular DBMS or AI shells. These Models will be comprised of two elements: high-level interface protocols and services for translating between the protocols and the tool language in question. The protocols represent SOCIAL's equivalent to a programming interface library.

Figure 5 depicts the functions performed by a MetaViews interface Model. The block on the left represents an application agent A embedded in a Receptionist; the right hand box represents a database or knowledge-based system B embedded in a suitable Gateway. A issues commands for controlling or accessing B in terms of the functional protocols. A's Receptionist translator services convert those commands into an efficient canonical data representation, which are dispatched via MetaCourier. B's Gateway translator services convert canonical commands into the tool-specific language using the (invertible) protocol library. A's Receptionist and B's Gateway use MetaAgents services to manage concurrent messages.

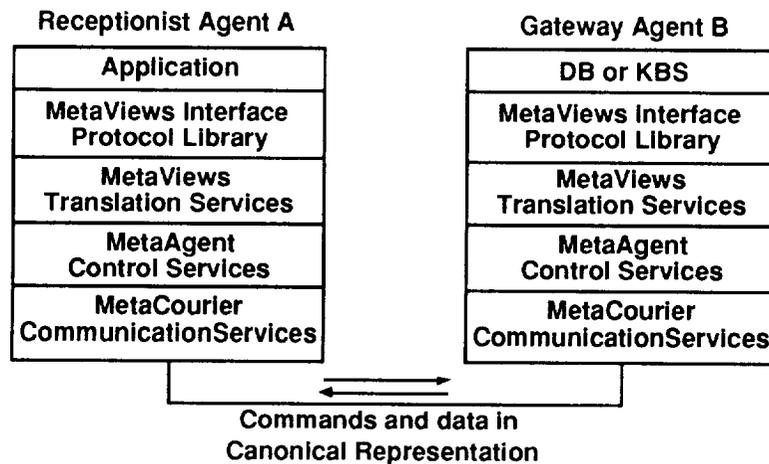


Figure 5: Operational Model of MetaCourier Communication Process

MetaViews technology is extensible to integrate other kinds of information system tools, such spreadsheets, computer-aided design tools, data analysis libraries, and data acquisition software.

Status: The MetaViews language design specification has been completed. Initial versions of MetaViews Receptionist and Gateway Models for ANSI C and Common Lisp for Oracle and Sybase relational databases. KEE and CLIPS AI shells will be complete by mid-1990.

Conclusions

Operations support of complex systems exemplifies "multiple problem" Distributed Artificial Intelligence (DAI) domains. These domains are distinguished by their heterogeneity. Domain problems vary in difficulty and degrees of interdependence. Application software and data resources can differ substantially with respect to structure, complexity, intelligence, and interfaces. Software and hardware platforms are also typically heterogeneous. The central design concerns in such domains are: (a) to integrate these diverse elements non-intrusively; and (b) to supply flexible coordination models to allow intelligent applications to interact cooperatively as a coherent, unified system.

SOCIAL is a generalized tool for developing DAI systems. It simplifies design and maintenance by enforcing a clear separation between application-specific functionality and distributed services. Application elements access services through high-level interfaces to predefined agent and agent organization Types. SOCIAL's interfaces reduce complexity by concealing the mechanics of distributed communication and control

across heterogeneous computing environments. "Standalone" applications, both intelligent and conventional, and data resources can thus be integrated non-intrusively. Moreover, SOCIAL allows intelligent applications based on different internal control schemes to be integrated within a single complex DAI system.

SOCIAL partitions distributed services into distinct object-oriented Models for: distributed communication (the substrate for all higher-level services); control services for managing processes and concurrency, and for coordinating agents on particular "single problem" DAI applications; and data translation. The SOCIAL architecture is open and extensible, with separate development interfaces to the library of generic agent Types and to the language-based Models that comprise them. These high-level tools free developers to concentrate on essential DAI architectural issues, such as designing strategies for coordinating intelligent subsystems.

Acknowledgments

The development of MetaCourier has been sponsored by the Department of Defense, U.S. Army Signal Warfare Center, under Contract No. DAAB10-87-C-0053. The development of SOCIAL has been sponsored by NASA under contract No. NAS10-11606.

References

- [Ad89a] R. Adler, B. H. Cottman. "A Development Framework for Distributed Artificial Intelligence." *Proceedings Fifth Conference on AI Applications, Computer Society of the IEEE, Miami, FL, March 6-10, 1989.*
- [Ad89b] R. Adler, A. Heard, and R. B. Hosken. "OPERA - An Expert Operations Analyst for A Distributed Computer Network." *Proceedings Annual AI Systems in Government Conference, Computer Society of the IEEE, Washington, D.C., March 27-31, 1989.*
- [Ad89c] R. Adler. "A Distributed Blackboard Architecture for Integrating Loosely-Coupled Knowledge-Based Systems." *Intelligent Systems Review*. Association for Intelligent Systems Technology, E. Syracuse, NY, 1989.
- [Ba88] S. E. Bayer, R. A. Harris, L. W. Morgan, J. F. Spitzer. *A Review of Space Station Freedom Program Capabilities for the Development and Application of Advanced Automation*. The MITRE Corp. Technical Report MTR-88D00059, McLean, VA, December, 1989.
- [Bi89] K. Birman et. al. *The ISIS System Manual V1.2*. Department of Computer Science, Cornell University, Ithaca, NY, June 1989.
- [Bo88] A.H. Bond and L. Gasser, eds. *Readings in Distributed Artificial Intelligence*. Morgan-Kaufmann, San Mateo, CA, 1988.
- [Br89] M. R. Barry. "PX1: A Space Shuttle Mission Operations Knowledge-Based System Project." *Proceedings Annual AI Systems in Government Conference, Computer Society of the IEEE, Washington, D.C., March 27-31, 1989.*
- [Bs87] R. Bisiani, F. Alleva, F. Correrini, A. Forin, F. Lecouat, and R. Lerner *Heterogeneous Parallel Processing: The Agora Shared Memory*. Carnegie-Mellon University, Computer Science Department. CMU-CS-87-112. March 1987.
- [Er80] L.D. Erman, F. Hayes-Roth, V.R. Lesser, and D.R. Reddy "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty." *ACM Computing Survey*, 12, pp. 213-253, 1980.
- [Ga86] L. Gasser, C. Braganza, and N. Herman. *MACE: A Flexible Testbed for Distributed AI Research*. Distributed Artificial Intelligence Group, Computer Sci. Dept. USC, 9-Aug-1986.
- [Gr87] P.E. Green. "AF: A Framework for Real-Time Distributed Cooperative Problem Solving." in [Hu87].
- [Ha86] B. Hayes-Roth. "A Blackboard Architecture for Control." *Artificial Intelligence*, Vol.262, pp.251-321, Mar, 1986.
- [Ha88] F. Hayes-Roth, L. D. Erman, S. Fouse, J. S. Lark J. Davidson. "ABE: A Cooperative Operating System and Development Environment." in [Bo88], pp. 457-489.

- [He87] A.E. Heard. "The Launch Processing System with a Future Look to OPERA." *Acta Astronautica*, IAF-87-215.
- [Hu87] M. N. Huhns, editor. *Distributed Artificial Intelligence*. Morgan-Kaufmann, Los Altos, California, 1987.
- [Ja88] V. Jagannathan, R.T. Dodhiawala, and L.S. Baum "Boeing Blackboard System: The Erasmus Version." *International Journal of Intelligent Systems*, Vol 3 Number 3, Fall 1988, pp. 281-294.
- [Ja89] V. Jagannathan, R.T. Dodhiawala, and L.S. Baum, eds. *Blackboard Architectures and Applications*. Academic Press, San Diego, CA, 1989.
- [Le83] V. R. Lesser and D. D. Corkill. "The Distributed Vehicle Monitoring Testbed: A Tool for Investigating Distributed Problem Solving Networks." *AI Magazine*, Fall 1983 pp. 15-33.
- [Mu89] J. F. Muratore, T. A. Heindel, T. B. Murphy, A. N. Rasmussen, R. Z. McFarland. "Applications of Artificial Intelligence to Space Shuttle Mission Control." *Proceedings Conference on Innovative Applications of AI, Stanford, CA, March, 1989*, pp 15-22.
- [Ni86] H.P. Nii "Blackboard Systems: The Blackboard Model of Problem-Solving and the Evolution of Blackboard Architectures." *AI Magazine*, pp. 38-53, Summer 1986.
- [Nu88] R.O. Nugent and R. W. Tucker. "An Architecture for Integrating Distributed and Cooperating Knowledge-Based Air Force Decision Aids." *Second Annual Space Operations Automation and Robotics Workshop (SOAR 88), Dayton, Oh, July 1988*.
- [Pa88] R. C. Paslay. *metaCourier: A Language For Distributed Heterogeneous Communication*. Symbiotics Inc. Cambridge, MA March 1988.
- [Pe89] L.L. Peterson, N.C. Bucholz, and R.D. Schlichting. "Preserving and Using Context Information In Interprocess Communication." *ACM Transactions on Computer Systems*, 7,3, August 1989.
- [Ru88] K. S. Rubin, P. M. Jones, C. M. Mitchell, T. C. Goldstein. "A Smalltalk Implementation of an Intelligent Operator's Assistant." *Proceedings Object-Oriented Programming Systems, Languages, and Applications, September, 1988*, pp 234-247.

A KNOWLEDGE BASE ARCHITECTURE FOR DISTRIBUTED KNOWLEDGE AGENTS

Joel Riedesel
MS: S-0550
Martin Marietta Astronautics
P.O. Box 179
Denver, Co. 80201
jriedesel@den.mmc.com

Bryan Walls
NASA, Marshall Space Flight Center
Bldg. 4487 EB12
Huntsville, AL 35812
bwalls@nasamail.arc.nasa.gov

ABSTRACT

In this paper a tuple space based object oriented model for knowledge base representation and interpretation is presented. An architecture for managing distributed knowledge agents is then implemented within the model.

The general model is based upon a database implementation of a tuple space. Objects are then defined as an additional layer upon the data base. The tuple space may or may not be distributed depending upon the database implementation. A language for representing knowledge and inference strategy is defined whose implementation takes advantage of the tuple space. The general model may then be instantiated in many different forms, each of which may be a distinct knowledge agent. Knowledge agents may communicate using tuple space mechanisms as in the LINDA model as well as using more well known message passing mechanisms.

An implementation of the model is presented describing strategies used to keep inference tractable without giving up expressivity. An example applied to a power management and distribution network for Space Station Freedom is given.

1. Introduction

In this paper a tuple space based object oriented model for knowledge base representation and interpretation is presented. The model provides a general knowledge language that is at once expressive and extendable. This allows it to be applied to many different domains including knowledge base management systems for expert system shells and architectures for distributed knowledge agents.

The field of Distributed Artificial Intelligence (DAI) is very complex. Besides the problems involving representing any particular agent, there is a whole new set of problems that are concerned with how multiple agents communicate with one another. This problem is more than just defining a mechanism but also involves protocols. How does one model of communication enhance the ease of solving one problem over another model?

The model presented in this paper supports DAI at a low level. This model presents a framework for defining multiple knowledge agents that must coordinate and cooperate with one another to solve some problem. This is different than most papers about DAI in that most papers are concerned with problems of communication and cooperation protocols. This model defines an architecture that supports the definition and implementation of diverse knowledge agents and their necessary protocols as the problem requires.

1.1. Three Requirements for Representing Distributed Knowledge Agents

There are a number of requirements a model will need to represent distributed knowledge agents adequately. Four interrelated requirements are identified and discussed here: Domain independence and expressivity, control knowledge, and communication.

1.1.1. Domain Independence and Expressivity

Domain independence and expressivity of a model are very closely related. Domain independence is concerned with the ability of the model to represent problems from any domain. The particular representation and storage of knowledge is important to domain independence. However, more than simply being concerned with expressing a problem in the model, domain independence is concerned with the ability of the model to integrate with the various problem domain environments. Expressiveness is concerned particularly with the ability and ease of stating a problem in the model language and not how the problem might have to deal with the environment of the problem.

The requirement for domain independence is concerned with the ability to represent a problem in the model and integrate that problem solution into the problem environment. This really implies that the language of the model must be extensible. It must be possible to enlarge the language using the base language as a start. This includes the ability to change inference mechanisms and define new models of inference. While most languages are concerned with representing data, this language is also concerned with representing control knowledge.

To support this requirement the language of the model presented here is a rule language built using object-oriented programming and extendable using the objects of the language. Rules are a primitive object and are evaluated and interpreted based upon a generic view of data stored in an independent database. A basic rule group object is provided with a forward chaining inference mechanism. A rule group may also be used to control the execution of another rule group thus enabling the definition of new inference mechanisms using the language. Additionally, more specific rule groups may be defined as sub-classes of the base rule group to support different inference mechanisms defined with meta-rule groups.

The other hand of domain independence is expressivity. There are at least two aspects to expressiveness: domain knowledge representation and control knowledge representation.

The problem of expressivity is that as more expressivity is allowed, along with more domain independence, the more intractable a language may become. The basic inference model for knowledge based systems consists of a match-select-fire cycle. The match phase determines those rules which are enabled and may be fired. The select phase selects one rule from the matched rules and the fire phase fires, or interprets, the selected rule. This basic paradigm captures the model of inference that knowledge base systems perform. To make this efficient, there are a number of options to the knowledge base designer. The language may be restricted, for example providing only universal quantification. Various compilation mechanisms may also be incorporated to make the match phase as static as possible. Both of these mechanisms are performed in OPS5 using the RETE network ([7, 8]).

The model presented here provides a large amount of expressivity while moving from the basic inference model presented above to something more tractable such as OPS5 and the RETE network.

To provide a large amount of expressiveness, the language of the model provides a frame system for representing structured knowledge as well as simple facts. Rules may use data from frame knowledge and fact knowledge. In [11], Hayes-Roth presents a number of knowledge categories that are needed for benchmarking different knowledge base systems. The categories fact, rule, class, entity, relation, and structure are provided by the model presented here. The remaining categories are not provided for but are planned as future work.

1.1.2. Control Knowledge Representation

Another requirement is the representation of control knowledge. The interpretation of the knowledge of different knowledge agents will need to be based upon the needs of the different knowledge agents. Some may require forward chaining while some may require backward chaining. Some may require more exotic strategies such as forward chaining with beam search.

There are two aspects of control knowledge identified here. One is the inference strategy used over a rule group and may be provided by defining meta-rule groups or LISP code. The other way of controlling rules is by providing a level of determinism into the ordering of rules themselves ([1, 9]). The way this is done in most expert system shells is by adding variable references into rules that provide control (e.g. IF [step = 1] ... THEN ...). This makes maintenance of the rules very difficult. An alternative is to use a transition table, allowing any level of non-determinism. Each entry in the transition table is used to index into the next possible rules. This is equivalent to defining a regular expression over a rule group ([9]).

The mechanism provided here is a transition table. As a rule is fired, that rule (its name) is used to index into the transition table to determine what rules are allowed to be fired on the next cycle. If the particular inference strategy being used allows more than one rule to be fired in a particular cycle, each of the fired rules is used to index into the transition table and the resulting lookups are unioned together. Complete non-determinism may be provided by not using a transition table and complete determinism may be provided by specifying only one rule as the next rule for any particular rule fire. Any mix of determinism and non-determinism may be specified using this mechanism. This is discussed further in section 2.

1.1.3. Robust Communication

The last requirement is for mechanisms for communication and coordination between knowledge agents. There is definitely not a consensus on the best mechanism for communication in the literature. The basic mechanism for communication and coordination provided here is a tuple space model based on LINDA ([5, 14]).

The tuple space model allows the addition of knowledge agents without having to modify existing knowledge agents about the communication interactions of the new agents. Communication is performed by inserting and removing tuples from the tuple space. If a knowledge agent is defined to take action based upon the existence of a tuple, the data-driven nature of the database will notify the knowledge agent of the ability of the rule to fire.

In addition to the tuple space model, it is entirely possible to perform message passing and other forms of communication by defining new functions that implement message passing to the

knowledge language. Thus the tuple space, although probably the primary mechanism for communication in this model, need not be at all restrictive to robust communications.

1.2. The Model Overview

Figure 1 shows the general architecture for supporting distributed knowledge agents. The database, to support the common tuple space, is the only module common to the agents. Even then, a common database is not a requirement if alternative forms of message passing are chosen. The database interface and Knowledge Base Management System (KBMS) are instantiated once on each physical platform.

In the figure a database is shown that exists independently of the KBMS. How the database is implemented is not important to the operation of the KBMS. The database interface is concerned with interfacing knowledge agents to the database. If a tuple is asserted to the database by a knowledge agent, the database interface will both add it to the database as well as notify any other knowledge agents that use that tuple of it. Rule Group 1 represents one knowledge agent and Rule Group 2 represents another. Both of these knowledge agents happen to be defined in a single instantiation of the KBMS system. Rule Group 3 and 4 are sub-rule groups and are part of the knowledge agent consisting of Rule Group 1. Another way to think of it is that each rule group represents a distinct knowledge agent where Rule Group 3 and 4 are strictly controlled by the agent of Rule Group 1. If another knowledge agent existed on another physical platform, there would be an additional instantiation of a database interface and KBMS. The database itself would remain (whether distributed or not) the same.

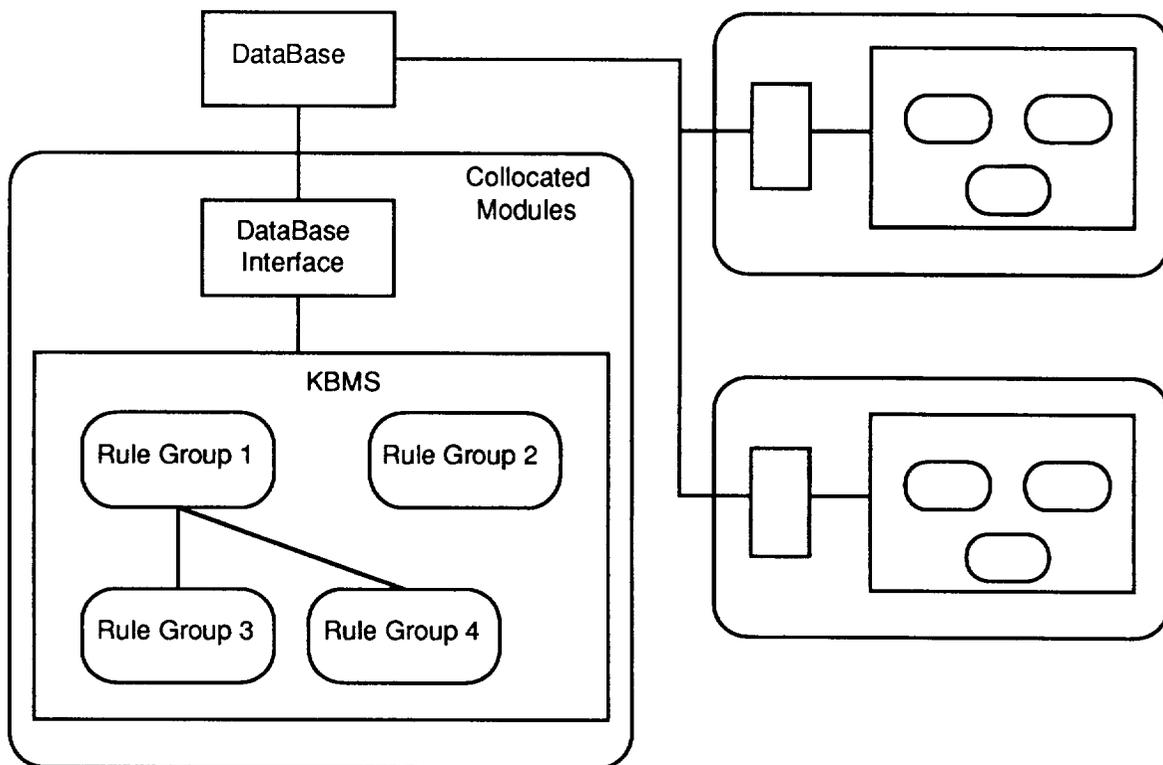


Figure 1 – KBMS Architecture

2. The Model: Its Design and Implementation

The model described here is based on a shared database, used to communicate between various knowledge agents through the passing of tuple data structures. Frames are defined on this base through a database interface to the knowledge agents, allowing data abstraction, inheritance, and structuring. Each knowledge agent contains domain knowledge in the form of rules and data and is controlled by the KBMS. The knowledge language of the KBMS provides the support to implement knowledge agents and allows arbitrary function calls. Tractability is maintained by a combination of rule compilation provided by the system and heuristic control knowledge provided by the knowledge engineer. The following subsections describe the design of the KBMS in much more detail.

2.1. The Base: The Database and Tuple Space

The first level of support for a KBMS is working memory, or the database. The architecture presented here views the database as a plug compatible module. This provides the ability to take advantage of existing databases and database management mechanisms such as ORACLE or INGRES for example.

The KBMS takes a tuple space view of the world and requires that the database represent them. A tuple is an ordered sequence of (possibly) typed fields. Conceptually, a tuple may be thought of as an object that exists independent of the process that created it. This implies that a place to store it is needed if it is to live a life independent of processes used to create and destroy it. The tuple space is sufficient for representing First Order Logic (FOL) and also sufficient for representing at least relational databases ([17]).

The tuple space has also been proposed as a mechanism for supporting distributed processing (the LINDA model, see [5]). There are three basic mechanisms that a process may perform on tuples in the tuple space: IN, OUT, and READ. The tuple fields in the IN and READ operations may optionally contain variable arguments for matching. IN and READ block until the tuple is present in the database. IN subsequently removes the tuple from the database while READ does not. OUT installs a tuple in the database. The benefit to distributed processing that the tuple space provides is the ability to add processes to the environment and have them communicate with existing processes without first having to encode knowledge about existing processes directly into their formalism. It has also been proposed that these basic tuple space operations are sufficient for supporting other communication mechanisms such as those defined in contract nets and Actors. On the other hand, the use of IN must be judicious in order to support the addition of other processes that may also need to use the tuple. See [5, 14] for more details.

The database provides three operations similar to the tuple space operations, these are: STORE, RETRIEVE/MATCH, and DB-REMOVE. The major difference between these operators and the tuple space operators IN, READ, and OUT, is that these do not block. Another difference is that while IN and READ will non-deterministically select one of multiple matching tuples, MATCH will return all of them (RETRIEVE will also act non-deterministically). Although the database operations defined here are not identical to the tuple space operations in the LINDA model (they do not block), the operations, in combination with the data-driven nature of a KBMS are probably sufficient for managing distributed processing protocols as LINDA does ([5]).

This view of the database allows for many different implementations of the database to be used, including distributed databases. A distributed database would provide the ability to support distributed knowledge agents defined using the knowledge representation presented here. Their

communication is then supported by the tuple space mechanism as implemented in the distributed database.

The implementation of the database for the KBMS described here also supports a form of integrity constraints and a restricted version of views. An integrity constraint is defined by a tuple. Fields of the tuple may contain one of four items: An actual field descriptor (formal argument); a variable (to match an argument); a type specifier, meaning that the field may match tuples with the corresponding field of the same type; and finally, a type declarator, meaning that for all tuples matched on the other three options, this field must be of the declared type. Views provide different databases for storage and retrieval. This is quite different to the traditional use of views in most databases. What this provides is a way to organize data into logical groupings. Currently, the use of integrity constraints is global across all views, in the future this should use the view mechanism just as any other storage or retrieval operation does.

2.2. The Next Level: Data Abstraction and Inheritance

The database interface is the module that links the database to the KBMS and vice versa. The interface provides the necessary hooks for proper accessing and notification of data in the database and of interest to the KBMS. The database interface provides the operations `STORE!`, `REMOVE!`, `MATCH!`, and `RETRIEVE!`. It also provides a data and procedure abstraction mechanism—a frame system ([10, 15]). Frames are used to extend the knowledge representation language of the KBMS for supporting complex domains requiring novel abstractions and inheritance.

Frames are an abstract organization of data into conceptual units. In this definition data may be any object, it may be simple data or even procedural objects. A frame may have any number of slots. Frames may be defined as children of multiple parents (making inheritance potentially more complex) and may also have code attached to them that is executed whenever a new instance of the frame or one of its children is created. Slots may have six optional aspects. The most used aspect is the `:value` aspect. This aspect is where a value for the slot is located. The `:if-needed` aspect is used to store code that is executed if a slot value is asked for. The `:if-added` aspect is used to store code that is executed whenever the slot gets a new value. There are two aspects that are used to constrain the value of the slot. The `:constraint` aspect is used to store code that checks if an added value passes the constraint. Since this is user-defined code there is no restriction on what it may do, only that it return a true or false status indicating the result of the constraint. The `:mustbe` aspect constrains the value to be one of a list of formal values or frames. Finally, the `:distribution` aspect is used to determine if the value of the slot is for global distribution, accessible to all knowledge agents, or only for local use.

Frame data is stored in the database as 5-tuples of the form: `(frame <name> <slot> <aspect> <value>)`. Obviously, there is no restriction on the value aspects, they can be any normal data type as well as executable code. Facts are stored in the database as 4-tuples: `(fact <name> :value <value>)`.

Frame inheritance information must also be stored in the database so that inheritance over frames may take place in response to different knowledge agent's requests for data values. The current implementation does not yet store this data in the database.

It is the responsibility of the database interface to both store and retrieve information and to notify the KBMS of changes to data made by other knowledge agents. Thus, if knowledge agent 1 makes a change to the value of a fact and knowledge agent 2 uses that fact on the left hand side (LHS) of a rule, it is the responsibility of the database interface (and distributed database) to notify knowledge agent 2 of the changed fact. And vice versa, it is the responsibility of the system, when

knowledge agents are being defined to notify the database interface of which knowledge agents index on which facts.

2.3. The Knowledge Base Management System

Now that the basic knowledge storing and retrieving mechanisms have been outlined, the heart of the system needs to be defined.

The knowledge representation language is defined as a set of objects up to the level of rules. A KBMS is then instantiated in the language using a combination of user-defined objects, user-defined code, rules and KBMS domain knowledge (i.e. knowledge about what the KBMS is). The KBMS instantiated here consists of the user-defined objects: `rule-group` and `knowledge-base`. The default inference strategy is implemented in code, but can also be implemented in the form of meta-rule groups. Rules for defining control strategy, for instance, are defined. The domain knowledge consists of frame knowledge about rule groups, knowledge bases, and knowledge of executable procedures.

To go one level further, an expert system is then instantiated from the KBMS. Here, rules and rule groups are provided for the domain; domain knowledge is provided, as well as more specific inference strategies for the expert system. To use an analogy from the `flavors` object oriented system, the knowledge representation language is like a base flavor. It is necessary to define a mixin to it in order to give it functionality. Finally, the flavor can be instantiated into a user-definable object (e.g an expert system for fault diagnosis).

The knowledge representation language provides functionality to the level of rules. A rule has the basic form of `LHS ::> RHS`, and may include else conditions. Quantified rules quantify over some set, binding a variable to successive values and executing the sub-rule of the quantified rule. Rules have the basic operations: `evaluate-lhs`, `evaluate-rhs`, `interpret-lhs`, `interpret-rhs`, and `interpret-else`. These operations may return one of three values: `:ok`, `:ng`, and `:missing-patterns`. If the rule evaluates `:ok` is returned, if conditions are not met `:ng` is returned, and finally, if patterns needed to check conditions do not yet exist (i.e. values do not exist for referenced variables) `:missing-patterns` is returned. These return values are used by various control strategies that can be user-defined.

The current implementation of the knowledge representation language assumes the existence of a `rule-group` definition that must include the slots: `*lhs-tickled-queue*` and `*rhs-tickled-queue*`. These slots are used to queue up rules whose LHS and RHS (respectively) are potentially `:ok`. This allows a wide variety of control strategies to be defined including forward and backward chaining as well as various combinations thereof.

The KBMS implemented here defines a knowledge base to consist of a number of rule groups as well as domain knowledge. Each rule group provides mechanisms for defining aspects of the inference strategy using either further rule groups or user-defined functions. A rule group also has a mechanism for specifying the level of determinism desired over rule execution ([1, 9]). Basically, the rule group inference strategy consists of a match, evaluate, and fire loop. The match phase is supported by the database interface which automatically queues up potential rules on the tickled-queues. The evaluate phase then checks the rules on the tickled-queues to determine which ones belong in the conflict set. Finally, one rule is selected and fired. This approach is completely non-deterministic in determining which rules get fired from the conflict set. In between the match and evaluate phases a control phase is added. This control phase consists of the definition of a regular expression that defines which rules may be fired next. In this implementation the transition table derived from the regular expression is given instead of the regular expression itself. This is much simpler from the user's perspective. For the user defining a number of rules and wanting to

insert some control over them, it is easier to specify a transition table over the rules than it is to define the regular expression that the transition table can be derived from. Furthermore, it is easier to maintain a transition table than a regular expression during rule group modification.

The knowledge representation language, although not completely independent of the KBMS, defines a very general mechanism for specifying knowledge and inference. The KBMS defined here is also very general and allows for a wide variety of specification of inference strategies over the knowledge. This can be done using the knowledge language or by alternatively writing executable code directly (i.e. interpreted vs. compiled).

2.4. The Tractability of the KBMS

Considering the generality and expressivity of the knowledge language and the KBMS, efficiency has the potential of getting lost. There are three ways of maintaining tractability in the knowledge language.

The first way of maintaining tractability is to manage the match phase of the inference cycle efficiently. The conceptual definition of the match phase is to check each rule and evaluate its LHS (RHS) and if it is ok then to queue it up. Obviously, this is also the slowest approach. The step this implementation takes is to compile rules into a rule constraint network that maintains rules in a form more suited for recognizing when data becomes available that has the potential of satisfying the LHS (RHS) of a rule. Consider the LHS of a rule. During evaluation, all the referenced variables must have values in order to determine if the LHS is satisfied. The rule constraint network represents variables as nodes in the network and rules as rule-nodes (see Figure 2). As variables get values, the nodes representing the variables are triggered. All the rule-nodes connected to the node are then triggered. These rule-nodes are then checked to see if all the nodes representing the LHS (RHS) variables have a value. If so, the rule represented by the rule-node is then put on the appropriate queue. As can be seen, this method allows rules to be tickled that may not be satisfied. The rule constraint network only checks to see if variables have values, not if they have the right value. Thus the rules on the tickled queues must still be evaluated for satisfaction.

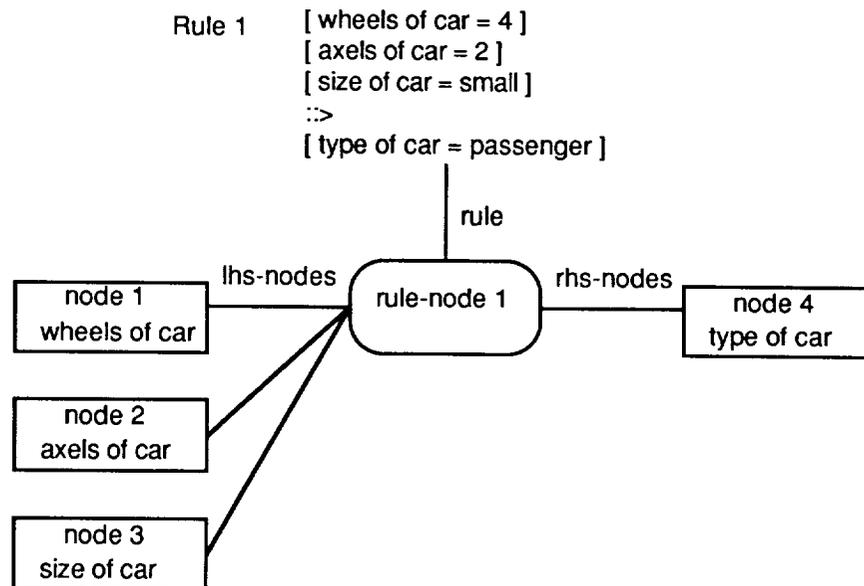


Figure 2 – Rule Constraint Network Example

Quantified rules are also compiled into this form as the variables being quantified over are given values. Thus, if a rule asks if there exists a symptom in the symptom-set and there are three symptoms in symptom-set, then three instances of this graph structure will be created, one of which may cause the rule to fire.

Alternatively, rules could be compiled more completely so that variables are checked to see if they have the right value. The complexity of the knowledge language implemented here makes this a difficult task and it has not been determined that it would be cost effective. Languages with less expressivity can be completely compiled much more easily (such as OPS5 using the RETE network [7, 8]). The expressivity and tractability trade-off turns up once again.

The second way of maintaining tractability is by providing code for inference strategies instead of providing rules defining the inference strategy. This is making use of the compiled vs. interpreted option. This means that a programmer fluent in the language that the knowledge language is implemented in needs to be available for both implementation and maintenance. However, considering the expressivity of the knowledge language, this may be a cost effective solution. It is very easy for the programmer to express what is desired without having to go into contortions over representational limits.

The third way to maintain tractability is by the effective use of knowledge. This is to make use of Raj Reddy's fifth principle: "Knowledge eliminates the need for search" ([16] see also [12]). In other words, the domain to be represented needs to be analyzed for maximum efficiency in terms of knowledge organization. For example, I can have two rule groups; one rule group forward chains on various data and computes a value for the variable *diagnosis*. The other rule group then uses the value of *diagnosis* to output the results to the user. If there are 50 rules in each rule group that use the variable *diagnosis*, then 100 rules are triggered whenever the value of the variable changes. If both of these rule groups are made to be sub-rule groups of a control rule group, the first rule group can compute a value for *diagnosis*. The control rule group can then set the value for the variable *the-diagnosis* as the value of *diagnosis*. *the-diagnosis* is then used by the second rule group instead of *diagnosis*. Now only 50 rules get triggered at one time.

The representation of knowledge should make maximum use of divide and conquer principles of knowledge organization. Another approach along the same lines is to provide strong heuristic knowledge to eliminate the need for search. This can come in many forms including domain guided inference strategies over rule groups as well as judicious use of control over the execution of rules in a rule group (i.e. the transition table method over rules).

3. A Power Management and Distribution Knowledge Agent

In this example a knowledge agent for managing power and distribution for Space Station Freedom is presented. The knowledge base consists of approximately 150 rules at this time. Naturally, space limitations prohibits the presentation of the entire knowledge base. This example should be sufficient to illustrate the representational capabilities of the knowledge language and how it can be applied to defining multiple agents for distributed processing tasks.

The first part of this example consists of the definitions required by the knowledge language to support the KBMS. The second part then describes the Power Management and Distribution Knowledge Agent.

3.1. Definitions for KBMS Support

These definitions define the user-accessible structure of a knowledge base and of a rule group. The lisp frame is for defining various functions that may be called from rules.

```
(frame :name knowledge-base
      :slots ((rule-groups :value nil)
              (agents :value nil)
              (name :value nil)))

(frame :name rule-group
      :slots ((rules :value nil)
              (name)
              (quantified-vars)
              (rg-var)
              (plan)
              (plan-state)
              (plan-table)
              (viable-set :value nil)
              (not-yet-viable-set :value nil)
              (fire-set :value nil)
              (local-variables)
              (conflict-set :value nil)
              (tickle-set :value nil)
              (satisfied-set :value nil)
              (unsatisfied-set :value nil)
              (cant-fire-set :value nil)
              (fired-set :value nil)
              (untickled-set :value nil)
              (*lhs-tickled-queue* :value nil)
              (*rhs-tickled-queue* :value nil)
              (termination-condition :value nil)
              (control-strategy :value
                               #'default-control-strategy)
              (conflict-resolution-strategy :value
                               #'default-conflict-resolution-strategy)
              (execute :value #'execute)))

(frame :name lisp
      :slots ((evaluate :value #'evaluate)
              (evaluate-lhs :value #'evaluate-lhs)
              (evaluate-rhs :value #'evaluate-rhs)
              (interpret-rhs :value #'interpret-rhs)
              (interpret-lhs :value #'interpret-lhs)
              (interpret-else :value #'interpret-else)
              (first :value #'first)
              (second :value #'second)
              (format :value #'format)
              (length :value #'length)))
```

3.2. Power Management and Distribution Knowledge Agent

The main knowledge base is defined here. It is called pmad and uses a domain file (domain.lisp) to define the various data structures (domain knowledge) relevant to the knowledge agent. The knowledge base consists of three rule groups; a control rule group for controlling the

diagnosis of hard faults, a hard fault rule group for computing a diagnosis, and a diagnosis rule group simply for printing a diagnosis.

KB : pmad

DOMAIN : domain.lisp

RULE-GROUP : control-rg

This control rule group currently manages the collection of fault information for diagnosis. It controls the execution of two rule groups: hard-fault and diagnosis, for performing and printing out diagnosis information respectively. Eventually rules will be added to this knowledge agent for soft fault and incipient fault analysis.

```
CONTROL : ((start (Control-Rule1))
           (Control-Rule1 (Control-Rule2))
           (Control-Rule2 (Control-Rule3 Control-Rule4
                           Control-Rule5))
           (Control-Rule3 (Control-Rule3 Control-Rule4
                           Control-Rule5))
           (Control-Rule4 (Control-Rule3 Control-Rule4
                           Control-Rule5))
           (Control-Rule5 (Control-Rule6))
           (Control-Rule6 (Control-Rule1)))
```

```
Control-Rule1
THERE EXISTS symptom-set in symptom-set-queue
< ::>
  [ the-symptom-set = symptom-set ]
  [ diagnosis-set = empty ] >
;
Control-Rule2
::>
[ clusters = power-domain :: cluster-symptoms
  ( the-symptom-set ) ]
;
Control-Rule3
THERE EXISTS cluster in clusters
< ::>
  [ symptoms of symptom-set1 = symptoms of cluster ]
  [ symptom-set1 = cluster ] >
ELSE
  [ ready-to-diagnose = false ]
;
Control-Rule4
THERE EXISTS symptom in symptoms of symptom-set1
< [ hard-fault :: execute ( hard-fault ) = :ok ]
  ::>
  [ diagnosis-set = diagnosis-set PLUS diagnosis ]
  [ the-diagnosis = diagnosis ]
  [ diagnosis = :unknown ]
  [ diagnosis-rg :: execute ( diagnosis-rg ) ]
  [ clusters = clusters MINUS symptom-set1 ] >
;
Control-Rule5
[ ready-to-diagnose = false ]
[ FOR ALL diagnosis1 in diagnosis-set
  < [ diagnosis1 = diagnosis-2 ]
```

```

        OR
        [ diagnosis1 = diagnosis-31 ] > ]
    ::>
    [ the-diagnosis = diagnosis-no-power ]
    [ diagnosis-set = diagnosis-set PLUS the-diagnosis ]
    [ ready-to-diagnose = true ]
    ELSE
    [ ready-to-diagnose = true ]
;
Control-Rule6
[ ready-to-diagnose = true ]
::>
[ diagnosis-rg :: execute ( diagnosis-rg ) ]
[ symptom-set-queue = symptom-set-queue
    MINUS the-symptom-set ]
[ symptom-set1 = :unknown ]
[ clusters = :unknown ]
[ ready-to-diagnose = :unknown ]
:

```

RULE-GROUP : hard-fault.rg

This rule group takes information about a fault and, after determining if any testing needs to be done and doing it, determines the fault and assigns diagnosis a value.

As can be seen, the control that is specifiable over a rule group may be quite complex. The whole transition table for this rule group is not given (although this is one-third of it, for a rule group consisting of about 90 rules), but the expressivity is still quite apparent.

```

CONTROL : ((start (init-rule))
           (init-rule (Rule1))
           (Rule1 (Rule2))
           (Rule2 (Rule3 Rule4.1 Rule5 Rule31 Rule31.1
                  Rule31.2 Rule35 Rule35.1))
           (Rule3 (Rule4))
           (Rule4 (Rule20 Rule32))
           (Rule32 (Rule33))
           (Rule33 (Rule34))
           (Rule4.1 (Rule4.2))
           (Rule4.2 (Rule4.3))
           (Rule4.3 (Rule4.4 Rule4.5 Rule4.6 Rule4.7
                  Rule4.8))
           ...
           )

```

```

Init-rule
::>
[ symptom-set = symptoms of symptom-set1 ]
[ possible-top-switches = empty ]
[ tripped-top-switches = empty ]
;
Rule1
::>
[ top-symptoms = power-domain ::
    top-symptoms ( symptom-set ) ]
;
Rule2
[ lisp :: length ( top-symptoms ) > 1 ]

```

```

::>
[ type = multiple-tops ]
ELSE
[ type = single-top ]
;
Rule3
[ type = multiple-tops ]
[ THERE EXISTS symptom in top-symptoms
  < [ lisp :: length
      ( switches-below of switch of symptom ) > 0 ] > ]
[ FOR ALL symptom in top-symptoms
  < [ fault of symptom = fast-trip ]
    OR
    [ fault of symptom = over-current ] > ]
::>
[ FOR ALL symptom in top-symptoms
  < [ tripped-top-switches = tripped-top-switches PLUS
      switch of symptom ] > ]
[ THERE EXISTS symptom in top-symptoms
  < [ possible-top-switches = switch of symptom PLUS
      siblings of switch of symptom ] > ]
[ THERE EXISTS symptom in top-symptoms
  < [ trip-type = fault of symptom ] > ]
[ type = multiple-top-current-trip ]
;
...
Rule4.4
[ type = multiple-tops ]
[ lisp :: length ( new-symptoms ) = 1 ]
[ THERE EXISTS symptom in new-symptoms
  < [ THERE EXISTS symptom1 in top-symptoms
      < [ switch of symptom = switch of symptom1 ]
        [ fault of symptom = fault of symptom1 ] > ] > ]
::>
[ diagnosis = diagnosis-54 ]
;
...
:
[ diagnosis ]

```

RULE-GROUP : diagnosis.rg

This rule group simply prints out some statements and communicates diagnostic information to the scheduling knowledge agent depending on the particular diagnosis encountered.

```

d-rule1 @@ backrush in load center
[ the-diagnosis = diagnosis-54 ]
::>
[ the-diagnosis = :unknown ]
[ lisp :: format ( t "~%The following load center
                    RPCs tripped on fast-trip~%" ) ]
...
;
...
:
[ the-diagnosis = :unknown ]

```

The rest of the knowledge base for this knowledge agent.

```
Domain-Knowledge :
  constants :
    t ; :ok ; :ng ; :missing-patterns ; yes ; no ;
    hard-fault ;
    diagnosis-rg ;
    multiple-tops ;
    single-top ;
    multiple-top-current-trip ;
    over-current ;
    under-voltage ;
    fast-trip ;
    ground-fault ;
    diagnosis-1 ;
    diagnosis-2 ;
    ...
    diagnosis-no-power .
  facts :
    empty = ( ) .
  frames :
    (fcreate-instance 'symptom-set 'symptom-set1) .

Begin : control-rg

END-KB

+++++
domain.lisp
+++++

(frame :name power-domain
      :slots ((top-symptoms :value #'top-symptoms)
              ...
              (close-switch :value #'close-switch)))

(frame :name symptom-set
      :slots ((symptoms)))

(frame :name symptom
      :slots ((switch) (fault)))

(frame :name switch
      :slots ((name)
              (type)
              (current)
              (switches-below :value nil)
              (switch-above :value nil)
              (siblings)
              ...
              (current-rating)
              (fast-trip-percent)))

...
```

Lots of domain knowledge here to build instances of switches and sensors, etc. Knowledge of the topology is encoded here. About 30k worth.

4. Conclusions and Future Work

An architecture for defining and modifying knowledge base management systems that may be used for applications in distributed AI has been presented. The architecture is very flexible and relatively efficient. It has been used to define three very different knowledge agents: One for solving a toy problem to compute how to send a package consisting of about ten rules; One for solving the monkeys and bananas problem consisting of about twenty fairly complex rules, this one was directly adapted from a solution given for OPS5; Finally the agent given in this paper and consisting of around 150 rules.

The results we have noticed so far have shown that this architecture provided the ability to easily implement a solution to a wide variety of problems. The monkeys and bananas problem has driven out many areas of weakness in the implementation that are being dealt with. The speed with which this architecture solves the monkeys and bananas problem is hardly even comparable to that of OPS5 at this point. However, the result of implementing our fault diagnosis problem for power management and distribution has turned out very well. Using simple forward chaining and lots of control knowledge in the hard fault rule group has enabled us to provide a solution that is very fast and easily maintainable. The maintainability is very important for this domain as the requirements for Space Station Freedom have not been completely specified.

4.1. Future Work

This system is being implemented as part of a much larger system, KNOMAD (Knowledge Management and Design System). We have identified a number of areas where knowledge needs to be added to support a completely robust, domain independent environment for specifying knowledge based systems. These include the addition of a constraint system, a temporal database, and analytical and qualitative reasoning. These additions will then support planning, scheduling, and causal reasoning at the least. Adding these components must involve how the KBMS will access and use these components as well as how these components will use the existing database and database interface mechanisms.

Work also needs to be pursued to determine the possibility of adding RETE-like structures as a part of the rule constraint network.

5. Acknowledgements

This work is being supported by NASA, Marshall Space Flight Center, contract NAS8-36433.

6. References

- [1] Baskin, A.B., "Combining Deterministic and Non-deterministic Rule Scheduling in an Expert System," AAMSI, 1986.
- [2] Bond, Alan H., and Les Gasser, Eds., "Readings in Distributed Artificial Intelligence," Morgan Kaufman, 1988.
- [3] Brodie, M.L., J. Mylopoulos, and J.W. Schmidt, (Eds.), "On Conceptual Modelling," Springer-Verlag, New York, 1984.
- [4] Buchanan, Bruce G., and Richard O. Duda, "Principles of Rule-Based Expert systems," Stanford Heuristic Programming Project Report No. HPP-82-14, 1982.

- [5] Carriero, Nicholas, and David Gelernter, "Linda in Context," *Communications of the ACM*, 32(4), 1989.
- [6] D'Angelo, Antonio, Giovanni Guida, Maurixo Pighin, and Carlo Tasso, "A Mechanism for Representing and Using Meta-Knowledge in Rule-Based Systems," *Approximate Reasoning in Expert Systems*, 1985.
- [7] Forgy, Charles L., "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," *AI* 19(1) 1982.
- [8] Forgy, Charles L., and Susan J. Shepard, "RETE: A Fast Match Algorithm," *AI Expert*, Jan. 1987.
- [9] Georgeff, M.P., "Procedural Control in Production Systems," *AI* 18, pp. 175-201, 1982.
- [10] Hayes, P.J., "The Logic of Frames," in Webber, B.L., and Nils J. Nilsson (Eds.) *Readings in Artificial Intelligence*, pp. 451-458, 1981.
- [11] Hayes-Roth, Frederick, "Towards Benchmarks for Knowledge Systems and Their Implications for Data Engineering," *IEEE Transactions on Knowledge And Data Engineering*, Vol. 1, No. 1, March 1989.
- [12] Lenat, Douglas B., and Edward A. Feigenbaum, "On the Thresholds of Knowledge," *International Workshop on Artificial Intelligence for Industrial Applications*, 1988.
- [13] Levesque, Hector J., "Knowledge Representation and Reasoning," *Annual Reviews of Computer Science*, 1986.
- [14] Matsuoka, Satoshi, and Satoru Kawai, "Using Tuple Space Communication in Distributed Object-Oriented Languages," *OOPSLA 1988 Proceedings*.
- [15] Minsky, M., "A Framework for Representing Knowledge," in P. Winston (Ed.) *The Psychology of Computer Vision*, McGraw-Hill, New York, pp. 211-277, 1975.
- [16] Reddy, Raj, Presidential Address at the American Association for Artificial Intelligence Conference, 1986.
- [17] Reiter, R., "Towards a Logical Reconstruction of Relational Database Theory," in [3], pp. 191-233, 1984.
- [18] Stefik, Mark, Jan Aikins, Robert Balzer, John Benoit, Lawrence Birnbaum, Frederick Hayes-Roth, and Earl Sacerdoti, "The Organization of Expert Systems, A Tutorial," *AI* 18, 1982.
- [19] Wilkins, David E., "Practical Planning: Extending the Classical AI Planning Paradigm," *Morgan Kaufman*, 1988.

**CREATURE CO-OP : Achieving Robust Remote Operations With A
Community of Low-Cost Robots**

R. Peter Bonasso (bonasso@ai.mitre.org)
Washington Artificial Intelligence Technical Center
The MITRE Corporation
7525 Colshire Drive, McLean, VA 22102

Abstract

This paper puts forth the concept of carrying out space-based remote missions using a cooperative of low-cost robot specialists rather than monolithic, multi-purpose systems. A simulation is described wherein a control architecture for such a system of specialists is being investigated. Early results show such co-ops to be robust in the face of unforeseen circumstances. Descriptions of the platforms and sensors modeled and the beacon and retriever creatures that make up the co-op are included.

INTRODUCTION

An alternative to building one robot for remote planetary operations, like the Mars Rover, is to build several robots whose combined capabilities can do the same task. A single Mars Rover will have extensive perception capabilities, robust navigation capabilities, intricate sample detection and acquisition capabilities, long-range fuel capability, and, in some designs, an orbit-achieving propulsion system. But damage to any major subsystem will halt the reconnaissance mission for months until another Rover can be launched and landed on Mars.

An alternative approach is to use a cooperative of robots of several sorts. One sort can carry fuel cells (fuel-bots), another sort has strong farsighted visual perception (beacon-bots), another sort has good mobility and dexterity and short-sighted vision (rover-bot), another sort can attach itself to objects and push and pull them (retriever-bot), and so forth. Each robot would have the basic capability to stay out of harm's way (e.g., avoiding sand pits, dodging meteors), and could communicate with the others via RF or some other wireless link.

Then, in the Mars Rover effort for example, the beacon-bots and fuel-bots could be placed in strategic positions throughout the reconnaissance area, serving as far-sighted eyes and gas-stations for the rovers which, near-sighted, negotiate rough terrain and acquire soil samples. The earth-bound human controller will always be in the loop, evaluating the evolving situation and sending high-level commands (but only every 40 minutes or so). Losing one of the beacon-bots or fuel-bots will only limit the overall system capability in a certain geographic area. Losing a rover-bot or retriever-bot will only burden another rover or retriever with more tasks to do of the same type. So this

community of robots could be ultimately more cost-effective than one large all-purpose, well-equipped robot.

This paper describes an ongoing investigation into robot cooperatives for accomplishing remote operations. It builds on research being carried out in situated reasoning, perception, and planning for autonomous agents. Specifically, this work relies on the ideas of reasoning via subsumption software architectures (3, 6) and reaction plans inspired by (13), and on the growing number of low-cost integrated robot platforms that are becoming available (e.g., (4), (5)). The idea of distributed robots is not new, dating at least to early-eighties work by Sacerdoti who coined the term disbots (12). But only in recent years has realizing such systems on low-cost integrated platforms been possible.

This paper does not address many difficult issues associated with distributed control. The assumption here is that a low-level of cooperation can be achieved much like a *hive gestalt*, and anything beyond that is handled by human intervention.

The next section details an object-oriented software environment on a micro-Explorer for conducting experiments in robot cooperation. These experiments are beginning to demonstrate that such cooperatives will be robust and relatively inexpensive. Through an example, the cooperating capabilities of the "creatures" (see (3)) are characterized and the implications for cost discussed. The last section describes follow-on work with actual robots in MITRE's Autonomous Systems Laboratory.

THE CREATURE CO-OP EXPERIMENTAL ENVIRONMENT

Figure 1 shows a top or plan view of the layout of the environment on a micro-Explorer computer screen to simulate the action of a cooperative of robots with varying levels of competence. The large circle is the work area within which all robot action takes place. The open geometric figures are candidate objects for retrieval. The filled-in circles are the creatures for the described experiments: the smallest ones are beacons with farsighted vision; the others are retrievers with circumferential sonars, and arms (initially retracted) that extend from the body to grasp or latch on to the object to be retrieved. The thin lines extending from each creature show their current orientation (they are all at 0 degrees; angles are measured counter-clockwise).

The right-hand pane shows a number of menu options available to the experimenter and the bottom pane is used for text output and user typein. The user can direct the creatures to retrieve objects, move to new locations, go to sleep and wake up.

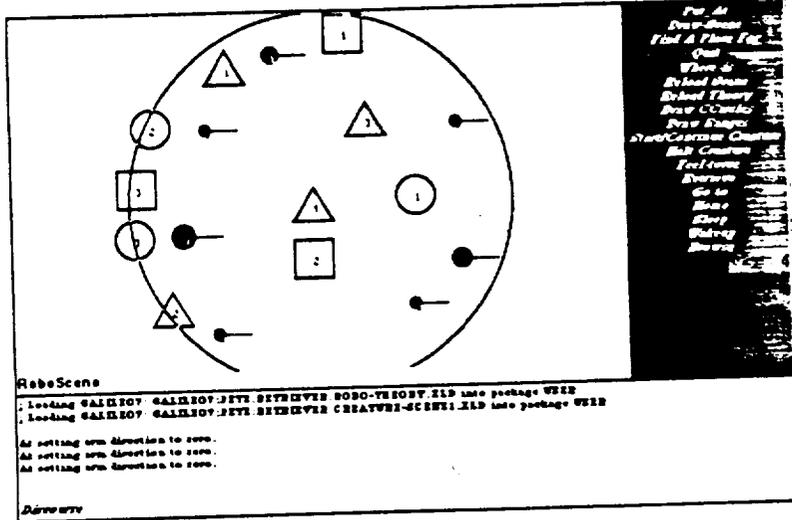


Figure 1

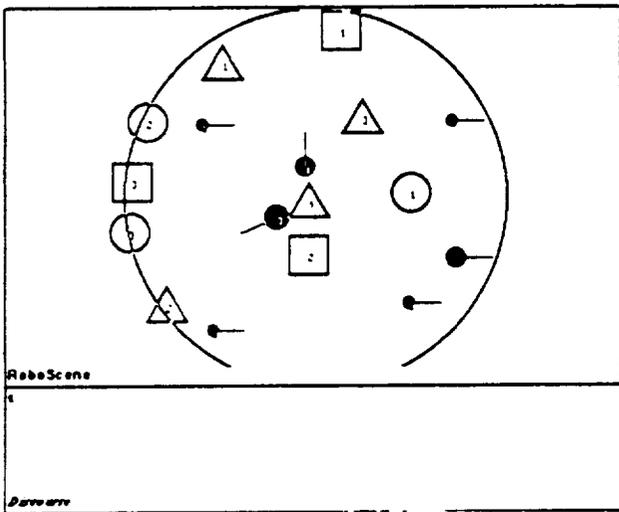


Figure 2

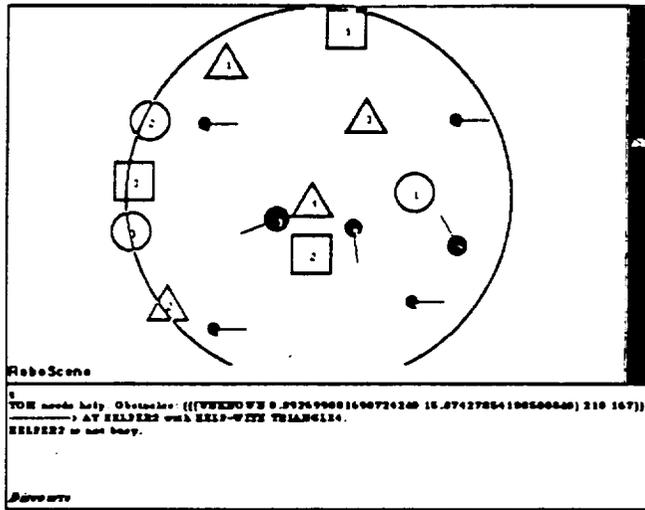


Figure 3

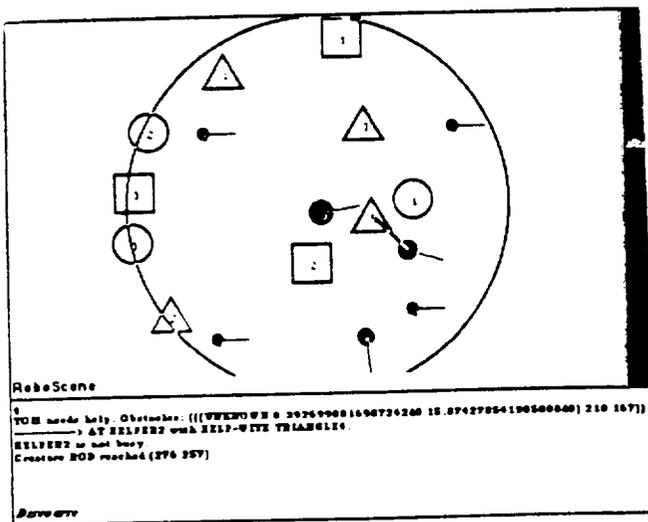


Figure 4

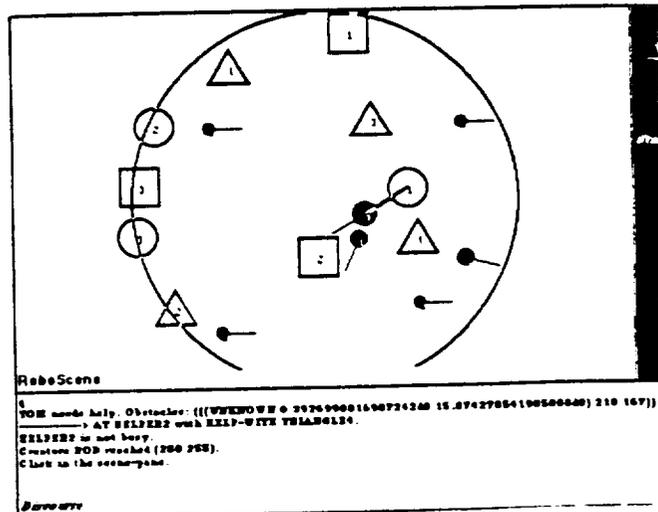


Figure 5

An Example Of Creature Cooperation

The following example demonstrates the utility that can be obtained with minimal competency "specialist" robots in a cooperative effort. In this example, the large retriever (#3) has the task to retrieve Cylinder1, and the small retriever at the top of the screen (#1) is to navigate to coordinates in the bottom part of the work area and return.

Selecting Start/Continue Creature generates a process for each creature that for each simulation clock tick causes the creature to continue to carry out a retrieval or a directed move and to avoid any obstacles. Without any retrieval or directed moves pending, no action would be detected. But if the user puts an object next to any creature, that creature would immediately move a safe distance away.

After being given a target-object, retriever #3, blind but for its short-range sonars, first sends a request via RF link for the location of Cylinder1. Only beacons have the requisite ranging and recognition capabilities. The beacon in the upper right of the picture locates Cylinder1 and transmits the coordinates to retriever #3, which begins to dead reckon to the object. Retriever #1 simply heads for the desired location at the bottom of the screen.

In Figure 2 both retrievers have encountered Triangle4. Both creatures have a low-level survival routine that usurps control from any higher level routine when an obstacle is sensed. The "angle of escape" and "speed of escape" computed by this routine are averaged with the "angle of purpose" and "speed of purpose" of any higher level routines to allow the retriever to angle back on track once the obstacle has been avoided. In Figure 3, retriever #1 is past the obstacles and is heading unobstructed for its destination.

Retriever #3, however, is blocked since in attempting to reach Cylinder1 it is ping-ponging between the Triangle4 and Block2. At this point, noticing that it is not making progress (see Getting Stuck below), retriever #3 broadcasts a help request indicating an obstacle near the coordinates of Triangle4. Any retriever that is not busy with another task can process the help request, and retriever #2 (to the right of Cylinder1) takes up the task, broadcasting a request for a beacon to identify the object near the coordinates of interest. Again, the beacon in the upper right locates and identifies Triangle4 for retriever #2.

When retriever #2 is close enough to the triangle, it extends its arm, latches onto the object, and begins the return trip to the location from which it received the help request, thus pulling the obstacle out of the way of retriever #3. Figure 4 shows retriever #2 returning and retriever #3 continuing toward Cylinder1. Retriever #1 has arrived at the required destination.

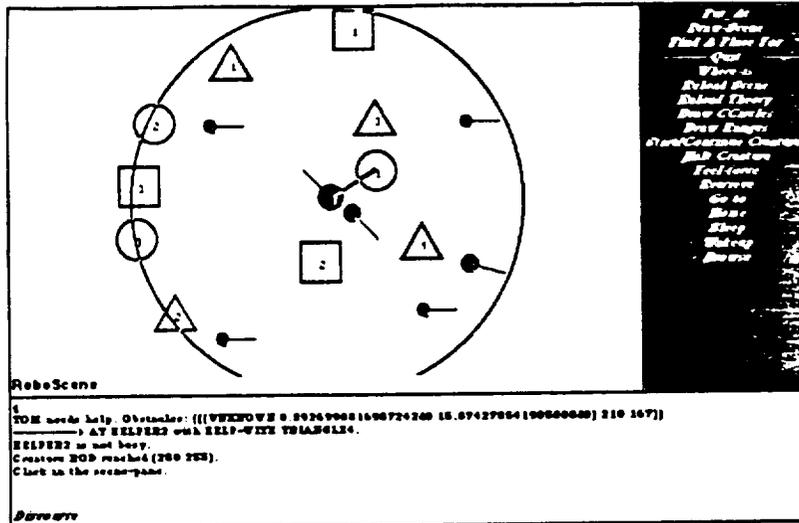


Figure 6

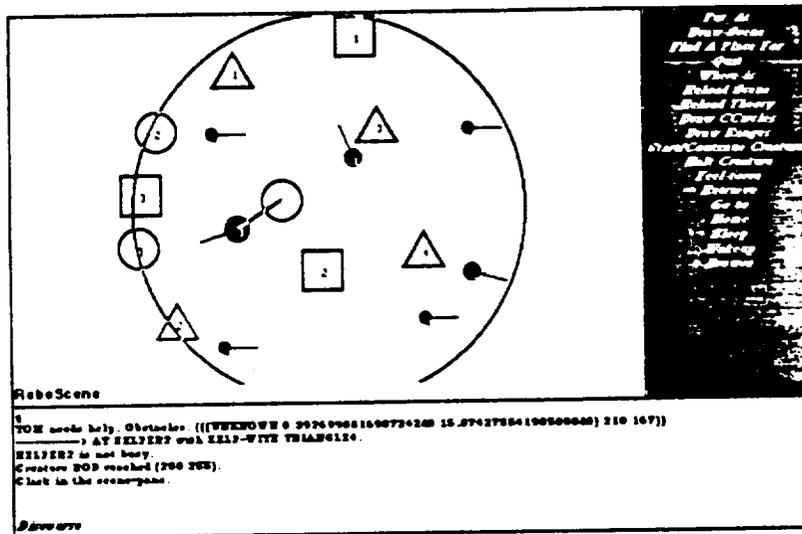


Figure 7

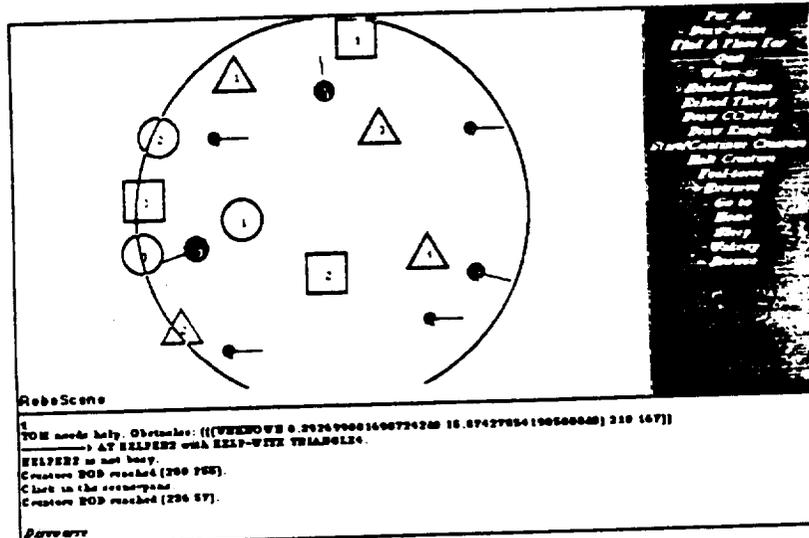


Figure 8

Figure 5 shows retriever #3, which is returning to its point of origin with Cylinder1, encountering retriever #1, which is returning to its point of origin. Both are blind but can avoid obstacles with their sonars, so they jostle each other as they move toward their destinations (see Figure 6). In Figure 7, both creatures are again on their way to fulfilling their tasks which are completed in Figure 8. The human user has simply specified two high-level tasks, and the retrievers were able to achieve them despite obstacles and the crossing of paths during the task execution.

CO-OP Objects and Operations

Table 1 shows the objects and operations used to achieve the capabilities discussed in the text of this report. There are six objects: three sensors, the generic one and the proximity and vision types; and three creatures, the generic one and one with and one without an arm. The types inherit data and operations from the generic objects. Operations are prefixed by colons; operations that are invoked after another operation are written with the word :after and the precedent operation name.

Each creature is instantiated via its position and orientation variables and the type and number of sensors it is given.

<u>Object</u>	<u>Type</u>	<u>Variables</u>	<u>Operations</u>
sensor		associated-creature, type	
	proximity-sensor	range-factor	:sense-from coords
	vision-sensor	range	:locate sought-object, :identify-object-at
creature		shape, number, cc-radius, coords, sensor-list, name, sleep	:sleep, :wakeup
	round-creature	speed, direction, target-direction, old-coords, caution-speed, obstacle-memory	:feel-force, :sense, :runaway, :move, :turn new-direction, :locate object, :locate-object-at coords, :wander
	round-creature-with-arm	target-object, target-location, arm-length, arm-direction, target-object-grasped, arm-extended?, blocking-obstacles, help-is-on-the-way, creature-being-helped, distance-and-time	:see-if-stalled, :move-to, :sense, :retrieve, :grasp object, :move, :help-with obstacle :after :set-blocking-obstacles

Table 1: Creature Objects and Operations

SENSORS

Sensors include a proximity sensor, analogous to a ringed set of ultra-sonic sensors; and an imaging sensor. Circumferential sound sensors are available with at least two low-cost commercial platforms (see (4), (5)). The vision system is a line-of-sight stereo system which yields range over a 2D matrix. In the experiments, only beacon creatures have vision sensors and cannot use them when the platforms are in motion.

Proximity Sensor--The proximity sensor has a range-factor which is a multiplier used with a creature's platform radius to get the range capability of the sensor ring. That a larger creature would have a greater proximity range seems intuitively correct (a more massive object needs more of a safety margin, speeds being equal).

The :sense-from operation simulates the action of the ring by extending a ray from the current platform coordinates out to the range limit, around the associated-creature's perimeter at 1 degree increments, and recording the angle to the nearest

obstruction. This simple algorithm was adequate for this study and has been since successfully implemented on a Hero 2000.

Vision Sensor--The vision sensor object has a limiting range. The :locate operation simulates a clockwise scan of the work area from the platform to locate and recognize an object. For each object within range, a LOS determination is made, and if positive, a "recognition" is made. Recognition is assumed to be a simple discrimination among classes of objects known to be in the environment. If the view is obstructed, the obstructing object is recognized, and its name is returned. If the object is recognized and it is the sought-object, then a ranging is made and the object name and its coordinates are returned. Otherwise the process continues until the whole work-area has been scanned, returning nil if unsuccessful.

The vision sensor can be asked to identify an object at or near a certain location. The :identify-object-at operation then simply trains the sensor on the given coordinates and, if unobstructed, returns the identity of the object nearest that location.

CREATURES

The generic creature has a shape, an identification number, a location, a list of on-board sensors, a name, and a sleep toggle for ignoring certain commands. The cc-radius is the radius of the equivalent circumscribing circle. For round creatures this is just the radius; for other objects, an equivalent radius is computed at instantiation. The :sleep and :wakeup operations set and reset the sleep variable.

Round-creature--We begin with creatures whose notions of motion are direction and speed. We believe many of the key ideas embodied in this report extend in a straight-forward manner to a space-based environment. The round-creature is a circular platform capable of omni-directional traverse. The creature has an organic 16-bit processor and 512k bytes of memory, which should be enough computing capability and memory to carry out the operations described below, and an RF broadcast transceiver capable of data communication.

The round-creature has speed (distance units per clock tick) and direction and can be given a target-direction in which to :wander. Caution-speed is the speed used when an obstacle is encountered, and obstacles are noted and placed into obstacle-memory. The obstacle memory is cleared after every 12 obstacle sensings (a kind of forgetting).

Sensing and Avoiding Obstacles--When the :feel-force operation is invoked in the creature's continuous control loop, the :sense operation is invoked, and if there is a positive sensing, the :runaway operation is invoked. Runaway sets the

direction to 180 degrees opposite the obstacle direction (by invoking a :turn in the new-direction), sets the speed to caution-speed, and executes the :move operation. Move advances the creature caution-speed units along its direction. Thus, if an obstacle or other creature approaches close enough, the creature will move away until :feel-force returns nil.

The :sense operation invokes the :sense-from operation of the creature's proximity sensors if any. When an obstacle is sensed, the token UNKNOWN and the location of the creature at the time are stored in obstacle-memory . Obstacle-memory is used to see if the creature is not making progress (see *Getting Stuck* below).

Locating and Identifying Objects--To locate an object, a round-creature asks its sensors each in turn to :locate the object. Only the vision sensors are capable of this operation. If the creature doesn't have such a sensor, it asks another creature to locate the object, which in turn recursively ask other creatures if necessary. This single request thread prevents simultaneous creature responses as in a broadcast method. Most often working creatures (e.g., retrievers) with only proximity sensors can't identify the object to be located. More typically, the request is to :locate-object-at the location which in turn converts to a call to the creature's vision sensor to :identify-object-at that location. In our typical setup, only beacons have vision sensors, and one of them ends up identifying and locating the object.

Round-creature-with-arm--This creature is typically endowed with a swivel retrieval mechanism: an arm that swivels either freely or under control about the platform axis, can be retracted and extended, and needs only to make contact with the objects to be retrieved (one can imagine simple or complex contact mechanisms from velcro pads to magnetized grippers). The :sense command is as described above but augmented to account for the objects in tow.

This creature is used primarily for retrieving a target-object at a target-location. It keeps track of whether its arm is extended or not, whether it has grasped the target-object and its rate of progress through the distance-and-time recordings. If impeded, the creature remembers the blocking-obstacles and whether help-is-on-the-way (stored as a list of the helper and its target-object). If it is helping another creature, it remembers the creature-being-helped.

Retrieving Objects--Retrieving objects is effected as a reaction plan, which is akin to the concept of a Universal Plan (13), once a target-object has been established by the user or another creature. Table 2 shows a retrieval plan. The preconditions of the plan are checked and the appropriate action is taken on each pass of the control loop.

<u>Precondition</u>	<u>Action</u>	<u>Comments</u>
(and target-object-grasped old-coords (< (distance coords old-coords) retrieve-speed))	Clear all memory of retrieval data, detach and close arm. If helping someone, clear helping data	Done
(and target-object-grasped (equal old-coords target-location))	move-to target-location	Heading home
target-object-grasped	set target-location to old-coords; if helping A, tell A to :set-blocking-obstacles to nil and to set help-is-on-the-way to nil	Turning for home
(and target-location (<= (distance coords target-location) arm-length) (not (object-usurped target-object)))	extend arm and :grasp object, setting target-object-grasped to true	Grab object
Target-object has a value	:locate target-object, and :move-to target-location	Still trying
No target-object	Stay out of trouble	Creature idle

Table 2: A Reaction Plan For Retrieval

To understand the plan, read up from the "Target-object has a value" precondition. When the creature is commanded to retrieve an object (target-object is set to the object name), it first :locates the target and then executes a :move-to the target-location from its old-coords (the position at which it received the :retrieve command). As soon as it is within an "arm's reach" of the object and the object is not being grasped by another creature, it grasps the object. Then it turns and heads for home, making the target-location the old-coords (move-to looks for a target-location). If the creature is helping another creature, it tells the other creature at this point that it has removed the blocking-obstacle and help is no longer on the way. When the creature is within retrieve-speed of the old-coords, the arm detaches from the object, is retracted and the retrieval is complete.

Since the preconditions are checked during each cycle of the control loop, the reaction plan provides robust operations, allowing for a successful retrieval in the wake of many unforeseen actions, e.g., the object moves, or another creature grabs the object first.

Getting Stuck--While retrieving, if the creature is blocked and help is being provided by another creature, the creature just sits and waits for the help. The obstacles are detected as part of the :move-to operation where the first action taken is a local sensing for obstacles. If any exist, a check is made to see if the creature is being stalled (:see-if-stalled) and then a :runaway is executed. Otherwise, a :move is executed at the resulting direction and speed(:move is the same as described above but taking any towed object into account).

To simply determine whether progress is being made, two cases are modeled: one, where, in a certain time period, the distance to the target-object has not changed appreciably; and two, where there has been a large time lapse since we last invoked the :move-to operation. The first case involves the instances where the creature is executing :move-tos and :runaways, but is not really getting any closer to the object. The second case involves executing continuous :runaways because of many obstacles as in the example above.

Each time a :move-to is invoked, the distance from the object and the time are recorded as lists in distance-and-time. For the first case, the difference between the current distance and the distance achieved the last time a :move-to was invoked is compared to a preset value. For the second case, if the elapsed time since the last invocation of :move-to is greater than a preset value, the creature is considered stuck. The values for this simulation (normal speed = 5 and caution speed =1) were 3 distance units and ten clock ticks respectively.

When :see-if-stalled determines the creature is stuck, the last obstacle recorded in obstacle-memory is considered the blocking-obstacle, and the :after operation for setting the blocking-obstacles sends a ":help-with obstacle" call out to other creatures. In the example, retriever #2 answered the :help-with obstacle call from retriever #3.

CONCLUSIONS AND FUTURE WORK

The robustness of the cooperative becomes more evident the more the simulation is run. Objects to be retrieved can be mischievously moved by the user, obstacles can be put in retrievers' paths, and in all but the most pathological of cases (two retrievers latch onto the same object and have a tug-of-war until the user intervenes), the creatures succeed. The current placement of the beacons is such that with only two beacons set at a diagonal across the work area, over half of the retrievals will be successful.

Two creatures can be sent after the same object with the result that the second creature chases and tracks the object until the other creature releases it. This behavior is interesting to watch in the simulation but underscores the need for human

guidance. Autonomous control of multiple agents is a current research endeavor. The described simulation experiments indicate that robot cooperatives will succeed well against unforeseen events but only for a time by themselves.

An enormous amount of work in algorithms for robot navigation has been done. Some of which are relatively simple to implement (e.g., (10), (8), and (1)), and we will be investigating them for our systems. In this vein, we plan to give each creature a simple map of the objects and the free space. We will then have the creatures broadcast the results of the retrievals so that all creatures can update their space maps, and thus better use navigation algorithms.

We are now moving the experiments onto actual robot platforms. Several low-cost commercial platforms are available (see for example (7) and (4)). In MITRE's Autonomous Systems Laboratory, we have two Hero 2000 robots for which many of the round-creature-with-arm competences have been implemented, but we are acquiring a more capable platform such as the Denning MRV-3. The robots are being programmed using the REX/GAPPS system (11) from Teleos Research. We plan to mount a stereopsis system which produces ranging information at video frame rates (9) on one of the Heroes (as a beacon); the Denning and the other Hero will be retrievers. Experiments with these systems will help us understand how well the co-op ideas stand the test of real systems in real environments.

REFERENCES

- (1) Arkin, Ronald C. Motor Schema-Based Mobile Robot Navigation, in *International Journal of Robotics Research*, Vol 8, No. 4 August 1989. pp 92-112
- (2) Bloom, Ben; McGrath, Debra; Sanborn, Jim, A Situated Reasoning Architecture for Space-Based Repair and Replace Tasks, *Proceedings of the Goddard Conference On Space Applications of AI*, NASA Conf Pub #3033, 1989
- (3) Brooks, Rodney A, A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, 2(1), March 1986
- (4) Brooks, Rodney A Autonomous Mobile Robots, in *AI In the 1980s and Beyond*, Grimson and Patil eds, MIT Press, 1987
- (5) *The Hero 2000 Technical Manual*, Heath/Zenith Systems, Benton Harbor, MI 1989
- (6) Kaelbling, L. P., An Architecture For Intelligent Reactive Systems, in *The 1986 Workshop on Reasoning About Actions and Plans*, M. Georgeff and A. Lansky eds, Morgan Kaufman, 1986
- (7) Long-ji Lin; Mitchell, Tom M.; Philips, Andrew; Simmons, Reid, A Case Study In Robot Exploration, CMU-RI-TR-89-1, CMU Robotics Institute, Jan 1989
- (8) Meng, Alex C.-C.; Wand, Marty; Hwang, Vincent S., A Methodology of Map-Guided Autonomous Navigation with Range Sensors in Dynamic Environments, *SPIE 3rd Conf on AI Applications*, 1988
- (9) Nishihara, H. K. Practical Real-time Imaging Stereo Matcher, in *Optical Engineering*, 23 (5), 536-545 (Sep/Oct) 1984
- (10) Rao, Nagewara S. V., Algorithmic Framework For Learned Robot Navigation in Unknown Terrains, *Computer*, June 1989
- (11) Rosenschein, Stanley J. and Kaelbling, Leslie P., The Synthesis of Digital Machines With Provable Epistemic Properties, SRI Technical Note #412, SRI International, Menlo Park, CA 1987
- (12) Sacerdoti, Earl D., Plan Generation and Execution For Robotics, SRI TR, 1983
- (13) Schoppers, Marcel, Universal Plans For Reactive Robots In Unpredictable Domains, in *Proceedings of IJCAI 10*, 1987

Design Knowledge Capture for a Corporate Memory Facility

John H. Boose, David B. Shema, Jeffrey M. Bradshaw

Boeing Advanced Technology Center, 7L-64
Boeing Computer Services, P.O. Box 24346
Seattle, WA 98124

Abstract

Currently, much of the information regarding decision alternatives and trade-offs made in the course of a major program development effort is not represented or retained in a way that permits computer-based reasoning over the life cycle of the program. The loss of this information results in problems in tracing design alternatives to requirements, in assessing the impact of change in requirements, and in configuration management.

To address these problems, we are studying the problem of building an intelligent, active *corporate memory facility* which would provide for the capture of the requirements and standards of a program, analyze the design alternatives and trade-offs made over the program's lifetime, and examine relationships between requirements and design trade-offs. Early phases of the work have concentrated on design knowledge capture for the Space Station Freedom. We have demonstrated and are extending tools that help automate and document engineering trade studies (the topic of this paper), and we are developing another tool to help designers interactively explore design alternatives and constraints.

1.0 Introduction - Overall Problem

Under NASA contract NAS2-12108, the Boeing Advanced Technology Center (ATC) is conducting research leading to a corporate memory facility (CMF). A CMF would provide facilities for capturing and using decision history and rationale throughout a major program's life cycle. This effort is jointly funded by OAST's AI Program and the Space Station Freedom Advanced Development Program.

Currently, much of the information regarding alternatives considered and trade-offs made in the course of a major program development effort is not represented or retained in a way that permits computer-based reasoning over the life cycle of the program. The loss of this information results in problems in tracing alternatives to requirements, in assessing the impact of change in requirements, and in configuration management (Boeing Computer Services, 1989a,b).

There is not an integrated set of capabilities to assist in generating and evaluating or reevaluating program alternatives. The lack of this capability results in such problems as belated reaction to changes in requirements and inability to consider a reasonable number of alternatives.

2.0 A Corporate Memory Facility

To address these problems, we are studying the problem of building an intelligent, active corporate memory facility which would provide for the capture of the requirements and standards of a program, alternatives considered and trade-offs made over its lifetime, and relationships between these. The CMF would provide for requirements traceability, impact assessment, automation and/or assistance in the generation and evaluation of alternatives, and configuration management.

The CMF would support interactive problem solving across diverse areas such as the aerospace engineering disciplines (propulsion, weights, and aerodynamics). In operational use, a CMF would reduce life-cycle flow time and cost and improve the quality of program deliverables. Similar benefits could be realized by applying information accumulated in the CMF for one program to other related programs.

In initial phases of this work, the ATC is studying core corporate memory facility ideas, preparing CMF technical reports detailing study results, and building feasibility demonstrations. In conjunction with NASA, the Space Station Freedom Program was selected as a testbed; within this test bed we are concentrating on design knowledge capture. In 1989 the ATC examined aspects of the Power subsystem and the Environmental Control and Life Support (ECLS) subsystem. We also used our tools in a portion of the 1989 Space Station Freedom technical audit to investigate the rationale for a previous design decision.

Through the series of demonstrations, we hope to show a novel integration and extension of design knowledge capture ideas by:

- a. Tailoring knowledge acquisition and process control tools for *engineering trade studies*, a significant and feasible part of design knowledge capture.
- b. Digitally recording speech as an unobtrusive method of capturing design rationale at the trade study workstation.
- c. Developing an interactive design alternative generation aid.

3.0 Automating Engineering Trade Studies

We are focusing on trade studies in the design knowledge capture area because -

- a. They exhibit a microcosmic path through the full cycle of design information, including requirements linkage, generation and comparison of alternatives, and decision documentation.
- b. Many design engineers are familiar with trade studies and are comfortable using them to compare alternatives in quantitative terms.
- c. Even though different methodologies for trade studies are available, little has been done to automate them.
- d. A trade study tool would be immediately useful in a variety of domains, regardless of the success of the overall design knowledge capture or CMF effort.
- e. Existing ATC tools could be extended to help perform portions of trade studies.

Trade studies are performed, in part, to avoid a designer's tendency to go directly to a design based on past experience, rather than trying to find a design that may better satisfy overall program requirements. Trade studies are often performed to help establish overall system configurations, study the detailed design of individual configuration items to provide the most cost-effective solution, and evaluate alternate solutions when the need for change occurs.

There are two general types of trade study criteria: limits which must be satisfied by any candidate system (*go/no go criteria* or *hard constraints*), and attributes upon which a ranking can be based (*soft constraints*)

Candidates are usually filtered using hard constraints and then ranked for comparison using soft constraints. Trade trees are used to decompose large numbers of candidates into groups for tractability. Paths through the tree show total configurations. Typical trade study criteria include accuracy, lifetime, power output, stability, sensitivity, bandwidth, low weight, low power, minimum dimensions, operational simplicity, electromagnetic compatibility, reliability,

survivability, schedule, cost, safety, and risk. Criteria are usually weighted. The results are usually shown in a *trade study matrix* - a table showing the alternatives, criteria, ratings, and weights.

After candidates are rated and scored, a sensitivity analysis can be performed. This shows the sensitivity of the decision to changes in the value of attributes, weights, costs, and subjective estimates.

In our early work on the CMF we demonstrated the capture of trade study information and rationale (Figure 1). In the future, this information will be available through the Technical and Management Information System (TMIS). We are examining several report formats based on current trade study practices. The information necessary for these reports will provide the foundation for the knowledge capture process.

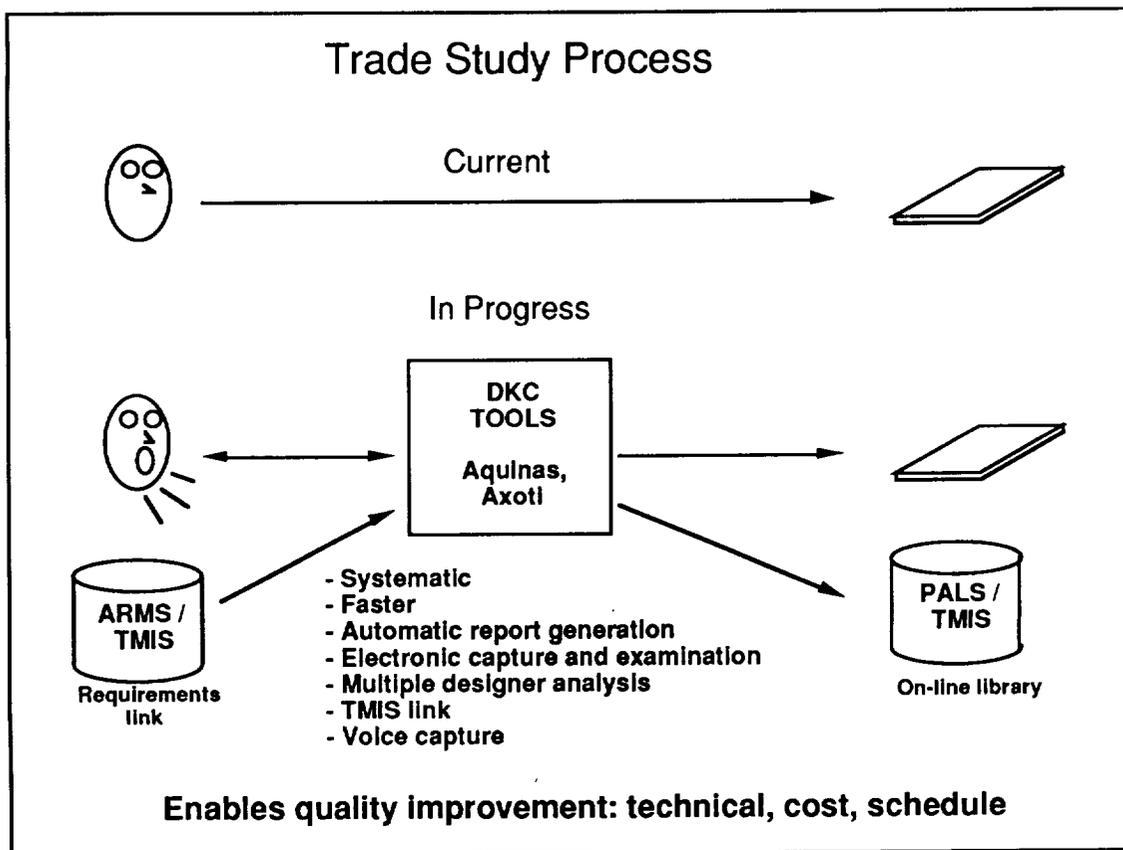


Figure 1. Automating the trade study process.

4.0 Design Knowledge Capture Tools

Two tools, Aquinas and Axotl, were used to build the first demonstration. An additional set of tools (MANIAC, HyperCard, and MacRecorder) was used to capture voice rationale and associate it with the Aquinas knowledge base for interactive playback.

4.1 Aquinas and Trade Studies

Aquinas interviewed experts in several trade study domains and captured candidate and criteria information leading to rank-ordered candidate selections. In the power domain, additional rationale was captured as voice input. In the ECLSS domain, conflicting opinions from multiple designers were captured, analyzed, and documented.

Power subsystem - Chuck Olson, a design engineer in Boeing Aerospace, used Aquinas to build two separate trade studies for the interface between a computer and automatic circuit breakers. Brian Smith, another Boeing Aerospace design engineer, offered advice on building an electronic trade study process assistant.

Environmental Control and Life Support subsystem - Jim Knox, a NASA design engineer at Marshall Space Flight Center, used Aquinas to build a trade study for carbon dioxide removal on Space Station Freedom in the year 2000. Allen Basckay, another NASA design engineer at Marshall Space Flight Center, added additional information to this trade study.

Technical Audit Item #85 - John Palmer, O'Keefe Sullivan, and Carl Case, Boeing Aerospace, used Aquinas to document a 1986 decision about the placement of the pressurized logistics module.

Aquinas is a workbench developed by the Boeing Advanced Technology Center for acquiring and analyzing expert knowledge for solving diagnostic, structured selection, classification, and other problems (Figure 2). In the CMF context, Aquinas is used to acquire knowledge about requirements and alternatives from individuals or groups of experts, and then assists in merging that knowledge into a single knowledge base. Weights may be assigned to both requirements and their refinements. This knowledge may be merged automatically by Aquinas or by consensus of the program staff using Aquinas as an assistant. Aquinas supports similar capabilities for acquiring compound alternatives.

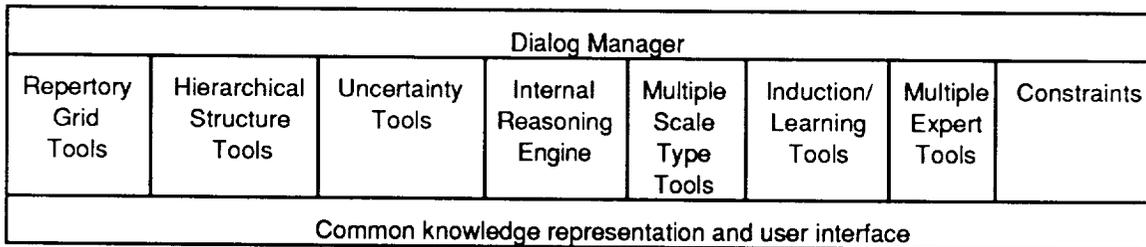


Figure 2. Aquinas consists of several tool sets that assist different knowledge acquisition tasks. General advantages of Aquinas include integration of multiple methods and techniques, rapid prototyping and feasibility analysis, generation of expert enthusiasm, multiple mediating representations, embedded testing, and life cycle support for verification, delivery, and maintenance.

Aquinas, an expanded version of the Expertise Transfer System (ETS; Boose, 1984, 1985, 1986a,b), combines ideas from psychology and knowledge-based systems to support knowledge acquisition tasks. These tasks include eliciting distinctions, decomposing problems, combining uncertain information, incremental testing, integration of data types, automatic expansion and refinement of the knowledge base, use of multiple sources of knowledge, use of constraints during inference, and providing process guidance (Boose and Bradshaw, 1987; Boose, Bradshaw, and Shema, 1989). Aquinas interviews experts and helps them analyze, test, and refine knowledge. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined. Results from user consultations are derived from information propagated through hierarchies.

Using Aquinas, rapid prototypes of knowledge-based systems can be built in as little as one hour, even when the expert has little understanding of knowledge-based systems or has no prior training in the use of the tool. The interviewing methods in Aquinas are derived from George Kelly's Personal Construct Theory and related work (Kelly, 1955; Shaw and Gaines, 1987;

Boose, 1988). Kelly's methods and theory provide a rich framework for modeling the qualitative and quantitative distinctions inherent in an expert's problem-solving knowledge.

Aquinas tools mentioned here are explained more fully elsewhere (Boose and Bradshaw, 1987; Boose, 1988; Kitto and Boose, 1988; Shema and Boose, 1988; Bradshaw and Boose, 1989).

Extended repertory grids in Aquinas are a compact and easily understood form of expertise representation for many types of knowledge. Repertory grids can be analyzed, refined, tested, and maintained more easily than a corresponding, larger rule or frame knowledge base. In Aquinas, we have augmented repertory grid structures to include hierarchies, constraints, structures for eliciting and reasoning about knowledge from multiple experts, multiple variable types, and accommodate forms of machine learning. Generally, these analysis capabilities and compact, higher-level *mediating representations* of expert knowledge make knowledge bases easier to inspect, analyze, maintain, test, and improve. We use a test case-based approach within Aquinas for performance measurement, verification, and maintenance, and automatic knowledge base improvement. This method helps find holes and weaknesses in the knowledge base, and provides facilities for verifying knowledge consistency, accuracy, and sanity range.

Refinement methods in Aquinas include implication and similarity analyses, completeness checking, hole filling, cluster analyses, generalization, automatic rule production, internal testing and debugging aids, and graphic representation transformation. Expertise from multiple experts or other knowledge sources can be represented and used separately or combined, giving consensus and dissenting opinions among groups of experts. Recent progress on Aquinas has been in the areas of knowledge base performance measurement, knowledge base maintenance, interacting trait constraints, consultation graphics, and eliciting strategic and procedural knowledge. Experiments show how Aquinas can automatically improve knowledge bases and even suggest new problem-solving information. Forms of interactive and automatic machine learning are also employed by Aquinas (Boose, Bradshaw, and Shema, 1989).

Aquinas exists in several "C"-based versions that run on different microprocessor platforms and a fuller development version that runs on Sun workstations and Xerox Lisp Machines.

4.2 Axotl System

In the first demonstration, Chuck Olson used Axotl to elicit an electronically-based model of the trade study process.

Axotl, developed at the Boeing Advanced Technology Center, integrates a set of computer-based decision analysis tools with a knowledge-based system. The decision analysis tools are designed for problems requiring careful consideration of uncertainty and complex tradeoffs. In the context of CMF, alternatives and requirements generated by Aquinas can be analyzed using decision analysis representations to determine the suitability of various alternatives and to gauge the impact of changes in design requirements or circumstances. Influence diagrams are used to represent information, alternatives, and preferences both graphically and mathematically. Our experience has shown that they are an effective way of communicating important issues among participants. Axotl also employs other forms of knowledge representation that may prove useful as part of a CMF. For example, Boeing has extended and generalized an AND/OR graph representation for goals and activities ("activity graphs") that can be used to dynamically construct and evaluate cyclic plans for achieving a set of process requirements.

Axotl is written in the ParkPlace Smalltalk-80 development environment on the Apple Macintosh II. Versions of Smalltalk-80 exist for Sun, Apollo, Hewlett-Packard, IBM, and Apple hardware.

4.3 MANIAC, HyperCard, and MacRecorder

Together, MANIAC, HyperCard, and MacRecorder were used to record and play back voice rationale.

In the first demonstration, design decision rationale was captured on a tape recorder during Aquinas sessions. To demonstrate feasibility, parts of these recordings were processed using MacRecorder on a Macintosh and stored in HyperCard. MANIAC, an ATC shell that controls communication between Axotl, Aquinas, HyperCard, and other application programs, receives commands from Aquinas to play back digitally recorded voice based on particular Aquinas knowledge base objects. Designers and others who later examine the trade study decision rationale can optionally play back this recorded voice information.

In future demonstrations we will link MacRecorder and Aquinas more directly so that designers may enter and edit voice input directly while using Aquinas. This will be a relatively unobtrusive way to enter rationale (as opposed to text entry) in a cost effective manner. Digitally recorded voice information could eventually be stored and played back as design decision rationale in TMIS in a manner similar to many digital phone message systems.

MANIAC is described more fully in (Bradshaw, Covington, Russo, and Boose, 1988).

4.4 CANARD

As part of the design process, competing alternatives are generated and evaluated for suitability. The best alternative emerges as the result. Unfortunately, constraints, tradeoffs, and other considerations made during the exploration of the design are usually lost, making it impossible to review or easily modify them at a later time. If a modification to the design is required, the designers may have to redo the entire task.

We started development of CANARD, an automated tool which uses possibility tables, constraints, and knowledge bases to capture significant portions of the design process and assist in the generation of alternative solutions consistent with design goals and design constraints (Shema, Bradshaw, Covington, and Boose, 1989). Using a possibility table, a designer identifies the components of an acceptable design, specifies possibilities for each component, develops criteria reflecting preferences among possibilities, and supplies constraints governing compatibility between components and overall design considerations. The designer next interactively explores design alternatives by selecting possibilities for each component, modifying and/or adding components and possibilities as insight into the solution is gained. He then analyzes and stores the many alternative solutions for later retrieval.

For large problems, an iterative search procedure hypothesizes new constraints based on examples of previously-defined design alternatives, and proposes new design alternatives based on permutations of the constraint space. The tool keeps track of what has been tried and assists the designer in covering important aspects of the possible solution space.

CANARD is written in the ParkPlace Smalltalk-80 development environment on the Apple Macintosh II. Versions of Smalltalk-80 exist for Sun, Apollo, Hewlett-Packard, IBM, and Apple hardware.

5.0 Example Trade Study - Technical Audit Item #85

In 1989 a technical audit was performed on the Space Station Freedom for the program's content and implementation planning in relationship to performance, design, and validation

requirements. One concern raised during the technical audit was a 1986 decision about the placement of the pressurized logistics module (PLM). Using Aquinas, we hoped to develop a process for capturing the decision rationale on this topic and similar ones.

First we described our problem and proposed process to a group of designers at Boeing in Huntsville, Alabama, who were or who are involved with the placement of the PLM. We then used Aquinas in two sessions with two teams of designers. One session lasted 1-1/4 hours, one session lasted 1-1/2 hours. We elicited trade study matrices from each team and combined the results, using Aquinas to show the combined rank-ordering. The decisions developed using Aquinas agreed with and documented the current placement of the PLM.

Here we describe the steps that were performed with Aquinas for the technical audit.

Step 1. Aquinas elicited nine alternative PLM locations from Team 1 (Node 1 Zenith, Node 1 Nadir, etc.).

Step 2. Aquinas elicited a preliminary set of decision criteria by using triadic comparison. Groups of three solutions were compared and designers were asked to give discriminating criteria:

Think of an important new criterion that two of NODE.1.ZENITH, NODE.1.NADIR, and NODE .2.ZENITH share, but that the other one does not. What is that trait? (Enter a CR to skip over.)

NEW TRAIT (EXTREME)** BETTER MSC REACH

What is that criterion's opposite as it applies in this case?

NEW TRAIT (OPPOSITE)** WORSE MSC REACH

What is the name of a scale or concept that describes BETTER.MSC.REACH / WORSE.MSC.REACH?

NEW TRAIT (CONCEPT)** MSC REACH

Think of an important new criterion that two of NODE.1.NADIR, NODE.2.ZENITH, and NODE.2.NADIR share, but that the other one does not. What is that characteristic? (Enter a CR to skip over.)

NEW TRAIT (EXTREME)** CLOSE TO HAB MODULE

What is that criterion's opposite as it applies in this case?

NEW TRAIT (OPPOSITE)** FARTHER FROM HAB MODULE

What is the name of a scale or concept that describes CLOSE.TO.HAB.MODULE / FARTHER.FROM.HAB.MODULE?

NEW TRAIT (CONCEPT)** HAB MODULE PROXIMITY

Step 3. The designers rated each alternative on each criterion. By default, Aquinas supplies ordinal scales from 1 to 5. Designers may change the scale type (to nominal, interval, or ratio) or range for convenience or more precision.

Step 4. The designers assigned a relative weight to each criterion. At this point an initial trade study matrix was complete (Figure 3).

TEAM.1. SOLUTION TRAIT									
4	1	4	1	2	3	2	2	1	1. (5) MSC.REACH: BETTER.MSC.REACH(1)/ WORSE.MSC.REACH(5) [ORDINAL 1]
4	4	1	1	1	4	3	2	2	2. (1) HAB.MODULE.PROXIMITY: CLOSE.TO.HAB.MODULE(1)/ FARTHER.FROM.HAB.MO
1	1	1	1	2	4	4	4	4	3. (3) CG.SHIFT.IMPACT: LESS.CG.SHIFT(1)/ GREATER.CG.SHIFT(5) [ORDINAL 1]
4	4	1	1	1	5	5	4	4	4. (1) JAPANESE.MODULE.PROXIMITY: CLOSER.TO.JAPANESE.MODULE(1)/ FARTHER.
2	1	2	1	5	4	4	5	4	5. (2) GROWTH./PATH: BETTER.FOR.GROWTH.PATH(1)/ WORSE.FOR.GROWTH.PATH(5)
5	5	5	5	5	3	3	3	1	6. (5) PAYLOAD.BAY.BLOCKING: BLOCKING.PAYLOAD.BAY(1)/ NOT.BLOCKING.PAYLO
1	5	1	5	3	1	3	3	5	7. (4) EXPOSURE.TO.MICRO.METEOROIDS: MORE.EXPOSURE.TO.MICRMETEOROIDS(1)
1	2	3	4	5	6	7	8	9	
1.	NODE.1.ZENITH								
2.	NODE.1.NADIR								
3.	NODE.2.ZENITH								
4.	NODE.2.NADIR								
5.	NODE.2.PORT								
6.	NODE.3.ZENITH								
7.	NODE.3.STARBOARD								
8.	NODE.4.PORT								
9.	NODE.4.NADIR								
TEAM.1.	SOLUTION								

Figure 3. Initial technical audit trade study matrix from Team 1.

Step 5. The designers used several of Aquinas' analysis tools to discover patterns in the collected information. Implication analysis showed logical generalizations that, for this application, provided a sanity check. A cluster analysis and similarity analysis showed the degree of similarity and redundancy between alternatives and between criteria.

Step 6. Aquinas scored the alternatives by eliciting preferred criteria values from the designers. For example, the designers said they would prefer alternatives that were *better for the station growth path* and had *less effect on the station center of gravity*. For Team 1, Aquinas produced the following results:

1 : NODE.2.NADIR	(1.00)
2 : NODE.1.NADIR	(0.93)
3 : NODE.2.PORT	(0.71)
4 : NODE.2.ZENITH	(0.61)
5 : NODE.4.NADIR	(0.54)
6 : NODE.1.ZENITH	(0.54)
7 : NODE.4.PORT	(0.48)
8 : NODE.3.STARBOARD	(0.48)
9 : NODE.3.ZENITH	(0.31)

Step 7. The second team used Aquinas to independently develop and analyze their own trade study matrix.

Step 8. Both matrices were combined and Aquinas again scored the alternatives, this time showing the *consensus* scores as well as the contributions from both individual teams. The teams are weighted in this example for purposes of illustration (Team 1 has received a weight of 40%, Team 2 a weight of 60%). Teams or individuals may be weighted for technical or other reasons.

Combined results:						
1	NODE_2_NADIR	0.89	TEAM_1	1.00	40%	TEAM_2 0.82 60%
2	NODE_1_NADIR	0.81	TEAM_1	0.93	40%	TEAM_2 0.73 60%
3	NODE_2_PORT	0.71	TEAM_1	0.71	40%	TEAM_2 0.70 60%
4	NODE_2_ZENITH	0.64	TEAM_1	0.61	40%	TEAM_2 0.65 60%
5	NODE_1_ZENITH	0.55	TEAM_1	0.54	40%	TEAM_2 0.56 60%
6	NODE_4_NADIR	0.48	TEAM_1	0.54	40%	TEAM_2 0.44 60%
7	NODE_4_PORT	0.43	TEAM_1	0.48	40%	TEAM_2 0.39 60%
8	NODE_3_ZENITH	0.40	TEAM_1	0.31	40%	TEAM_2 0.46 60%
9	NODE_3_STARBOARD	0.28	TEAM_1	0.48	40%	TEAM_2 0.15 60%

Given this information, Aquinas displayed the most dissenting opinion beside the consensus. The dissenting opinion is found by computing a correlation score between each team and the consensus; the team with the lowest correlation score is listed as the dissenting opinion. Dissenting

opinions show the user the range of opinion about a decision, not just the top rated list. In this case, both teams showed a high correlation - both teams were in substantial agreement. This can give the user confidence that the top rated alternatives were sound choices.

Correlation scores for all experts:

TEAM_2 .96
TEAM_1 .90

TEAM_1 has the most dissenting opinion.

TEAM_1	/	Consensus
1 : NODE_2_NADIR 0.81	/	NODE_2_NADIR 0.89
2 : NODE_1_NADIR 0.72	/	NODE_1_NADIR 0.81
3 : NODE_2_PORT 0.71	/	NODE_2_PORT 0.71
4 : NODE_2_ZENITH 0.66	/	NODE_2_ZENITH 0.64
5 : NODE_4_NADIR 0.62	/	NODE_1_ZENITH 0.55
6 : NODE_1_ZENITH 0.42	/	NODE_4_NADIR 0.48
7 : NODE_3_STARBOARD 0.41	/	NODE_4_PORT 0.43
8 : NODE_4_PORT 0.34	/	NODE_3_ZENITH 0.40
9 : NODE_3_ZENITH 0.21	/	NODE_3_STARBOARD 0.28

Near-future capability. We will be building a TMIS-based menu query mechanism that would be able to answer several types of questions about a trade study:

Q. Why did NODE.2.ZENITH do better than NODE.1.ZENITH?

A. It rated higher on CLOSE.TO.HAB.MODULE (1 vs. 4 on a scale of 1 to 5) and CLOSE.TO.LAB. MODULE (1 vs. 4 on a scale of 1 to 5)

Q. Why did NODE.1.NADIR and NODE.2.NADIR do better than NODE.1.ZENITH and NODE.2.ZENITH?

A. They always rated higher on BETTER.MSC.REACH, BETTER.FOR.GROWTH, and LESS.EXPOSURE.TO.MICROMETEORIDS.
They sometimes rated higher on CLOSE.TO.HAB.MODULE and CLOSER.TO.JAPANESE.MODULE.

Q. If LESS.EXPOSURE.TO.MICROMETEORIDS were the only criterion, how would the alternatives be ranked?

A. 5: NODE.1.NADIR, NODE.2.NADIR, NODE.4.NADIR
4:-
3: NODE.3.STARBOARD, NODE.4.PORT
2: NODE.2.PORT
1: NODE.1.ZENITH, NODE.2.ZENITH, NODE.3.ZENITH

Q. What are the most critical criteria?

A. (Perform a sensitivity analysis to determine critical criteria; list criteria with high weights.)

Results. Boeing and MSFC engineers who used the tool were very enthusiastic about its potential. It was decided to try and use this methodology for other aspects of the station's preliminary design phase.

6.0 Conclusions and Future Work

The Boeing Advanced Technology Center (ATC) is conducting research leading to a Corporate Memory Facility (CMF). A CMF would provide facilities for capturing and using decision history and rationale throughout a major program's life cycle.

Initially the ATC is preparing CMF technical reports and building feasibility demonstrations. In conjunction with NASA, The Space Station Freedom Program was selected as an application; within this domain we are concentrating on design knowledge capture. We examined aspects of the Power subsystem and the Environmental Control and Life Support (ECLS) subsystem. We also participated in one aspect of the Space Station Freedom technical audit.

Significant progress was made in helping automate the process of performing engineering trade studies. Other steps in the design knowledge cycle - alternative generation, comparison, evaluation, and documentation - were also demonstrated. In the next phase we will continue to

extend our tools to further automate trade studies, strengthen our links to TMIS, and continue work on CANARD.

Acknowledgements

Thanks to Guy Boy, Kathleen Bradshaw, Ray Carnes, Carl Case, Arch Cousins, Stan Covington, Walter Cowart, Tom Dollman, Bob Drosdzal, Larry Frank, Mike Freeman, Peter Friedland, Scott Herman, Randy Humphries, Randy Johnson, Rich Keller, Cathy Kitto, Jim Knox, Wendy Nikolich, Guy Olney, Chuck Olson, John Palmer, Tim Rau, Tim Saito, Cecelia Savaard, Suzanne Shema, O'Keefe Sullivan, Gregg Swietek, Bryan Walls, David Weeks, and Al Winters for their advice, contributions, and support. ETS, AQUINAS, AXOTL, CANARD, and MANIAC were developed at the Boeing Advanced Technology Center, Boeing Computer Services, Seattle, Washington.

References

- Boeing Computer Services (1989a). Information to be Retained in a Corporate Memory Facility, CMF TR1, NASA NAS2-121108, April.
- Boeing Computer Services (1989b). Corporate Memory Facility Demonstration 1: Knowledge Representation and Trade Studies, CMF DEMO1, NASA NAS2-121108, April.
- Boose, J. H. (1984). Personal Construct Theory and the Transfer of Human Expertise, in the Proceedings of the National Conference on Artificial Intelligence (AAAI-84), Austin, Texas, p. 27-33.
- Boose, J. H. (1985). A Knowledge Acquisition Program for Expert Systems Based On Personal Construct Psychology, *International Journal of Man-Machine Studies*, Vol. 23, p.495-525.
- Boose, J. H. (1986a). Expertise Transfer for Expert System Design, New York: Elsevier.
- Boose, J. H. (1986b). Rapid Acquisition and Combination of Knowledge from Multiple Experts in the Same Domain, *Future Computing Systems Journal*, Vol. 1, No. 2, p. 191-216.
- Boose, J. H. (1988). Uses of Repertory Grid-Centred Knowledge Acquisition Tools for Knowledge-Based Systems, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, Vol. 29, No. 3, p. 287-310.
- Boose, J. H., and Bradshaw, J. M. (1987). Expertise Transfer and Complex Problems: Using Aquinas as a Knowledge Acquisition Workbench for Expert Systems, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 1, *International Journal of Man-Machine Studies*, Vol. 26, No.1, p.3-28; also in Boose, J. H., and Gaines, B. R. (eds), *Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems*, New York: Academic Press, 1988, p. 39-64.
- Boose, J. H., Bradshaw, J. M., Shema, D. B. (1989). Recent Progress in Aquinas: A Knowledge Acquisition Workbench, *Knowledge Acquisition: An International Journal*, in press.
- Bradshaw, J. M., Covington, S. P., Russo, P. J., and Boose, J. H. (1988). Folie a Deux: Integrating Aquinas, a Personal-Construct-Based Knowledge Acquisition Workbench, with Axotl, a Knowledge-Based Decision Analysis Workbench, *International Journal of Man-Machine Studies*, in press.
- Bradshaw, J. M., and Boose, J. H. (1989). Decision Analysis Techniques for Knowledge Acquisition: Combining Information and Preferences Using AQUINAS, Special Issue on the Second Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, in press.
- Carnes, J. R., Olson, A. E., and Praharaj, M. (1988). Design Knowledge Workstation, 1988 Final Report, Boeing Computer Services, PO Box 1470, M/S JA-74, Huntsville, AL.
- Kelly, G. A. (1955). *The Psychology of Personal Constructs*, New York: Norton.
- Kitto, C. M., and Boose, J. H. (1988). Heuristics for Expertise Transfer: The Automatic Management of Complex Knowledge Acquisition Dialogs, in Boose, J. H. and Gaines, B. R. (eds.), *Knowledge-Based Systems Vol. 2: Knowledge Acquisition Tools for Expert Systems*, London: Academic Press.
- Shaw, M. L. G., and Gaines, B. R. (1987). Techniques for Knowledge Acquisition and Transfer, special issue on the 1st Knowledge Acquisition for Knowledge-Based Systems Workshop, 1986, Part 5, *International Journal of Man-Machine Studies*, Vol 27., No. 3, p. 251-280.
- Shema, D. B., and Boose, J. H. (1988). Refining Problem-Solving Knowledge in Repertory Grids Using a Consultation Mechanism, special issue on the 2nd Knowledge Acquisition for Knowledge-Based Systems Workshop, 1987, *International Journal of Man-Machine Studies*, v. 29 n. 4, p.447-460.
- Shema, D. B., Bradshaw, J. M., Covington, S. P., Boose, J. H. (1989). Design Knowledge Capture and Alternative Generation Using Possibility Tables in CANARD, Proceedings of the 4th Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada, October, 1989.

Capturing Design Knowledge

Brian A. Babin and Rasiah Loganantharaj

Center for Advanced Computer Studies
University of Southwestern Louisiana
P.O. Box 44330, Lafayette, Louisiana 70504

Abstract

This paper proposes a scheme to capture the design knowledge of a complex object including functional, structural, performance, and other constraints. Further, the proposed scheme is also capable of capturing the rationale behind the design of an object as a part of the overall design of the object. With this information, the design of an object can be treated as a case and stored with other designs in a case base. A person can then perform case-based reasoning by examining these designs. Methods of modifying object designs are also discussed. Finally, an overview of an approach to fault diagnosis using case-based reasoning is given.

1 Introduction

At the abstract level a design task involves finding a consistent assignment for a set of variables that together define the object desired and satisfy the functional, structural, performance, and other constraints placed upon the object. In other words, a design task involves solving a constraint satisfaction problem where the constraints define the functional, structural, performance, and other requirements of the object[6]. Unfortunately the constraints defining a new object are often incomplete, ill-defined, or inconsistent. In such situations the design process involves the iteration of the following steps: refinement of the object specification followed by partial constraint satisfaction[2][3]. This process is repeated until a complete specification of the object is achieved.

The design process can be systematic, as we briefly discussed, or ad hoc. Unfortunately many of the designs to date were developed in an ad hoc manner. The method of systematic design is not very well understood. As a result, there have been a number of recent efforts towards a better understanding of the design process[3][6][10][11].

In this paper we are interested in capturing the design knowledge of an object. The purposes are twofold: (1) capturing the design knowledge makes it much easier

to understand, modify, or enhance the design, and (2) studying the designs of objects and the knowledge which needs to be represented will help in systematizing the design process.

In order to understand the design knowledge and representation issues involved, we have chosen to study the process of designing complex objects. Loosely stated, a complex object is anything with a nontrivial design consisting of a number of parts and their interconnections. Examples of complex objects include toasters and table lamps at the relatively simple end and jet engines at the complex end. Hereafter a complex object is simply referred to as an object.

When representing an object we need to capture the decompositional, hierarchical, functional, structural, and physical knowledge for that object[1]. We also need to capture the design knowledge including the decisions made while designing the object, the rationales behind these decisions, and any alternatives that were considered. Given the design of an object, it may be relatively easy to capture the decompositional, hierarchical, functional, structural, and physical knowledge of that object. However, the acquisition of design decisions, alternatives, and rationales is likely to be quite challenging, especially for an ad hoc design.

All of this information about an object can be gathered together to form a particular *design experience*. A designer may study this design experience while designing another object which is similar in some respect. The desire to examine past design experiences makes case-based reasoning (CBR) a natural way to access these experiences. CBR can be applied here by treating a design experience as a case and building a case base of various design experiences. Background information on CBR or, more generally, memory-based reasoning can be found in [5], [7], and [9]. Designs can be indexed according to parts used, functionality, and other features. This allows a designer to examine the design experiences of other objects with similar parts or functionality to help in making design decisions. We also briefly address the issue of diagnosing faults which cause an object to malfunction. We employ CBR to find the cause of a malfunction by examining the causes of previous similar malfunctions.

The rest of the paper is arranged as follows. First, the physical representation of an object is described since it is the object itself upon which everything else is based. Next, the method of designing an object from constraints is discussed. The design process produces not only the physical representation of an object, but also *why* it was designed the way it was and *how* it satisfies its constraints. Then various types of design modifications which may be required or desired are examined. Finally, a brief look is given at fault diagnosis for complex objects.

2 Object Representation

Central to the idea of representing the design of a complex object is the representation of the object. Indeed every other aspect of the design is at least indirectly dependent

on the representation of the object. For example, the proof that an object has certain desirable properties will likely contain a reference to the properties of a physical component in the design.

There are two alternatives for representing an object: (1) have separate conceptual and physical decompositions of an object with a mapping between them, or (2) combine the conceptual and physical decompositions into a unified representation. When the conceptual decomposition of an object matches with its physical decomposition, the cognitive complexity of the system is reduced and hence the representation of the object becomes easier to understand. Therefore we favor matching the conceptual and physical decompositions of an object. To capture the relevant design knowledge, we should be able to represent the decompositional, hierarchical, functional, structural, and physical knowledge for the object[1]. We use a frame-based scheme to capture the above design knowledge since it is capable of representing the required information and provides an easy means for manipulating this information.

Decompositional knowledge of an object is represented in a hierarchy. An object is composed of parts, each of which can be composed of subparts, and so on. A collection of subparts may be thought of as a component of the object. Relationships between parts in the hierarchy are represented by using HAS-PART and PART-OF links. An object is ultimately realized by a collection of parts which are elementary or are themselves complex objects with their own designs. A part is considered elementary if it is essentially nondecomposable. Examples of elementary objects are nails, screws, sheet metal, and glass. Physical knowledge is represented by describing elementary objects in terms of size, shape, color, composition, mass, etc. This scheme allows different objects to use the same part in their designs without having to each give the physical representation of that part. It also allows the physical design of an object to be examined in as much detail as is desired.

To take a simplified example, a lamp can be decomposed into the following parts: base, cord, switch, shade holder, and shade. The switch is an example of a complex object which has its own design. The base can be partially described as being white, ceramic, cylindrical, and 12 inches tall.

The structural knowledge of an object is represented by explicitly specifying the interconnections between the parts. This provides among other things a method of specifying the orientation of one part with respect to another and the type of connection (i.e. glue, weld, interlocking parts) between parts. The interconnections between parts are represented by links. The specification of an interconnection link is necessarily complex because of the nature of the relationship being represented. Not only must the type of connection between parts be specified, but the spatial orientation between connected parts must be specified as well. To use the lamp example, it can be specified that the bottom of the switch screws into the top of the base. Thus the physical design of an object is represented by a hierarchy of parts and their interconnections.

Functional knowledge is represented by describing an object in terms of what it

does. As mentioned earlier, every object serves some purpose. For example, the purpose of a toaster is to toast things. The exact representation required to represent functional knowledge has not yet been decided upon.

The taxonomy of objects is represented by classes and subclasses using IS-A and SUBCLASS links, respectively. A particular object is an instance of some class of objects. Thus objects which are similar can be located and compared. The collection of the object designs forms the case base. An indexing scheme is needed to be able to locate and examine objects with specific characteristics. This may be useful when designing an object since choices of already existing objects can be examined by the designer when deciding what to use for a particular part. Also, the PART-OF links allow the designer to analyze how a part was used in other designs. Using this information, the designer can decide whether an existing object will suffice or a new object must be designed. As is described later, a new object may be created by modifying the design of an existing object.

3 Designing from Constraints

The design of an object provides a consistent assignment for a set of variables that together define the desired object and satisfy the constraints placed upon that object. General classes of constraints include functionality, cost, performance, size, and weight. An example of a cost constraint is "The lamp must cost no more than 30 dollars at current prices." Obviously there are many other classes into which constraints may fall.

The constraints for an object are specified first and then a design for the object is developed to meet these constraints. There exists a gap between the constraints and the design since there is often no reference to how each constraint is satisfied in the design. Some sort of bridge is needed to link the design of an object with the set of constraints which it must satisfy. This is especially important when a new object with slightly different constraints from an existing object is desired. By identifying the parts of the existing design dependent on the altered constraints, the new design can be obtained by modifying the existing design to meet the new constraints.

As the complexity of an object grows, so do the number of constraints. A structured form of specifying constraints is needed to manage complex descriptions. The method presented here uses top-down constraint refinement and bottom-up constraint satisfaction, similar to the plausibility-driven design method described in [3]. Indeed, the latter method will be used as a basis in formally developing the former method.

Initially, very general constraints are given for an object. These constraints, referred to as *top-level* constraints, generally include properties of the object as a whole. Each top-level constraint is then refined into more specific constraints. It must be shown that satisfying all of these more specific constraints will cause the top-level constraints to be satisfied. Thus if constraint A is refined into constraints X, Y, and

Z, it must be shown that A is satisfied if X, Y, and Z are all satisfied. Each of the more specific constraints are then refined into even more specific constraints in the same fashion. The refinement process continues until sufficiently precise constraints are reached. These constraints are referred to as *elementary* constraints. An elementary constraint corresponds to a feature which must be present in the design of the object. An example of an elementary constraint is “The lamp switch must have the capacity for a 100 watt light bulb”. An elementary constraint is satisfied if the design contains that particular feature. Because of the nature of the constraint refinement, the top-level constraints are satisfied if all of the elementary constraints are satisfied.

Once the constraints for an object have been specified, the design of the object is then developed to meet these constraints. The designer specifies the design of the object as a hierarchical decomposition of parts and their interconnections. In order to prove that the object designed satisfies its constraints, it is necessary to show how each elementary constraint is satisfied in the design. These relationships are also needed to indicate the constraints with which a particular component of the design is involved. This allows modifications to be made to the design without violating constraints which were already satisfied.

Many of the major decisions made while designing an object regard determining how the object should be decomposed to form its design. Some decisions concerning the structure of the object may be made when the constraints for the object are specified, however most decomposition decisions will probably be left to be made when the design of the object is created. The physical decomposition of a similar previously defined object may be used as a guide to decompose the object currently being designed. The designer may instead choose to use selected components of the physical decompositions of other objects in places in the current design. It is also possible for the designer to incorporate innovative ideas into the current design. Thus the designer has two sources of knowledge regarding how an object can be decomposed – previous designs and innovative thinking.

It is therefore important for a designer to be able to access and examine the designs of existing objects to help in making design decisions. However, this may still leave a critical gap in the information needed by a designer. This gap is caused by the fact that while a design describes the physical structure of an object, the rationale behind the design is often not represented. Capturing the knowledge involved in designing an object is of great importance in many areas, especially where the lifetime of the project is expected to exceed the time of involvement of the designers, as in [4]. Thus it is crucial to be able to retain not only the design of the object but also the reasoning behind the design so that it can be referenced in the future. This problem can be solved by including the decisions made while designing an object with the actual design of the object. By having the rationale behind previous designs as well as the designs themselves, a designer can make more educated decisions regarding the object being designed.

When the design of an object is being created, there are usually many ways in which various constraints can be satisfied. Thus at any given point in the design the

designer may have a number of alternatives in choosing which part to use or how a particular component should be decomposed. When a decision is reached, it is important for the designer to document the object design with the justification for that decision. This justification should include the choices considered, the reason why the decision was made as it was, and reasons why other choices were not selected.

For example, while designing a lamp, a designer needs to decide on the switch to use given the constraints “The lamp has a two position on-off switch” and “The lamp switch must have the capacity for a 100 watt light bulb”. By examining various types of light switches available, the designer narrows the choice down to three switches – a cheap two position sliding bar switch, a slightly more expensive two position sliding bar switch, and a two position rotating switch. The designer is then informed that for a two position switch, the sliding bar type is preferred over the rotating type of switch. The third switch is ruled out because of this fact. The designer then chooses the first switch citing the fact that it is less expensive than the second switch. However after tests show that the second switch is considerably more durable than the first switch, the designer chooses to use the second switch.

The justification of the decision to choose the second switch is stated as follows: (1) In study X, it was found that the sliding bar type of switch is the preferred type for two position switches. The rotating switch is ruled out because of this fact. (2) The cheap sliding bar switch is chosen initially because of its cost. (3) Test Y showed that the slightly more expensive sliding bar switch considerably outlasted the cheap sliding bar switch. The former switch was then chosen replacing the latter switch because of its quality.

In this way the evolution of an object including the rationale behind the design can be captured in an integrated form which can then be analyzed and/or critiqued by others. With the justifications, the design can be more easily understood by persons not involved in designing the object.

It is important to note that objects which are not designed in the manner outlined here can still have their designs represented. The design would only consist of the physical design of the object as described in the previous section along with whatever constraints and design decisions are available. This information could be obtained from the available design documentation and supplemented by interviewing the designers involved in the project. Without this, all previously designed objects could not be represented, thereby rendering this entire scheme useless.

4 Design Modification

Some time after the design of an object is completed and judged to satisfy all of its constraints, there may be a need to make modifications to the design. Modifications may be necessary because a problem arose with the original design, some constraints were left out of the original design, an improvement can be made, or an enhancement

to the original design is desired.

Through the constraint satisfaction method every effort is made to insure that the design developed satisfies the constraints imposed on the object. In developing the design of an object, the designer will likely use parts which have been previously designed and whose functionality matches what is needed in the design being developed. If it is discovered that the functionality of a part does not meet what was claimed, a problem may arise in objects which use the part. If such a problem does occur then the design of objects which use the part must be modified in order to satisfy the constraints which were violated as a result of the part not performing as expected.

It may be the case that some constraints were left out of the original design. The constraints may have been either overlooked or not thought to be important. As a result, the design of the object turned out to not quite match what was desired. Thus the constraints omitted are added and the object is redesigned to incorporate the new constraints.

The design of an object can be modified to reflect an improvement in some area(s) of the design. For example, a new part may become available which can replace a part in the original design and is cheaper, faster, smaller, or more reliable than the part originally used. Thus if an inexpensive switch which never wears out is developed, it can be substituted for the switch currently used in the lamp design. An improvement to the design can be accomplished by changing the constraints referencing the old part to reference the new, improved part. The new part must still satisfy those constraints. The reference to the old part is retained in order to reflect the history of the design.

Modifications can also be made to enhance the functionality of a design. For example, the simple on-off switch on the lamp may be replaced by a three-way switch to produce a more versatile lamp. Constraints indicating the enhancement are added where appropriate and the design of the object is altered to satisfy the new constraints. It must be insured that previously satisfied constraints remain satisfied after the design modification is made.

The difference between these last two types of modifications is that an improvement makes a design better without changing its basic functionality while an enhancement extends the functionality of an object (but does not necessarily make it better).

As is the case in the process of creating the original design, justifications of design modifications should be included in the design. This is necessary to maintain the complete history of an object design.

Modifying a design need not always replace an old design with a new one. Instead, the old design must be allowed to exist as an object as long as it is still useful. This is especially the case regarding enhancements to a design. The old and new designs may remain interconnected if desired so that a change in one effects a change in the other. On the other hand, the old and new designs can be made totally independent of one another if significant changes are made to the old design.

5 Fault Diagnosis

A common characteristic of every complex object is functionality – each complex object serves some purpose. For example, the function of a toaster is to toast things and the purpose of a lamp is to provide light. The design of an object is such that the functionality desired is achieved. However it may be the case after some period of time that the object does not function properly, i.e. the toaster does not toast or the lamp does not provide light. In other words, there is a fault associated with the object. The fault must be diagnosed in order to correct the problem. A fault can be diagnosed using either deep level reasoning[8] or shallow level reasoning.

One method of diagnosing a problem is by using deep level reasoning to analyze the design of the object and determine the cause of the malfunction. Thus by knowing that the lamp does not provide light, analyzing the design of the lamp will discover that the fault is caused by a burnt bulb, a broken switch, or a bad cord. Note that the function of the light bulb must be included in the design of the lamp since it is the bulb which actually produces the light. This method by itself may have difficulty dealing with nontrivial malfunctions where the failure of one part leads to failure of others. In any case, it is desirable to avoid having to analyze the entire structure of an object every time a malfunction occurs.

Another approach to problem diagnosis is to use case-based reasoning to find similar malfunctions which have occurred previously. This approach uses shallow level reasoning since it does not try to reason from the physical design of an object, only from past experiences with malfunctions. If a similar malfunction has already occurred with that object, the corrections used to eliminate the previous malfunctions are examined. If a successful correction is found and the circumstances are similar enough, the correction is tried on the current problem. If the circumstances are not quite the same, the correction may have to be adapted to apply to the current problem. The success or failure of the correction is stored along with the circumstances under which the correction was applied. In the case of a failure, an explanation of why the correction failed (if known) is also stored.

If no cases of malfunctions with similar circumstances are found for the object, similar objects can be examined. If possible, analogical reasoning could then be used to adapt the corrections applied to one object to apply to the current object. If no past experience can be used in the current situation, a diagnosis by an expert or one based on the design of the object (using deep level reasoning) must be formulated. As time goes by and varied types of malfunctions occur, the case base grows and the fault diagnosis capability for an object improves.

6 Conclusion

This paper introduces a design scheme which integrates the process of designing complex objects within a framework that allows for the capture of the design knowledge

that went into the design. The framework is intended to be sufficiently general so that any object can be represented. The presence of a broad domain model eliminates most of the redundancy and wasted effort caused by the inability to integrate rigidly defined domains.

Case-based reasoning is used to provide designers with knowledge of parts, past designs, and the rationale behind these designs to assist in the design process. CBR is also used to help diagnose problems which occur in an object while it is in use.

Future work will be focused mainly on formalizing many of the ideas presented in this paper. The process of designing objects will be investigated further to provide more insight into what is required to fully capture the knowledge utilized when designing an object.

References

- [1] Addanki, Sanjaya and Ernest Davis. A representation for complex physical domains. In *Proceedings of the Ninth IJCAI*, 1985.
- [2] Agüero, Ulises. *A theory of plausibility for computer architecture designs*. Ph.D. dissertation, Center for Advanced Computer Studies, Univ. of Southwestern Louisiana, 1987.
- [3] Agüero, Ulises and Subrata Dasgupta. A plausibility-driven approach to computer architecture design. *Communications of the ACM*, 30(11):922-932, November 1987.
- [4] Freeman, Michael S. The elements of design knowledge capture. In *Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications*, November 1988.
- [5] Kolodner, Janet L. Extending problem solving capabilities through case-based inference. In *Proceedings of the 4th Annual International Machine Learning Workshop*, 1987.
- [6] Mostow, Jack. Toward better models of the design process. *AI Magazine*, 6(1):44-57, 1985.
- [7] Schank, Roger C. *Dynamic Memory - A Theory of Reminding and Learning in Computers and People*. Cambridge University Press, Cambridge, England, 1982, Chapter 2.
- [8] Sembugamoorthy, V. and B. Chandrasekaran. Functional representation of devices and compilation of diagnostic problem-solving systems. In Janet L. Kolodner and Christopher K. Riesbeck, editors, *Experience, Memory, and Reasoning*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1986, Chapter 4.

- [9] Stanfill, Craig and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, December 1986.
- [10] Steele, Robin L. Cell-based VLSI design advice using default reasoning. In *Proceedings of the Rocky Mountain Conference on AI*, 1988.
- [11] Steinberg, Louis I. Design as refinement plus constraint propagation: the VEXED experience. In *Proceedings of the Sixth AAAI*, 1987.

An Application of Design Knowledge Captured from Multiple Sources

Preston A. Cox
John H. Forbes
Lockheed Missiles and Space Company
Org. 62-81, Bldg 579
1111 Lockheed Way
Sunnyvale, California 94089-3504

Abstract

The Hubble Space Telescope Operational Readiness Expert Safemode Investigation System (HSTORESIS) is a reusable knowledge base shell used to demonstrate the integration and application of design knowledge captured from multiple technical domains. The design of HSTORESIS is based on a partitioning of knowledge to maximize the potential for reuse of certain types of knowledge.

Introduction

The Hubble Space Telescope Operational Readiness Expert Safemode Investigation System (HSTORESIS) is a knowledge based system which demonstrates the integration and application of design knowledge captured from multiple technical domains. The domains of interest are the electrical power and pointing control systems of the Hubble Space Telescope (HST). The types of design and engineering knowledge contained in HSTORESIS pertain to the analysis and resolution of system anomalies known as safemode events.

HSTORESIS is motivated by the HST Design/Engineering Knowledge-base (HSTDEK) project. The primary goals of the HSTDEK project are to enable major NASA projects to capture design and engineering expertise and to support the use of the captured knowledge in multiple applications [2]. HSTORESIS addresses these goals by providing a reusable knowledge base shell which can access a variety of device models and rule bases to allow a user to solve a variety of problems.

The following sections discuss some key technical issues addressed in HSTORESIS and describe major features of HSTORESIS.

Knowledge Partitioning

It has been said that one important approach to managing the computational cost of causal reasoning is structural abstraction [3]. In this spirit, the design of HSTORESIS is based on a

partitioning of the knowledge typically contained in a knowledge based system. The knowledge partitioning helps manage the computational cost inherent in rule based systems and increases the opportunities for knowledge reuse.

It is not practical or possible to procedurally define all of the behaviors of a complex device even though, from an engineering perspective, each subcomponent may be precisely defined. In order to reason about a device as complex as the HST, a system must include both procedural or algorithmic knowledge and heuristic or partial knowledge. Traditionally, applications using the production system approach tend to merge both types of knowledge into one rule base.

Merging procedural and heuristic knowledge contributes to system brittleness and reduces the opportunity for knowledge reuse. To overcome these problems, HSTORESIS provides the hooks for the knowledge engineer to partition a knowledge base into procedurally oriented device models and heuristically oriented production system rules. This knowledge base partitioning allows more than one set of heuristics, or production rules, to be applied to an HST component or subsystem. This increases the potential for knowledge reuse.

For simple systems, the encoding of the procedural knowledge in device models is often sufficient to describe the system's behavior. However, because of its complexity, the HST is not a fully described system. Some of its behaviors cannot be procedurally described. For example, a design engineer might know from experience that if a reaction wheel spins at 2,200 rpm for more than two minutes the rotor bearings will experience excessive wear. The engineer might therefore recommend that maneuvers be avoided that would cause the reaction wheel to over spin for more than one and a half minutes.

The important distinction is that heuristic knowledge is only approximate and is subject to different interpretations in different situations. For example, how much bearing wear can be tolerated might depend on the importance of making a particular maneuver or the nearness in time to a service interval. In contrast, the calculation of angular momentum or the mass of the reaction wheel is a fixed characteristic of the device.

The reason for making this distinction is that the procedural knowledge has a greater potential for reuse. This reuse can be achieved in two ways:

- By having more than one set of heuristic rules reason over

- the same device model, and
- By combining device models to create new models. (This type of reuse, composite models, is possible because both single and multiple inheritance are supported by the object oriented programming tool used by HSTORESIS.)

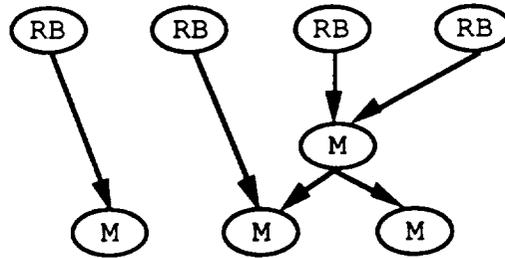


Figure 1. Example of Reuse in HSTORESIS

Figure 1 illustrates both types of reuse. In one case, two rule bases (the bubbles labelled RB) reason over one model. The model itself is a composite of two other models.

In contrast to the procedural knowledge, the heuristic knowledge is subject to more change over time. For this reason, the proposed partitioning will make knowledge bases easier for the knowledge engineer to maintain and modify.

Reusable Interface

An analyst interacts with the knowledge contained in HSTORESIS through the Interface Management System. The basic script that the analyst follows to define a problem for analysis is suggested by Figure 2. The analyst selects a device model from a menu of all models known to HSTORESIS. The analyst also provides a time interval over which the device model is to be analyzed.

The Interface Management System retrieves from the selected device model a list of associated engineering data. A query is then made of an external source to retrieve values for the engineering data for the time interval selected by the user.

The final part of the problem description provided by the user is the rule base to be applied to reason about the device model. Each model knows what rule bases are associated with it, and the list of associated rule bases is provided to the Interface Management System for presentation to the user via a menu.

Three major design criteria implemented by the Interface Management System are to:

- provide a point-and-click style of interface that minimizes use of the keyboard and maximizes use of the mouse,
- provide a set of reusable display objects that give a consistent look and feel across applications, and
- establish a protocol that application developers may follow to access and use the display objects.

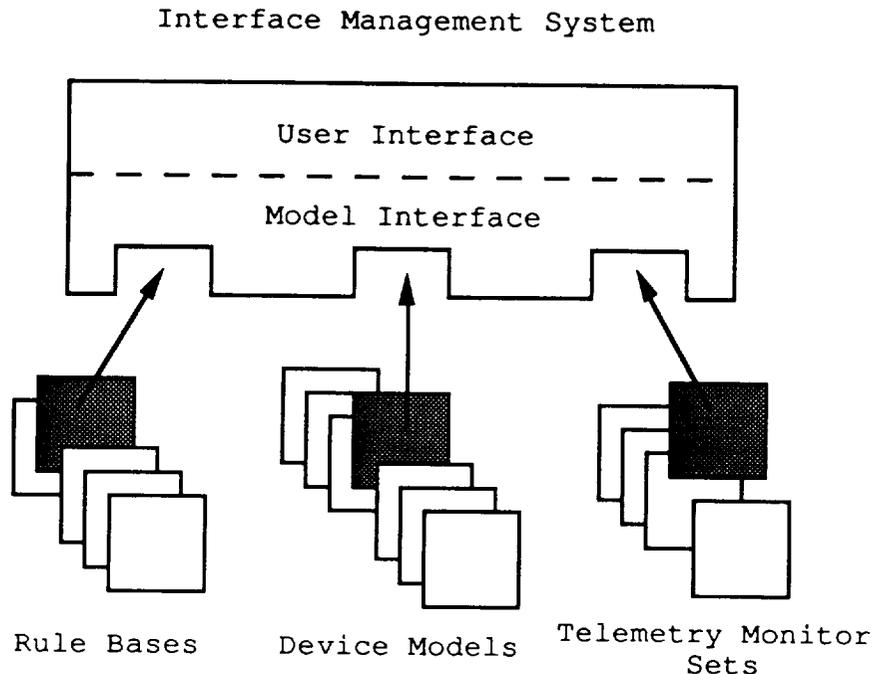


Figure 2. The HSTORESIS Concept

Examples of the types of display objects that are available include buttons, query panels, menus, telemetry monitor display panels, message windows, and time displays. Graphical images are also included. For example, one graphical image depicts the orientation and location of the HST relative to the earth, moon, and sun. More will be said later about the protocol available to application developers for using display objects.

HSTORESIS implements a display page library. A display page consists of a collection of display objects accessible by the user. Display pages may be built by a user and saved in a display page library. Display pages are indexed by user name and problem type. A user may build, modify, and save any display pages owned by the user. Display pages owned by other users may not be changed, although pages belonging to other users may be copied into the current user's work space and then modified, if desired.

The complete set of display objects provide a powerful tool

for use by the application developer. The objects are implemented using object oriented programming techniques and can be accessed via a simple messaging scheme.

The Interface Management System manages the creation and display of the objects, the collection of answers from the user, and the return of the user inputs to the messaging object. Two important benefits derive from this. First, the amount of interface programming that an application developer must do is significantly reduced. Second, the reuse of the set of display objects provides a consistent look-and-feel for the user across problem solving sessions.

Reusable Telemetry Database

The source of information about the behavior of the HST is engineering data obtained from monitors on board the HST. Data is collected and communicated via telemetry to a ground station. There are approximately 5,500 telemetry monitors associated with the HST. The interesting technical issues concerning the monitors are how to represent them in a knowledge base and how to obtain descriptions of the monitors associated with a device model.

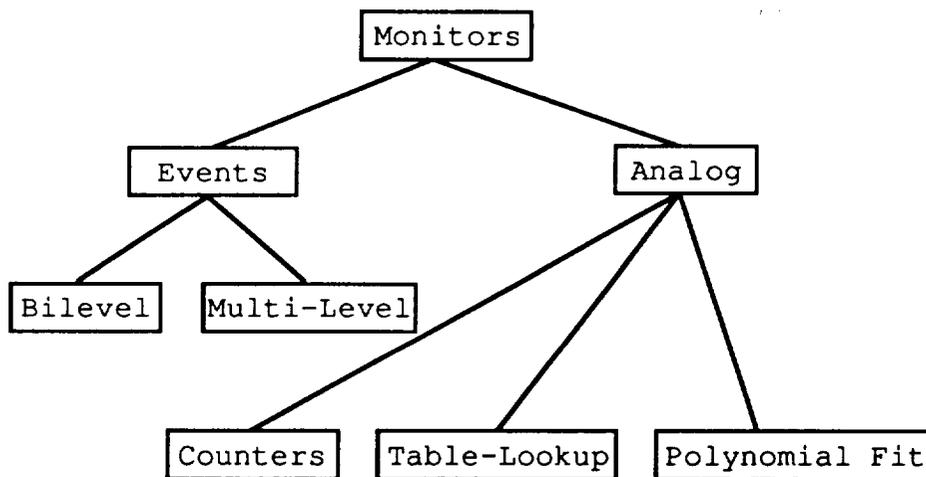


Figure 3. Monitor Classes

The solution to the representation issue is illustrated by the design in Figure 3. All 5,500 telemetry monitors conform with the design.

Event monitors have values that are either bi-level or multi-level. Bi-level monitors have measurements with only two states (e.g., on or off). Multi-level monitors have measurements with more than two states (e.g., high, medium, or low).

Analog monitors are either counters, table look-ups, or

polynomials. Counters represent uninterpreted telemetry counts. If a counter has a value of three, then three is the correct meaning of the value. A table look-up requires retrieval from a table of the analog value corresponding to the monitor value. For example, a monitor value of three might correspond to an analog value of 1.67 volts. A polynomial fit is a monitor whose value is inserted into a polynomial equation. The meaning of the monitor's value is the solution of the polynomial equation. For example, if a polynomial equation for a monitor has the coefficients of 1.5, 0.03, and 2.1, then a monitor value of 3 has the meaning:

$$1.5*3^0 + 0.03*3^1 + 2.1*3^2 = 20.49.$$

Given the design, the problem of extracting the descriptions of the desired telemetry monitors becomes simply a matter of generating a list of the desired monitors, locating them in a master database of monitor descriptions, and then creating a knowledge base from the descriptions of the monitors. All of this is automated by HSTORESIS which eliminates the need for manually producing the descriptions.

Reusable Device Models

Within HSTORESIS, a satellite telemetry point is represented as an object with its own data and set of behaviors. In one sense, the instantaneous state of the HST is represented by the collective output of its 5,500 telemetry monitors. However, this representation is extremely weak since it lacks information about component connectivity and component behavior. Although rules can be written that reason exclusively in terms of telemetry values, human experts do not usually think in these terms.

By extending the above analogy one step further, the state of each major component of the HST is represented by the values of some set of monitors. The mapping between a set of monitors and a component forms the nucleus of device models used in HSTORESIS.

The monitor mappings, however, are only part of the model abstraction. A complete device model will include all of the following:

- a mapping between the model and a set of monitors,
- pointers to the rule bases that are capable of reasoning about the device,
- behaviors (methods) that represent the conceptual or physical functioning of the device/component, and
- features (slots) that hold state information that is not included in the satellite telemetry stream.

The first two items are used dynamically to establish bindings between the HST telemetry database, the HSTORESIS device model, and the user interface. The final two items encode the procedural knowledge that relates to the device. For example, if a reaction wheel were being modelled, an example of the final two items might be a method for computing angular momentum, or features like the wheel's mass or composition.

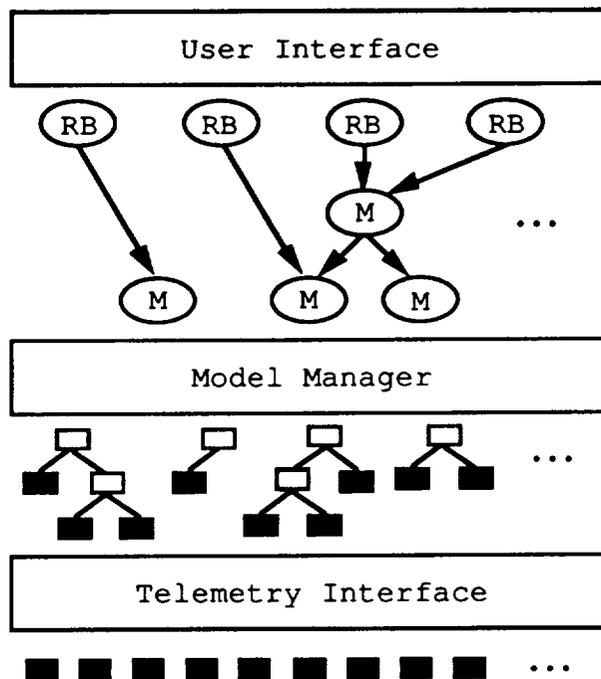


Figure 4. HSTORESIS Architecture

Figure 4 illustrates how these ideas have been incorporated into HSTORESIS to transform raw telemetry data into a level of abstraction which is closer to the mental representations that human experts use. The telemetry interface stands between the raw telemetry data and the telemetry object abstraction. The model manager lies between the telemetry objects and device models (indicated by bubbles labelled with an M). The user interface layer provides the user with direct access to the rule systems that reason over device models.

Graphical Object and Rule Integration

An important feature of the Interface Management System, mentioned previously, is the protocol for accessing graphical objects. The protocol supports development of rules that can both deliver information to and solicit information from the user. The use of graphical objects can be described by referring to Figure 5, which depicts a schematic of the screen layout for HSTORESIS.

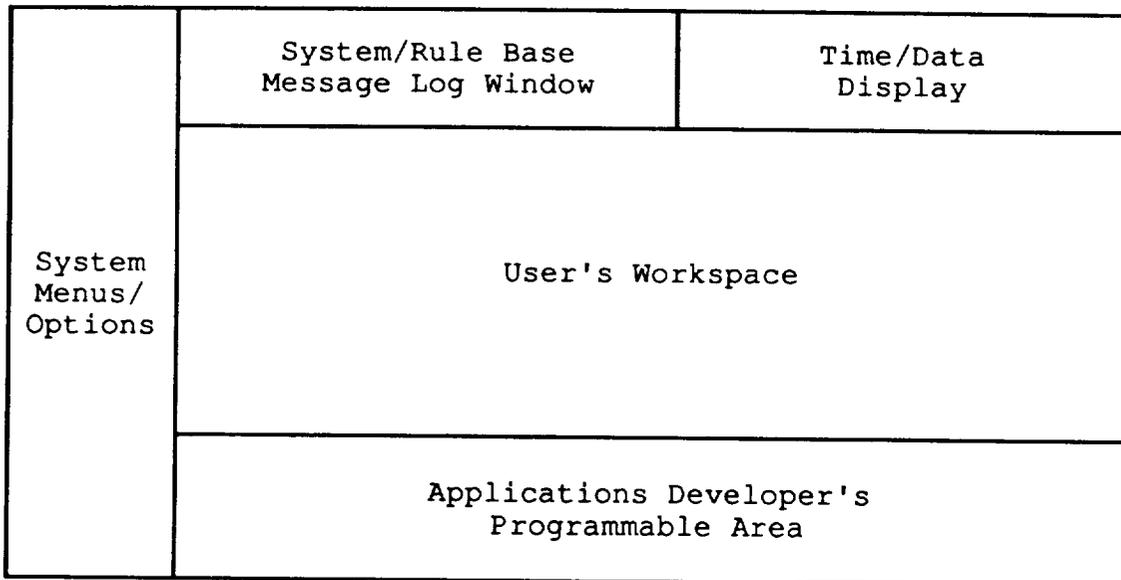


Figure 5. HSTORESIS Screen Layout

The System Menus/Options area permits the user to access application independent menus and options for defining a problem set, editing the workspace, and launching and managing a query. Queries that can be made by a user are determined by the rule base selected by the user to be applied to a set of telemetry data. User queries are controlled by start, pause, and step options. The basic visual metaphor for interacting with this area is a button.

The Application Developer's Programmable Area is reserved for the display of application specific options. The application developer is provided with a simple protocol for programmatically displaying and removing options that are application specific. Again, the basic visual metaphor for user interaction with this area is a button. An application developer may choose to provide buttons in this area that correspond to a set of queries that a rule base can answer about the telemetry data to be analyzed.

The System/Rule Base Message Log Window area is reserved for the display of system messages (e.g., warnings) and the results of inferencing. The Log Window provides the user with icons for controlling which message is displayed. Messages may be scrolled through the window, or a specific message may be selected for display. A counter indicates how many messages are available and the number of the message which is currently displayed.

The Time/Data Display depicts three items. The spacecraft time is indicated. The frame of the telemetry data that is

currently being analyzed is indicated. A slide bar indicating the percentage of telemetry data that has been (or remains to be) analyzed is also provided.

The User's Workspace allows the user access to display objects that can be used to monitor the state of the device model as the analysis session progresses. Some of the objects that may appear in this area are monitor display objects, graphical images such as the one used to depict the relationship of the HST to the sun, earth, and moon, and pop-up objects such as query panels. Most of the interaction between the user and HSTORESIS will occur in this area.

Pop up query panels permit rules or other objects to ask questions of the user. Query panels are also used to obtain information from the user at the start of the analysis session. The user must provide a user name, a device model, and a time interval over which the analysis is to be made.

Pop up dialog boxes permit rules to provide information to a user. Optional buttons may be associated with a dialog box to provide additional capabilities for the user. Figure 6, for example, depicts a rule that creates a pop up dialog box with a button labelled "Recovery" which permits access to information about recovery from an event called "RWA-Speed-Limit-Test".

```
(IF
  (TEXT (RWA-SPEED-LIMIT-TEST HAS FIRED))
  THEN
  (LISP
    (UNITMSG '(SIS-SCREEN-MANAGER SIS-SCREEN-MANAGER-KB) 'IN-BOX
      'DISPLAY-POP-UP-MESSAGE :TEXT
      "RWA Speed Limit Failure: check for:
~%Too large vehicle maneuver
~%momentum management performance
~%misconfigured software
~%other database problems."
      :BUTTON-VALUES
      '(("Recovery"
        '(UNITMSG '(SCREEN-MANAGER SCREEN-MANAGER-KB) 'IN-BOX
          'DISPLAY-POP-UP-MESSAGE :TEXT
          "1. Monitor wheel speed in sun point until
            it returns to normal.
~%2. Dump memory to re-verify the database
~%3. Work through Section 8.0 recovery
            procedure."))))))
```

Figure 6. Sample Rule

A more interesting use of pop up dialog boxes is to provide a nonlinear text or hypertext functionality. For example, the rule

in Figure 6 could be modified to provide a button connected to a function that could access the "Section 8.0 recovery procedure" in the appropriate design document. Then, reviewing the recovery procedure would be as simple as clicking on a button. An obvious extension to this capability is providing access to a video disc containing schematic drawings or other graphical images pertinent to the analysis being performed.

Conclusions

HSTORESIS demonstrates a successful approach to integrating knowledge from multiple domain experts into a single knowledge base system. An adaptive, knowledge-based interface facilitates interaction between a user and domain specific rule and knowledge bases. The application demonstrated by HSTORESIS is analysis of safemode events, which is diagnosis problem. However, HSTORESIS could easily be extended to other applications such as training, scheduling, design, etc. Additionally, HSTORESIS provides a capability for accessing on-line design documents in a nonlinear manner. This allows the user to access design knowledge not specifically contained in the HSTORESIS knowledge bases.

Acknowledgement

The work described in this paper has been performed under the NASA Hubble Space Telescope Contract number NAS8-32697, Modification 695.

References

- 1 "Systems Autonomy Technology Program (SATP) Plan," (FY89 draft), NASA/Ames Research Center, p. A-160, August 1988.
- 2 Freeman, M. S. "The Elements of Design Knowledge Capture." Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications, pp. 39-46, November 1988.
- 3 Genesereth, M. R. "The use of design descriptions in automated diagnosis." Artificial Intelligence, 24:411-436, 1984.

THE INTEGRATION OF AUTOMATED KNOWLEDGE ACQUISITION
WITH COMPUTER-AIDED SOFTWARE ENGINEERING
FOR SPACE SHUTTLE EXPERT SYSTEMS

Dr. Kenneth L. Modesitt
Head, Department of Computer Science
Western Kentucky University
Bowling Green, KY 42101

ABSTRACT: The phrase "expert systems" will disappear within ten years. Somewhat less likely to suffer the same fate will be the term "knowledge acquisition." The field of software engineering will expand to include both of these terms, wherein expert systems will be a form of advanced software engineering. The incorporation will permit more complex domains to be addressed, and the unique qualities of expert systems will render the resulting software more transparent. System specification and requirements analysis will be augmented by knowledge acquisition techniques to enable prototypes to appear earlier for customer inspection, with the end result being a software product for which the customer has a real need, and which performs up to her expectations (Ref. 2)

The current qualities of expert systems will become embodied in various components of software engineering methodologies and end products. The most likely candidate for this process in Computer Aided Software Engineering (CASE) tools. For once, we in the software engineering world will not have to continue to be the shoemaker's children. We have constructed powerful, useful, and extensible automated tools for our own use, rather than only building them for others. The features of expert systems will be used in most parts of CASE, including needs assessment, requirements analysis, design, implementation, testing, and maintenance and enhancement. Project planning, documentation and software quality assurance will also benefit. The growing interest in "reverse" software engineering, of going from existing ill-structured and non-documented code to modular design representations, will be a ripe field for expert system contributions. The critical nature of user interfaces will be addressed by our expertise in transparency and explanation-based learning of expert systems.

Many of the above "predictions" are not really futuristic at all. They were incarnated in the process of constructing an automated test analysis computer system for the Space Shuttle Main Engine (SSME) by the Rocketdyne Division of Rockwell International (Refs. 3,4,5). The development effort was successful in bringing the system SCOTTY on-line in June, 1988 at somewhat over 25% of the eventual full system, in terms of thoroughness of the SSME test analysis procedure. Progress has continued to date, and has spawned other automated SSME systems, plus ones related to other Rocketdyne programs such as expendable launch vehicles, the engines for the National Aerospace Plane, and the Space Station power system.

This successful development was made possible by an optimal mix of vision, personnel, tools, procedures and management. The personnel included an excellent young mechanical and aerospace engineering staff, and a knowledge engineer with both industry and academic credentials. The tools were recommended by the knowledge engineer, and included both an industrial-strength inductive Expert System Building Tool, running on a multi-processor supermini computer from Concurrent Computer Corporation, as well as a PC-based CASE tool.

Management direction was given by an enthusiastic senior technical manager who was very well respected in the company, and who had realistic expectations about expert system abilities. He also ensured that the personnel and financial commitments to the program were long-term ones.

Since the knowledge engineer had a substantial background in software engineering, both as a practicing professional and as an academic since 1963, it was natural that the "front-end" of the development effort would receive considerable attention. The desirability of this front-loading has manifested itself innumerable times throughout the software industry in the savings accrued in the "back-end" of the software life cycle. Maintenance, including all three types: corrective, perfective, and adaptive, has long been recognized as the real cost driver in software.

Consequently, a great deal of attention was paid to the interactions among the expert, the user, the prototype system, and the knowledge engineer. Many alternatives were considered for this knowledge acquisition process. The recent book by Karen McGraw and Karan Harbison-Briggs (Ref. 1), with a preface by this author, would have been invaluable. Figure 1 for some knowledge acquisition alternatives is from the book.

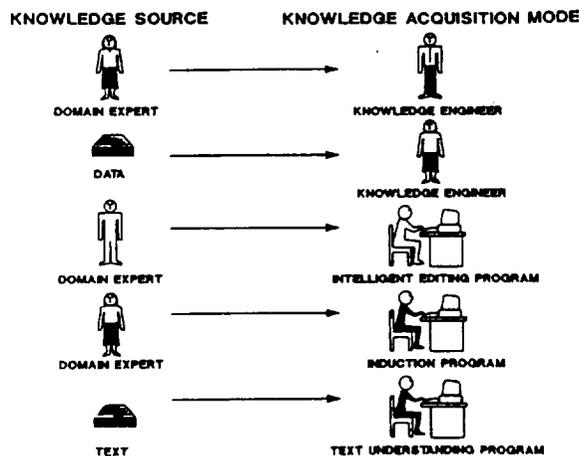


Figure 1. Variations in Possible Knowledge Acquisition Mode

(Reprinted with permission of Prentice-Hall from Knowledge Acquisition: Principles and Guidelines by Karen McGraw and Karan Harbison-Briggs, 1989.)

Initially, in 1984, a small PC-based inductive system was used to demonstrate a feasibility prototype. This took only a few days, with the expert quickly learning the tool, and appreciating the power, vs. having to codify his own rules. The order-of-magnitude increase of having the expert express his knowledge as examples, and then having the ESBT generating the rules was obvious, and has been well-documented many times since then. It was also obvious that a more powerful tool would be required for a production test analysis system. The vendor of the small tool was just ready to announce such a product, which was a near-ideal fit. In addition to induction, it also generated Fortran code, which is the lingua franca of the engineering world. This is critical as the code output by the ESBT is readable, i.e., not "magic", and it is trivial to interface it to the 100K+ lines of Fortran code which already exist for SSME software support, plus new codes which would surely be written in the future.

Good management practices and documentation guidelines dictated that all of this effort be tracked. Requirement documents were generated, as were data flow diagrams and structure charts. These were invaluable in not becoming lost as SCOTTY grew in complexity. Moreover, the tools used in CASE were easily grasped by the mechanical engineers. The fact that expert system tools were being used did not obviate the fact that it was still very much of a software engineering process, albeit in a complex domain.

In the future, it is clear that expert systems and software engineering will intertwine even more closely. A few such considerations include data bases, and automatic code generation from structure chart modules. In the case of the latter, the author and a colleague were the first to build an interface which permitted the ESBT to interact with the millions of bytes of test data from the 1000+ previous SSME tests. This interface has now been expanded by a joint venture between Intelligent Terminals Ltd. and Concurrent Computer Corporation to become a commercial product. In the case of the latter feature -- automatic code generation -- it will not be long before a CASE vendor adds inductive programming to the tool chest.

In summary, a prediction was made that the terms "expert systems" and "knowledge acquisition" would begin to disappear over the next several years. This is not because they are falling into disuse; it is rather that practitioners are realizing that they are valuable adjuncts to software engineering, in terms of problem domains addressed, user acceptance, and in development methodologies. A specific problem domain was discussed, that of constructing an automated test analysis system for the Space Shuttle Main Engine. In this domain, knowledge acquisition was part of requirements systems analysis, and was performed with the aid of a powerful inductive ESBT in conjunction with a CASE tool. The original prediction is not a very risky one -- it has already been accomplished!

References

1. McGraw, K. and K. Harbison-Briggs, Knowledge Acquisition: Principles and Guidelines, Prentice-Hall, 1989.
2. Modesitt, K., "Inductive Learning in Engineering". Tutorial for the International Special Interest Group on Inductive Programming, Detroit, MI, April, 1989.
3. Modesitt, K., "Experience with Commercial Tools Involving Induction on Large Databases for Space Shuttle Main Engine Testing," Invited talk for Fourth International Expert Systems Conference, London, England, 1988, pp. 219-229.
4. Modesitt, K., "Space Shuttle Main Engine Anomaly Data and Inductive Knowledge Based Systems: Automated Corporate Expertise," Third Conference on Artificial Intelligence for Space Applications, NASA, Huntsville, 1987, pp. 203-212.
5. Modesitt, K., "Space Shuttle Main Engine Test Analysis -- A Case Study for Inductive Knowledge-based Systems Involving Very Large Data Bases," Co-authored with Dr. Djamshid Asgari. IEEE Computer Society International Conference on Computers and Application Conference (COMPSAC), Chicago, October, 1986, pp. 65-71.

Case-Based Reasoning in Design: An Apologia

Kirt Pulaski

Martin Marietta Manned Space Systems
Post Office Box 29304 D3691
New Orleans, Louisiana 70189

Position 1: The process of generating solutions in problem solving is viewable as a design task.
Position 2: Case-based reasoning is a strong method of problem solving.
Position 3: A synergism exists between case-based reasoning and design problem solving.
This paper presents and defends these three positions.

1 Introduction

The Problem

Design issues are omnipresent in everyone's day-to-day activities. People design business deals, they design sports strategies, they design physical objects, and they design an innumerable array of other tangible and intangible things. The issues which make designing difficult are basically two-fold: the design must serve some function, and it must do so while satisfying some set of constraints.

The Paradigm

Case-Based Reasoning (CBR) is a rather unique paradigm of artificial intelligence which weaves the history of experience into problem solving. Unlike other techniques which blindly and repeatedly solve new problems from scratch, CBR consults a memory of past cases to avoid re-solving recognizable problems. The quality and efficiency of reasoning gradually increases as new cases are acquired, analyzed, and added to the CBR memory.

The Solution

Experience is the most important factor which hones design skills, enabling the mastery of a core set of principles, rules, laws and techniques. Design experience is inseparable from design expertise, therefore it seems only natural that the next generation of intelligent computer-aided design tools should utilize CBR techniques.

The Apologia

The remaining sections of this paper build a case for case-based reasoning. The domain of discussion is CBR in engineering design. The goal is to get the research community involved in some issues and to get practitioners involved in some applications. By accelerating the inevitable (i.e., the use of CBR in design), beneficial applications for space technology can be realized sooner rather than later.

Section 2 describes a hypothetical session with an intelligent CBR design tool. The section whets the appetite and motivates the need for such a system. Section 3 details our current view of the engineering design process, without intelligent computers. Section 4 presents our proposed view of the same process, with intelligent computers.

Section 5 introduces the fetus of our research: the functional model of the CBR Designer's Assistant. Section 6 describes each module in more detail. Section 7 compares our model with other research projects and gives a compendium of work in the field. Section 8 concludes with some discussion and recommendations.

2 The Scenario

To help set the stage, a typical interactive session between a designer and an intelligent computer will be presented. Let us call the system the CBR Designer's Assistant.

A design task is defined for a part belonging to a family of parts. The design requirements and constraints are presented to the designer, who comes to the design workstation and enters the CBR Designer's Assistant environment.

The designer requests assistance in performing a new design. A search is initiated in the CBR memory for previous cases of designs with the same or similar requirements and constraints. The system opens a new case, begins recording the design session, and prepares itself to track/guide the designer's goals throughout the session.

The designer selects a part from the list of retrieved cases and it is presented on the CAD screen. The designer browses the features, design decisions, and other knowledge associated with the past case. The designer selects and browses other similar designs (if there are any) until the closest match is found. If no relevant cases are found, the designer inputs features-based parametric commands to generate a new seed design.

The designer takes actions (makes decisions) within the system such as changing feature parameters, deleting features, adding features, editing relations, and so on. As this activity is performed, the system monitors and attempts to explain the actions (rationale) based upon its expectations. If needed, the system makes suggestions to the designer, guiding the design process.

The system activates a special design knowledge capture facility to prompt the designer to assist in the construction of explanations for actions which the system cannot self-explain. The facility associates the explanation information with the current case which will eventually be stored in memory. All aborted designs and dead ends are also captured and noted to enable the avoidance of failures in the future (through reminding).

The design evolves by specification/modification of features and parameters, which leads to more remindings and brings to bear pertinent lessons learned from the past. The embedded knowledge in the remindings enforces considerations related to form, functionality, production cost, materials, producibility, inspectability, and so on.

When the designer indicates to the system that the current preliminary design is satisfactory the resultant case is finalized and stored into the CBR memory.

The CBR Designer's Assistant is then prepared to digitally output/convert the information for the completed design case, at the designer's request, to downstream product definition systems. If the designer is not finished but wants to stop work, the system closes the design session and stores the unsatisfied goals with the case in memory. The designer can then resume the session at a later time and/or get progress updates.

3 Without Machine Intelligence

Figure 1 illustrates our functional model of the design process. In both Figure 1 and Figure 2 the heavy arrows represent the main input/output flow of producing a design. The light arrows represent the input/output growth of design experience. The lines without arrows represent mechanisms and controls that support the design process. The shaded regions represent the background enhancements that concurrent engineering concepts provide. The rounded boxes indicate non-computer activities, square boxes indicate computer-assisted activities.

The design process begins (cf. Figure 1) with the definition of the design task. This definition consists of requirements and constraints which are input to the designer. The designer combines creativity and innovation with an experiential design history to generate a new preliminary design. The design history is used to retrieve lessons learned and other relevant knowledge to increase the quality of the preliminary design.

Two mechanisms are then applied iteratively to evolve the design into a preliminary product definition. Problem solution tools are applied to specify and analyze solution concepts (e.g., geometric modeling, "what-if" parameter manipulation, rules and principles, and performance analyses). Design process management is applied to communicate decisions and reasoning (e.g., design plan, goal tracking, design reviews, and recording of design decision rationale).

The resulting preliminary product definition is then used to update the experiential design history. This ensures that future designs can benefit from the growth of design experience and lessons learned.

The process of design must blend many considerations and expertise from multiple disciplines (e.g., engineering, production, cost estimation, and quality). Many design groups choose to utilize a concurrent engineering approach, bringing scrutiny from other disciplines into the evolving preliminary design process. This concurrent approach greatly increases the efficiency of the design process. It saves time and money which is normally spent in implementing post-preliminary design changes (inherent in the non-concurrent approach).

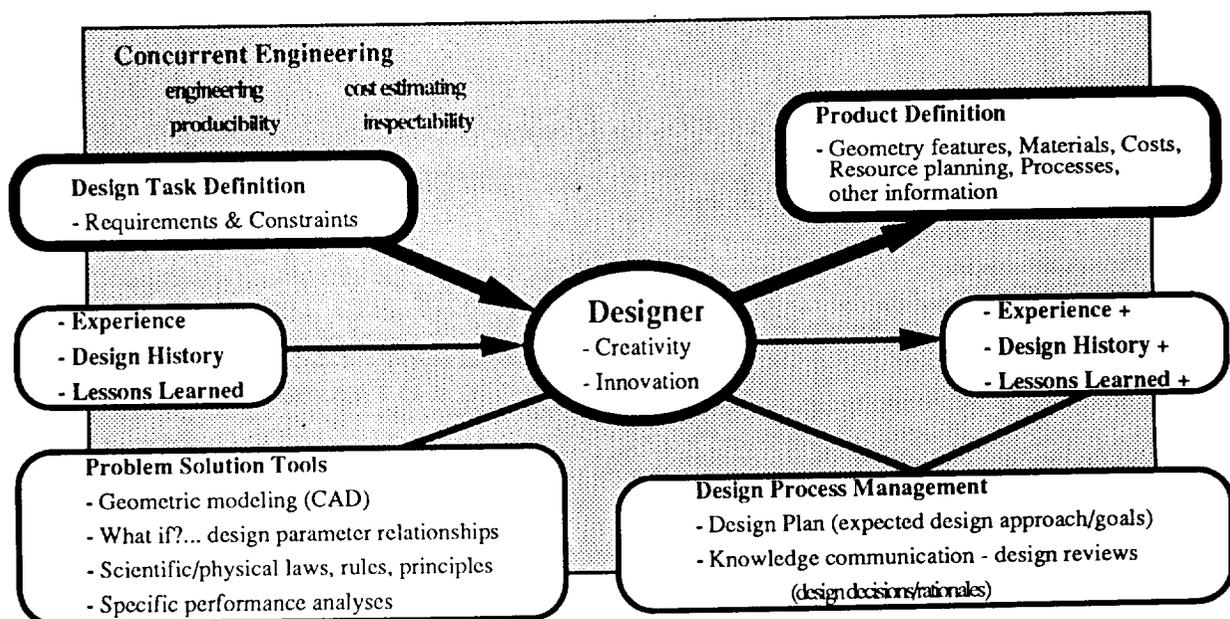


Figure 1: Without Machine Intelligence

4 With Machine Intelligence

Figure 2 shows an improved view of the Figure 1 model where some parts of the process are computer-assisted. The position of this paper is to promote the inclusion of CBR and other machine intelligence techniques when implementing this model.

The new model maps the design process depicted in Figure 1 onto a computer architecture as shown in Figure 2. The hashed line surrounding the square boxes defines the boundaries of the CBR Designer's Assistant. Creativity and innovation is still left to the designer, only now the computer allows the designer to focus more on these and less on the other mechanisms.

Figure 2 shows that the growing experiential design history is implemented as a CBR Module. This module also contains concurrent engineering knowledge. The problem solution tools are implemented as an Intelligent Computer-Aided Design (IntCAD) Module, also containing concurrent engineering knowledge. The design process management is implemented as a Design Session Manager (DSM) Module.

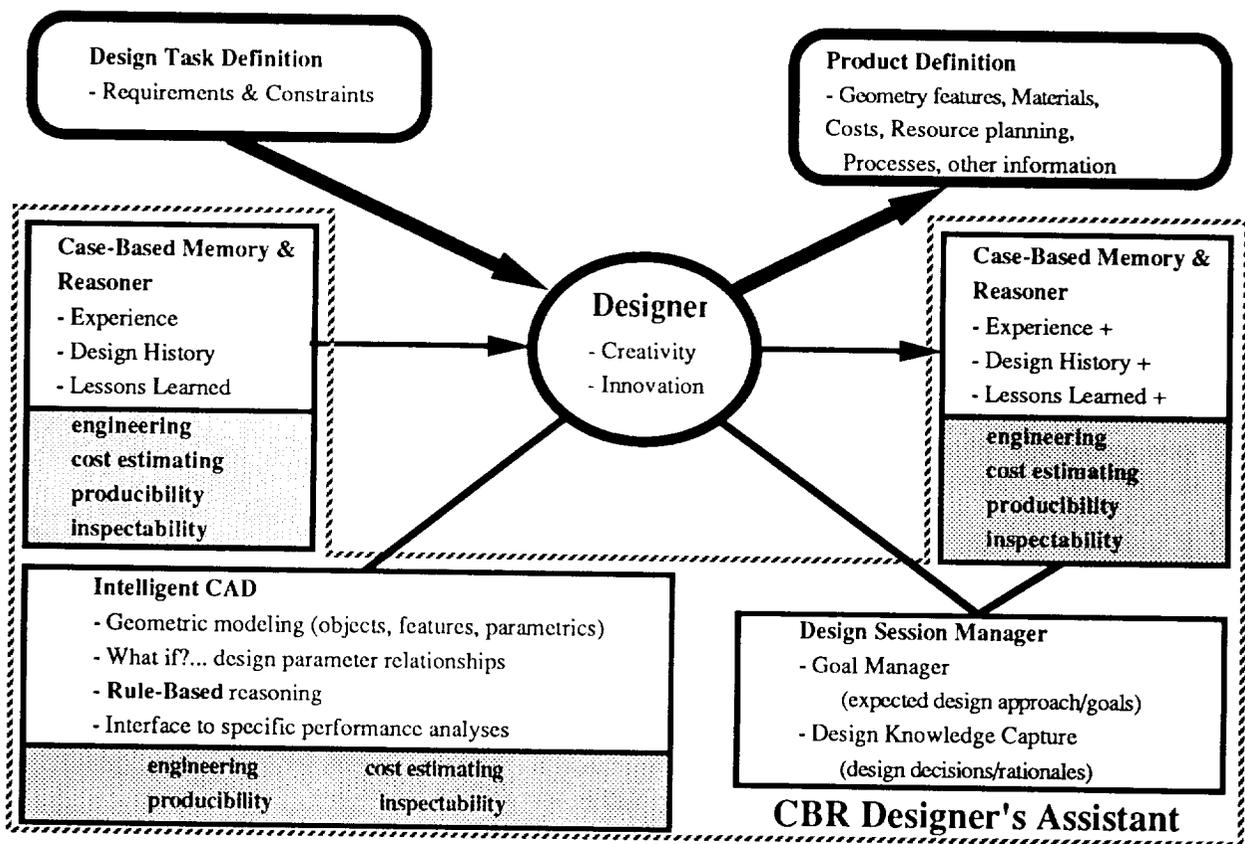


Figure 2: With Machine Intelligence

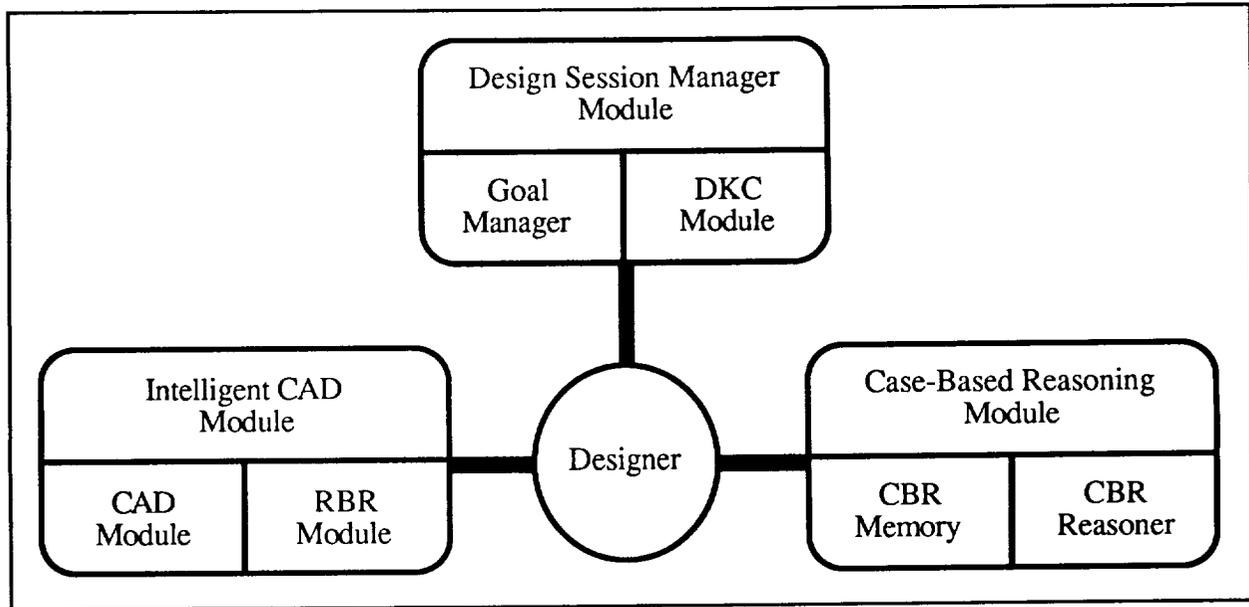


Figure 3: Case-Based Reasoning Designer's Assistant

5 The CBR Designer's Assistant

Figure 3 shows the architecture of the CBR Designer's Assistant. This section introduces the machine intelligence techniques which are used to provide the required functionality of each module.

The CBR Module has a memory component containing the design history, lessons learned, and concurrent engineering experience and knowledge. It also has a reasoning component to analyze relevant cases from memory and make pertinent suggestions to the designer. The memory grows and learns over time as the system is used. The machine intelligence techniques to develop this module come from the paradigm of case-based reasoning.

The IntCAD Module has a features-based parametric CAD component to help the designer generate preliminary designs. It also has a rule-based reasoner which enables "what-if" parametric analyses, interfaces to performance analysis routines, and manipulates concurrent engineering knowledge. The machine intelligence techniques to develop this module are rule-based reasoning and object-oriented programming.

The DSM Module has a goal manager to provide the design plan enforcement mechanism. It also has a design knowledge capture component to record decision rationale. The machine intelligence techniques used in this module are expectation-based processing and explanation-based processing.

6 The Modules

CBR Module

CBR is a paradigm of problem solving which uses past solutions and lessons learned to solve new problems. Researchers agree that the quality of decision-making particular to a design

can be vastly improved if features from similar past design cases are considered. In fact, the training of novice designers is case-based [Sycara 89c].

Our experience (including analysis of other researchers' results) indicates that CBR offers the following general advantages:

- 1) CBR is an efficient "jump-start" technique: focused problem solving is achieved faster, through reminders, than blind search (such as rule-based) techniques.
- 2) Reminded cases provide efficient "repair" strategies [Sycara 89b] when debugging partial solutions: previous successes, failures, and lessons learned are applicable piece-wise, even when the reminded pieces are from different (but related) domains.
- 3) Cases enable learning: feedback with respect to the success or failure of a proposed solution, along with justifications as to why, are recordable and become new cases in memory to benefit future problem solving.

Simply stated, advantages 1 and 3 provide intelligent "book ends" to a problem solving session; advantage 2 steers the process in between. The following paragraphs in this section present how these general advantages apply in the CBR Designer's Assistant.

Advantage 1 is used to "jump-start" the design process. A features-based definition of a part to be designed is input. The most relevant design matching those features is retrieved from memory. Presenting the retrieved design to the user/designer provides an immediate seed preliminary design to focus the design process.

Advantage 2 is used to optimize the preliminary design piece-wise. As the user/designer modifies pieces of the preliminary design (assuming it did not exactly match the input features), additional similar pieces are retrieved from memory. The reminded pieces help repair the desired changes by reporting violated constraints (such as producibility, cost, or inspectability) from the past and adapting their solutions to the present.

Advantage 3 enables the CBR Designer's Assistant to learn. Explanations and justifications as to how/why design decisions are made (both good and bad) are recorded to benefit future similar designs.

The introduction to the DARPA Machine Learning Program Plan [DARPA 89] lists five advantages of CBR: performance enhancement, uncomplicated learning, cases serve as generalizations, scalability of methods, and easier knowledge acquisition. We agree with these, however space here does not permit a detailed comparison.

IntCAD Module

Currently there exist software tools which enable features-based parametric design. Some of the more novel tools incorporate object-oriented programming and rule-based knowledge as well (ICAD from ICAD, Inc., and Wisdom Ware from Wisdom Systems, Inc.).

There are two basic issues with respect to the IntCAD Module. The first issue relates to the difficulty of customizing these off-the-shelf products. Considerable effort, expense, and commitment by engineering groups is required, however the results are well worth it.

The other issue is an integration concern. After customization, the tool must integrate with the rest of the modules in the CBR Designer's Assistant.

DSM Module

The DSM Module interfaces with the designer and manages the case-based and rule-based reasoning processes. Within the DSM Module are the Goal Manager and the Design Knowledge Capture (DKC) Module.

A popular theory of human intelligence states that learning involves several steps:

- 1) Expectation First there needs to be some form of expectation.
- 2) Surprise When things do not happen as expected, surprise occurs.
- 3) Explanation A surprise spawns the desire for an explanation.
- 4) Learning The analysis of an explanation leads to learning.

Step 1 is known as expectation-based processing. Representations of goals enable the Goal Manager to build expectations of what the user/designer will do.

Step 2 occurs when an expectation is not met. The ways in which the satisfied and unsatisfied goals relate to each other are used by the Goal Manager to further identify the type of surprise.

Step 3 is known as explanation-based processing. Representations of rationale enable the DKC Module to prompt the user/designer for acceptable explanations.

Step 4 occurs as a result of building an acceptable explanation for a surprise. The DKC Module records the explanation and associates it with the current design case. The case gets stored in the memory thereby learning an explanation which can be used for future cases.

7 State-of-the-Research

There are two central issues in CBR research: retrieving relevant cases from memory, and reasoning from the cases which are retrieved. For comparison, [DARPA 89] details three central issues: index selection, rank-ordering reminders, and adaptation. We would include the first two of these within our retrieval issue.

The most relevant research that we are aware of is that which is being performed by Sycara and Navinchandra at the Robotics Institute of Carnegie Mellon University. Their research is aimed at integrating case-based reasoning and qualitative engineering design. The domain of their research is in the area of the automated design of mechanical assemblies.

Another related research effort is being performed by Finger et al. [Finger 88]. Their research effort, named The Design Fusion Project, is large-scale in which they address automated design from a product life-cycle view. The domain of their research is electro-mechanical assemblies.

Martin Marietta Laboratories is working on the larger issues of how to manage and integrate concurrent engineering on a broad spectrum of design types, domains, activities and users in a project called Integrated Concurrent Engineering (ICE) [Mills 89]. The work is also addressing the issue of how to manage and integrate the large variety of design aids emerging from research laboratories including algorithmic, expert system-based, finite element-based, simulation-based, and so on. The domain of their research is mechanical assemblies.

There are numerous other intelligent design research efforts [Brown 85], [Bulkeley 89], [Dixon 88], [Hayes-Roth 83], [Ullman 88]. One last project in engineering design will be cited. The KIDS (Knowledge Integrated Design System) project at Wright-Patterson Aeronautical Laboratories [KIDS 89] is focused on many of the same issues as the ICE project. The domain of that research is process-oriented: rolling, forging, casting, and some advanced composites processes.

The domain of design is actually present in many human-based activities, not just engineering. Our research is related to many other research efforts which look at design from a non-engineering view: for example, two parties designing a mediation to a dispute [Simpson 85] or a chef designing a meal [Hinrichs 89], [Hammond 86]. The reader is referred to [DARPA 88] and [DARPA 89] which offer excellent compilations of work in CBR.

There are some researchers in the CBR community that have just recently begun to address the design domain. The sheer growth in the number of these relevant research projects, compared to just a year ago, indicates that CBR has the potential to offer many benefits to the domain of design. Synopses of a representative sample of these research projects follows.

Birnbaum and Collins [Birnbaum 89] consider "design themes" as part of an indexing vocabulary to provide cross-contextual reminders.

Goel and Chandrasekaran [Goel 89] work in the adaptation of previous design cases by considering the designer's functional model of his/her causal understanding of the behavioral and structural aspects of a design (called a device model).

Alexander et al. [Alexander 89] are formalizing a representation scheme (called a design tree) which allows design cases to be transformed into new designs using a calculus they have developed.

Barletta and Hennessy [Barletta 89] have developed a method which adapts pieces of previous cases to optimize the placement of composite parts to be cured in an autoclave.

Sycara and Navinchandra [Sycara 89a], [Sycara 89c] are using a multi-layered representation scheme (structural, functional, causal, and qualitative) to effect index transformation during reminding.

Our own work [Pulaski 88a], [Pulaski 88b], [Hightower 89] has been focused on building CBR memories automatically from a case base (a database plus a knowledge base), and then optimizing the memory using neural network processing.

Although outside the scope of this paper, it should be mentioned that the IntCAD Module and DSM Module each involve many issues and relate to other ongoing research. Perhaps the area most needy of a breakthrough is in design knowledge capture. We agree with Wechsler [Wechsler 86] that part of the solution needs to be explanation-based. The reader is referred to Freeman [Freeman 88] for an excellent overview of the design knowledge capture problem. Also, the DSM Module would benefit from a more sophisticated user model, beyond simply goal tracking.

8 Discussion

Several observations are notable regarding current work in CBR and integrated design. The ICE, KIDS, and Design Fusion projects are large-scale. Much of this and other work has focused on examples involving complex electro-mechanical assemblies. Although these projects should some day produce very significant results, the size of the efforts and the complexity of the domain will make progress very difficult.

We believe that there is a shortage of small-scale projects focused on simple designs. The study of CBR in design will benefit sooner by addressing the issues with simpler examples. It is our recommendation that single-component designs be considered, such as molds, fasteners, or composite parts. These examples will reduce the amount of complexities arising from sub-component interactions, yet results will still be beneficial and usable.

Another observation is that much work is geared toward a high level of design automation. Our position is that using machine intelligence to assist a designer (rather than replace him/her) will produce real benefits faster and at less expense than attempting to automate the construction of an original design (without including a human designer).

It is generally agreed that while spending only 5% of manufacturing costs during design, decisions made during design commit 95% of manufacturing costs. There are many stages throughout the design process which can benefit from machine intelligence, however we feel that the stage of preliminary design can benefit the most. Using CBR to bring downstream considerations into the preliminary design process will assist the designer to optimize the upstream design for cost as well as other considerations (e.g. quality, producibility, maintainability, repairability, and other life-cycle concerns).

Acknowledgement

The author would like to extend his warmest thanks to Mr. Thomas J. Leahey who selflessly made himself available for numerous brainstorming sessions and discussions. Tom's expertise in the area of design contributed significantly to the ideas presented in this paper.

References/Bibliography

- [Alexander 89] Perry Alexander, Gary Minden, Costas Tsatsoulis and J. Holtzman, "Storing Design Knowledge in Cases," in [DARPA 89], pp. 188-192.
- [Barletta 89] Ralph Barletta and Dan Hennessy, "Case Adaptation in Autoclave Layout Design," in [DARPA 89], pp. 203-207.
- [Birnbbaum 89] Lawrence Birnbbaum and Gregg Collings, "Reminders and Engineering Design Themes: A Case Study in Indexing Vocabulary," in [DARPA 89], pp. 47-51.
- [Brown 85] D. C. Brown and B. Chandrasekaran, "Expert Systems for a Class of Mechanical Design Activity," Knowledge Engineering in Computer-Aided Design (underlined), J. S. Gero (Editor), Elsevier Science Publishers B. V. (North-Holland), 1985, pp. 259-283.
- [Bulkeley 89] Debra Bulkeley, "Get Smart: Software that Knows the Answers," *Design News* (italics), February 27, 1989, pp. 72-80.
- [DARPA 89] Proceedings of the Case-Based Reasoning Workshop, Sponsored by DARPA, Pensacola Beach, Florida, May 31 - June 2, 1989, Morgan Kaufmann Publishers, San Mateo, California.

- [Dixon 88] J. R. Dixon, "Designing with Features: Building Manufacturing Knowledge into more Intelligent CAD Systems," in Proceedings of Manufacturing International '88, Atlanta, Georgia, April 17-20, 1988, pp. 51-57.
- [Finger 88] Susan Finger, Mark S. Fox, Dundee Navinchandra, Friedrich B. Prinz, and James R. Rinderle, "The Design Fusion Project: A Product Life-Cycle View for Engineering Designs," a proposal, May 26, 1988.
- [Freeman 88] Michael S. Freeman, "The Elements of Design Knowledge Capture," in Proceedings of AISA 88, November 15-16, Huntsville, AL, NASA Conference Publication 3013, pp. 39-46.
- [Goel 89] Ashock Goel and B. Chandrasekaran, "Use of Device Models in Adaptation of Design Cases," in [DARPA 89], pp. 100-109.
- [Hammond 86] Kristian Hammond, "Case-Based Planning: An Integrated Theory of Planning, Learning and Memory," Ph.D. Thesis, Yale University, New Haven, CT.
- [Hayes-Roth 83] Frederick Hayes-Roth, "Using Proofs and refutations to Learn from Experience," Machine Learning (underlined), Tioga Publishing Company, Palo Alto, California, 1983, pp. 221-240.
- [Hightower 89] Ron R. Hightower and Kirt Pulaski, "An Investigation of Enhancing ELMO with Neural Network Processing," poster session at DARPA CBR Workshop, Pensacola Beach, Florida, June 1, 1989.
- [Hinrichs 89] Thomas R. Hinrichs, "Strategies for Adaptation and Recovery in a Design Problem Solver," in [DARPA 89], pp. 115-118.
- [Kids 89] "KIDS: Knowledge Integrated Design System," RFP No. F33615-89-R-5619, Headquarters Aeronautical Systems Division, Wright-Patterson Air Force Base, Ohio, May 3, 1989.
- [Mills 89] John Mills, "ICE: Integrated Concurrent Engineering Project Overview," Martin Marietta Laboratories, Baltimore, Maryland, 1989 (unpublished).
- [Pulaski 88a] Kirt Pulaski, "ELMO: An Episodic Long-term Memory Organizer for Case-Based Reasoning," AAAI Case-Based Reasoning Workshop, St. Paul, Minnesota, August 23, 1988, pp. 107-112.
- [Pulaski 88b] Kirt Pulaski and Cyprian E. Casadaban, "Case-Based Reasoning: The Marriage of Knowledge Base and Data Base," in Proceedings of Fourth Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, November 15-16, 1988, NASA Conference Publication 3013, pp. 183-190.
- [Simpson 85] Robert L. Simpson, "A Computer Model of Case-Based Reasoning in Problem Solving: An Investigation in the Domain of Dispute Mediation," Ph. D. dissertation, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia, Technical Report GIT-ICS-85/18, 1985.
- [Sycara 89a] Katia Sycara and Dundee Navinchandra, "Index Transformation and Generation for Case Retrieval," in [DARPA 89], pp. 324-328.
- [Sycara 89b] Katia Sycara and Dundee Navinchandra, "Representing and Indexing Design Cases," in Proceedings of IEA/AIE-89, Tullahoma, Tennessee, June 6-9, 1989.
- [Sycara 89c] Katia Sycara and Dundee Navinchandra, "Integrating Case-Based Reasoning and Qualitative Reasoning in Engineering Design," in Proceedings of AI in Engineering 89 Conference, Cambridge, U.K., July, 1989.
- [Ullman 88] David G. Ullman, Thomas G. Dietterich and Larry A. Stauffer, "A Model of the Mechanical Design Process Based on Empirical Data," *AI EDAM* (italics), 2(1), pp. 33-52, Academic Press Limited, 1988.
- [Wechsler 86] D. B. Wechsler, "An Approach to Design Knowledge Capture for the Space Station," in Proceedings of Space Station Automation II, sponsored by SPIE, Vol. 729, 1976, pp. 106-113.

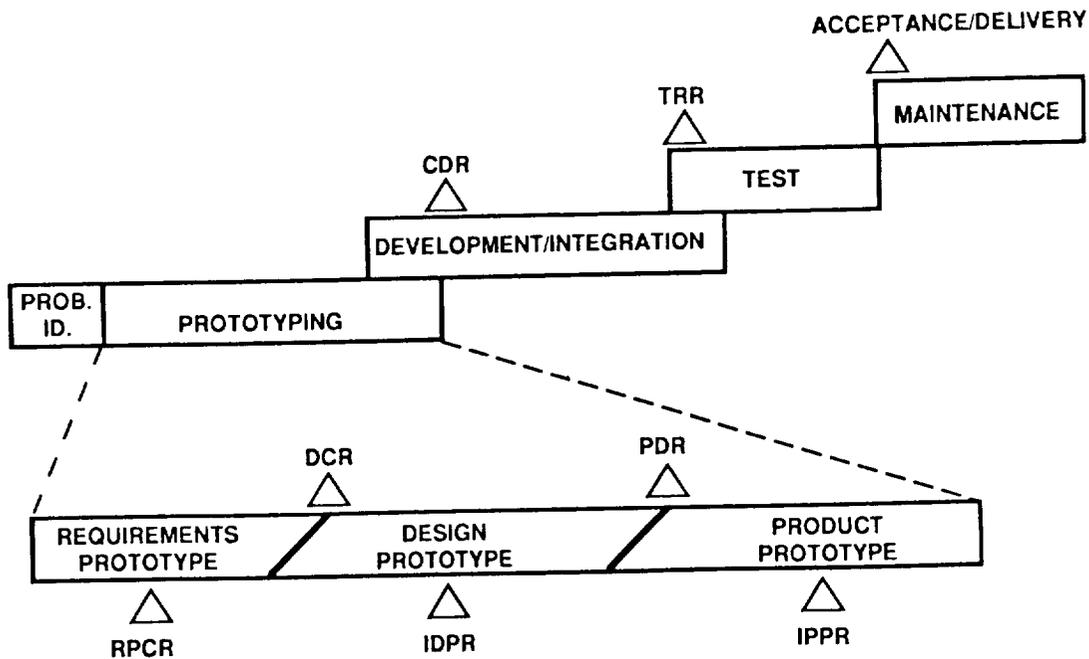
ARTIFICIAL INTELLIGENCE SOFTWARE ENGINEERING (AISE) MODEL

Peter A. Kiss
 SENTAR, Inc.
 3225 Bob Wallace Ave.
 Huntsville, Al. 35805

ABSTRACT

The American Institute of Aeronautics and Astronautics has initiated a Committee on standards for Artificial Intelligence. Presented here are the initial efforts of one of the working groups of that committee. The Purpose of this paper is to present a candidate model for the development life cycle of Knowledge Based Systems. The intent is for the model to be used by the aerospace Community and eventually be evolved into a standard.

The model is rooted in the evolutionary model, borrows from the spiral model, and is embedded in the standard Waterfall model for software development. Its intent is to satisfy the development of both stand-alone and embedded KBSs. The phases of the life cycle are shown in the figure below and detailed in the paper, as are the review points that constitute the key milestones throughout the development process. The applicability and strengths of the model are discussed along with areas needing further development and refinement by the aerospace community.





Detecting Perceptual Groupings in Textures By Continuity Considerations

Dr. Richard J. Greene

Abstract. This paper presents a generalization of the second derivative of a Gaussian D^2G operator to apply to problems of perceptual organization involving textures. Extensions to other problems of perceptual organization are evident and a new research direction can be established. The technique presented is theoretically pleasing since it has the potential of unifying the entire area of image segmentation under the mathematical notion of continuity and presents a single algorithm to form perceptual groupings where many algorithms existed previously. The eventual impact on both the approach and technique of image processing segmentation operations could be significant.

Introduction

The notion of "continuity" provides a unifying framework for representing and solving the problems of perceptual organization. For example, the vertical lines in Figure 1 tend to be organized into three distinct groups based on proximity. From this example, one can infer the existence of a distance threshold among the lines. Lines whose spatial neighbors are within the distance threshold maintain continuity of distance relations and are grouped together. Lines which exceed the threshold cause a discontinuity in distance relationships and form another grouping. Hence, a notion of continuity/discontinuity can provide a convenient concept for formalizing the language and mathematical treatment of perceptual organization.

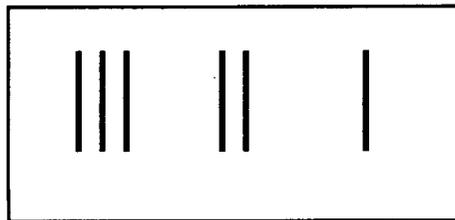


Figure 1. Continuity by proximity.

More generally, given a "space", the notion of continuity depends on a definition of a distance metric and "smoothness" over that space related to the distance metric. Discontinuities occur at places within the space which are not "smooth enough". Perceptual groups are homogeneous precisely because they are "continuous" with respect to some grouping property. In short, the elements grouped together may differ in exact detail from one another but only within limits. Thus, the goal perceptual organization is to locate discontinuities among the perceptual elements and, by doing so, isolate "continuous" groups of elements.

Unfortunately, as the paragraphs above imply, an operational definition of "continuity" can be elusive. Natural language lacks the precision to adequately define "enough" or "too much".

Euclidean geometry, on the other hand, can be too precise in characterizing the concerns of continuity and thus result in large descriptions consisting of a superposition of special cases.

This paper presents several problems of perceptual organization using a simple texture to illustrate the problems of determining continuity. The Laplacian of the Gaussian convolution operator D^2G is generalized to detect discontinuities within this context and thus form perceptual groupings.

Perceptual Organization in Textures

Textures may be defined as a regularity of the spatial distribution of texture elements (or texels). For the sake of illustration, assume the texels are a short, equal length line segments, each associated with an angle it makes with an imaginary horizontal line oriented from the left of the image to the right. Consider the texture of the "wavy plane" in Figure 2. The texels are the line segments and their spatial distribution defines a specific texture. Textures may also define perceptual grouping: Places where the texels do not vary "smoothly" define a discontinuity and may indicate perceptual groupings.

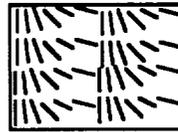


Figure 2. Lined Texture.

Notice how the perceptual groups differentiated by continuity of texel angle emerge from Figure 2. In some cases, the change among the texels is not drastic enough to cause a "discontinuity". However, in other cases, a group of texels will vary enough from their neighbors to form a new perceptual group. Even from this simple example, one can experience the detection of discontinuities among texel groups.

Clearly, the notion of continuity offers a useful conceptual framework for discussing perceptual grouping. However, to be applied, we must formalize the detection of texture discontinuities within a mathematical framework and derive an algorithm from the mathematical theory. This is done in the next section.

Detecting Discontinuities

The literature of edge detection has documented many techniques for discontinuity detection with respect to intensity values[2,3]. A line is defined by its edge points which, in turn, lie on a "steep change" in intensity among the edge point and some of its neighbors. In fact, the line itself is often modeled as a step function and edge points detected by taking spatial derivatives in order to locate the maxima of the step function. The maxima of the step function is located by locating places in the image where second spatial derivative crosses zero.

One theoretically compelling technique to emerge from this approach has been the Laplacian of the Gaussian convolution operator D^2G [4]. The motivation for this operator is twofold: convolving the image with a Gaussian lessens intensity "noise" and a critical point (maximum or minimum) of the edge's step function will be found wherever the second spatial derivative (Laplacian) crosses zero. Furthermore, the D^2G is the least complex operator which is rotationally symmetric. In other words, the Gaussian convolution lessens the overall effect of relatively isolated intensity changes while the spatial Laplacian indicates areas where the intensity values change the "fastest". Because the operator is rotationally symmetric, edges from any angle are detectable. These areas of maximal change are prime candidates for "discontinuities". The theory of the D^2G operator is explained in Marr.

However, the D^2G operator has been applied only to detecting discontinuities in intensity values. D^2G could be generalized easily if one can find a way to transform other perceptual grouping cues into intensity values and then apply the D^2G operator to the resulting transform image. This was the approach taken for texture. From the examples presented, one can readily see that different textures have different properties and more complicated definitions of continuity. Nevertheless, once the texels are transformed to intensities, the D^2G operator can locate where the property changes the fastest. This then will locate a candidate for a discontinuity.

For illustration purposes, uniformly sized simple texels were selected and distinguished only by the angle each makes with a horizontal reference line placed at the bottom of the texel. For the sake of simplicity, a simple 10 digit code was used. Figure 3 illustrates this. For example a line with two vertical texels and two horizontal texels would be encoded as 5,5,0,0. Any other coding scheme which defines a metric space over the textures is acceptable. The metric space restriction is discussed below.

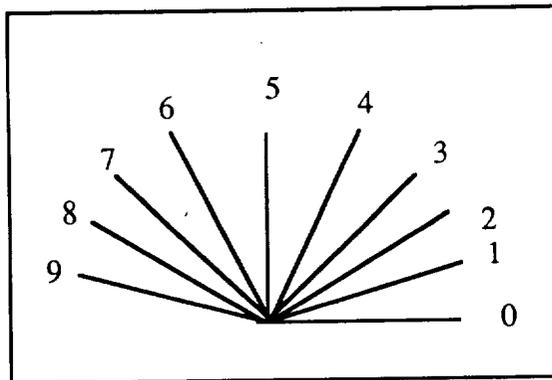


Figure 3. Texel properties.

There are two challenges in applying the D^2G operator to an image consisting of these texels: first, how to transform texels to intensity values associated with their individual texel distinctions yet indicative of the spatial distribution of texels, and, second, how to associate the transformed intensity value in the transform image to a texel in the original image? The texels were transformed into intensity values by mapping the texel's angle (0 - 180 degrees) to an intensity by using its code (0..9). Thus texels with similar angles have similar intensities. The second

problem, that of associating a texel to a transformed image is handled by making the transformed image's pixels represent a polygon which covers exactly one texel. In effect, the original image is overlaid with a grid of polygons or "tiled". The original image must be divided into tiles such that every texel is contained in exactly one section and that every pixel in the original image is accounted for by exactly one tile.

Once this tiling is accomplished, the texel must be transformed to an intensity value which satisfies the axioms of a metric space [1]. The metric space restriction insures that the distance metric between texels is continuous and that two texels that are far away in metric value are indeed different. For example, if the metric defined above had been extended to include a horizontal texel "10", a "0" and "10" would be far apart in the metric space but very close (indistinguishable!) visually. In short, the metric space restriction allows us to ignore "wraparound" phenomena; i.e. two objects deemed "far apart" by the metric are actually very close together because the objects are about to disappear from one end of the space and appear on the other end.

The transformed image is convolved with the D²G operator and the zero crossings are noted: these candidates for the texel angle discontinuities. Since the mapping between the transformed image resolution and original image is uniform, the corresponding image pixels can be identified as candidate points of discontinuity.

Results

This section presents texture images, their corresponding transform images, and the output of the transform image convolved with a D²G operator with zero crossings marked. These zero crossings are candidates for discontinuities and hence, perceptual groupings. Space limitations restrict a full treatment of the experiments as well as the details of the D²G operator. The standard deviation of the Gaussian used was three pixels; this parameter proved sufficient to detect sharp discontinuities on the simple textures used. Given below are the results of a simple texture. This example is intended to clearly demonstrate the technique and is not intended to limit the range of spatial scales or texture types amenable to processing.

Figure 4 shows a simple texture image with uniform length texels and Figure 5 shows the transformed intensity image. Finally, Figure 6 shows the results of convolving the transformed intensity image with the D²G operator. Note the sharp lines locating where the texture became discontinuous or changed most rapidly.

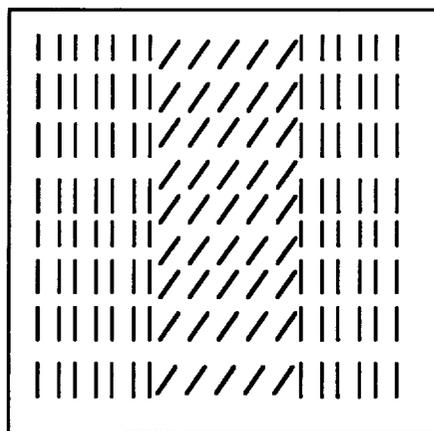


Figure 4. Texture Image.

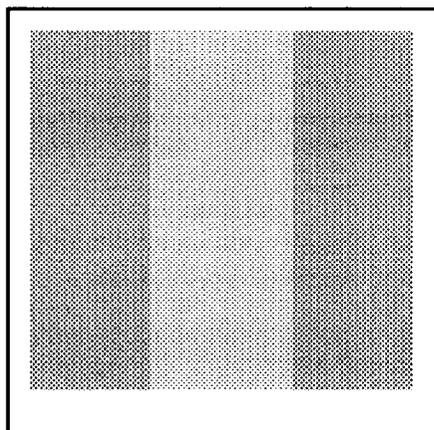


Figure 5. Transformed Intensity Image of Texture Image.

In Figure 6 the lines formed by the zero crossings indicate the distinct perceptual groupings. Note how the groupings formed by the zero crossing curves closely parallel intuition. Other, more comprehensive, experiments support these results and suggest the technique handles a variety of textures and spatial scales.

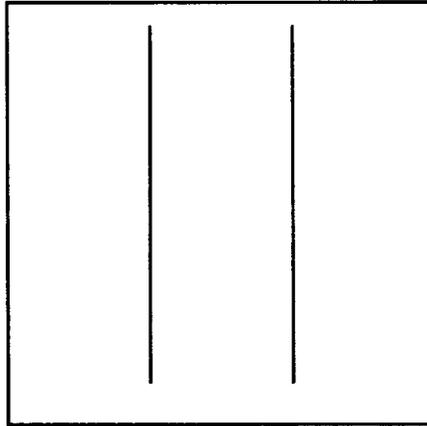


Figure 6. Zero Crossings of the Convolution.

Conclusions

This section suggests how the D^2G operator can be extended to apply to distance or any other criteria which exhibits a "continuity". The experiments with the simple texture images suggest that continuity considerations can play a key role in determining perceptual groupings. First the property in question must be represented by an intensity value. This can be done by defining a metric space on the property and using the metric as an intensity value. There are no restrictions on the dimension of the metric. For example, there is no reason to prohibit a two-dimensional metric combining texel length and angle. Discontinuities could then represent breaks in texel lengths, texel angles, or both. Perceptual groupings can be formed from either of these and very strong groups can occur where both of these properties change the fastest. Research is already investigating multi-dimensional properties.

The resulting image is convolved with the D^2G operator and zero crossings noted. The D^2G operator is well-suited to detecting the discontinuities on various scales and is rotationally symmetric. Finally, the zero crossings represent where the property in question may be discontinuous. In fact, some psychophysical evidence suggests that the eye/brain may implement some form of D^2G convolution for the location of discontinuities in intensity value[3, 4]. It would certainly be theoretically pleasing to discover that discontinuities in other properties such as texture are located in essentially the same way.

References

1. BRYANT, V. *Metric Spaces*, Cambridge University Press Cambridge, England 1987
2. CANNY, M. Computational Edge Detection In *Readings in Computer Vision: Issues, Problems, Principles, And Models*, Fischler, M. , and Firschein O. , Eds, Morgan Kaufmann Publishers, Los Altos,CA. , 1987

3. HILDRETH, E. The Detection of Intensity Changes by Computer and Biological Vision Systems, *Computer Vision, Graphics, and Image Processing*, 22,1 (April 1983) 1-27

4. MARR, D. *Vision*, W. H. Freeman and Company, San Francisco, CA. ,1986

A Vision-Based Telerobotic Control Station

Brian Tillotson
Advanced Civil Space Systems, Boeing Aerospace and Electronics
P.O. Box 240002, Huntsville AL 35824-6402

ABSTRACT

A telerobotic control station is described. In it, a machine vision system measures the position, orientation, and configuration of a user's hand. A robotic manipulator mirrors the status of the hand. This concept has two benefits: control actions are intuitive and easily learned, and the workstation requires little volume or mass.

INTRODUCTION

Telerobotic systems have great potential for making space assets more flexible and productive. In particular, astronauts might use telerobotic systems to reduce the need for extravehicular activity or to handle hazardous tasks. The design of a telerobotic control station for use in manned spacecraft must meet several constraints. The control station interface should be easy to use and to learn. Mass and volume are strongly constrained in spacecraft, so the control station's mass and volume must be minimized. Repair or replacement of spacecraft equipment can be costly or impractical, so reliability must be high.

The requirement to be easy to use and learn is often unmet due to configuration differences between the manipulator and the control interface. Many robot manipulators bear a superficial resemblance to the human hand. They have grippers which crudely approximate the human thumb and fingers, and wrists which pitch and roll as the human wrist does. These manipulators are often attached to robot arms which approximate the human elbow and, to a lesser extent, the shoulder. In contrast, human control interfaces to robot arms and manipulators have typically used joysticks. The joystick is not a particularly natural interface, whether for robots or for other equipment. Substantial training and practice are required to reach moderate skill levels, and relatively few individuals have enough talent to efficiently use the device in tasks which demand precision. (The late Dr. Judith Resnick was noted for her exceptional skill at manipulating the Space Shuttle's robot arm via the joystick-based control station.)

Besides being a poor interface, a joystick control station takes up scarce volume and wall area. This space is unavailable for other needs even when the control station is not in use.

Joysticks are mechanical devices which must endure friction and stress. To be reliable, they must be robustly built. This requires substantial mass, expensive high-strength material, or both.

Two existing possible substitutes for joystick control are the DataGlove¹ and the Sensor Frame². The DataGlove is a special glove which measures the position, orientation, and configuration of the user's hand. The Sensor Frame is a set of infrared light sources and detectors arrayed around the screen of a video display; the user's fingers interrupt beams of light, thus revealing the fingers' positions. Both DataGlove and Sensor Frame are expensive and require special hardware.

This paper describes a vision-based control station which is intended to avoid the disadvantages of joysticks, and to do so with lower cost and less dedicated hardware than the DataGlove and Sensor Frame. It provides a natural-seeming interface, requires little mass and volume, and has no moving mechanical parts. A simple implementation of the control station concept has been assembled and tested. The first section below describes the physical configuration of the control station, its computational support, and its connection to the robot. The second section describes

the algorithm for image interpretation and robot control. The third describes two alternatives for mapping hand positions to manipulator actions and explains the rationale used for selection between them. The fourth section briefly presents the results of initial tests with the control station. Finally, areas for further development are discussed.

INTERFACE CONFIGURATION

The control station consists of a video camera, a known visual background, and a specially marked glove which the user wears. In a fully developed control station, the glove should be unnecessary. (The control station also includes a video display, which provides feedback about the robot's actions and environment. This paper does not address the subject of feedback.) The user places his or her gloved hand in the video camera's field of view. An image processing computer detects the hand and measures its position, orientation, and configuration. These measurements are mapped into a set of manipulator commands which are sent to the robot. Thus, the user can direct the robot to flex its wrist by tilting his or her hand, or can direct the manipulator to close its gripper by bringing his thumb and forefinger together.

The current hardware configuration of the system is shown in Figure 1. The user's hand motions, observed by the video camera, determine the actions of a robot gripper in another room. The Sun workstation handles most of the non-image processing load; the Explorer workstation only transfers commands from the Ethernet to the robot. Not shown in the figure are a television camera and a monitor which allow the user to view the robot.

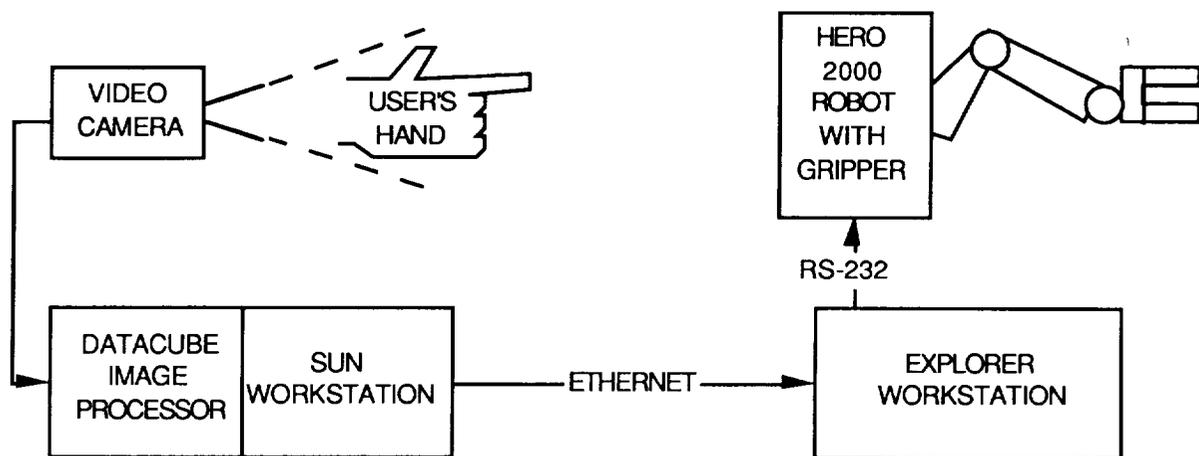


Figure 1. Current lab configuration of control station.

The glove is marked as shown in Figure 2. Viewed against a dark background, the light glove is easy to detect by thresholding. The dark rings on the thumb and index finger are detected by morphological image operations, allowing the hand's roll position and its grip width (distance between thumb and index finger) to be measured.

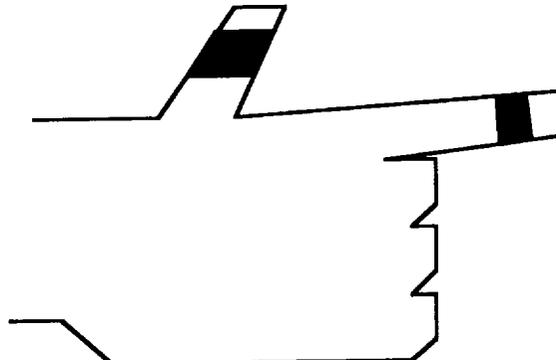


Figure 2. Marks on glove.

ALGORITHM

The robot manipulator has several parameters, e.g. pitch or grip width, each having a finite number of states. Each parameter is independent, that is, the state of one parameter has no effect on the state of another. The purpose of the program is to control the state transitions of each manipulator parameter and thereby to control the manipulator itself.

The control station algorithm is presented below as Algorithm 1. The algorithm's outer structure is an eternal loop. In each pass through the main loop, the first step is for the computer to examine an image, determine whether a glove is present, and if so, measure its position and orientation parameters. Each parameter is discretized, with each discrete value mapping to one state of the corresponding manipulator parameter. If the indicated state does not match the current manipulator state, then the program issues a command to change the robot's state. To avoid spurious state changes, the algorithm only sends a command to the robot if the same change is indicated in two consecutive images. If the glove is removed from the field of view, the robot maintains its current state.

```

Initialize the current-state for each manipulator parameter.
While (True) Do:
  Acquire image.
  Threshold glove from background.
  If glove in view:
    Measure glove position/orientation parameters.
    For each glove parameter:
      Quantize.
      Map to indicated state for manipulator parameter.
      If indicated-state matches current-state for the parameter:
        Do nothing.
      Else:
        If indicated-state matches previous-indicated-state:
          Set current-state to indicated-state.
          Send appropriate command to robot.
        Else:
          Set previous-indicated-state to indicated-state.
  
```

Algorithm 1. Hand position interpreter

MAPPING STRATEGIES

There are two straightforward ways to interpret glove position: either as a direct indicator of manipulator position, or as an indicator of the manipulator's rate and direction of motion. In position-to-position mapping, the robot hand moves to the same orientation as the user's hand. In position-to-rate mapping, the user moves his hand from the neutral position to a position which indicates motion in some direction or about some axis, e.g. to increase the wrist pitch, the user flexes his wrist upward. The robot wrist rotates upward continuously until reaching its limit or until the user returns his wrist to a neutral or downward position.

Position-to-rate mapping was selected because it allows the robot wrist to roll through its full range of motion, i.e. infinite in either direction, as long as the user's hand is tilted to the side. An implementation with position-to-position mapping would be unable to roll the wrist beyond the finite limits of human wrist motion. The position-to-position mapping may be easier to learn and use than the position-to-rate mapping, but that hypothesis has not been tested in this work.

Manipulator parameters currently supported are wrist pitch and gripper width. Wrist pitch is controlled as described above. Gripper width is determined by distance between the tips of the thumb and the index finger: small distance closes the gripper, large distance opens it, and neutral distance stops the gripper at the current width.

RESULTS

The system currently lets the user control the robot manipulator in two parameters with fair reliability (estimated at about 80% for each command motion). Response time is about 1.5 seconds, due to the rather slow image processing software, the specification that two consecutive images must agree on a command, and a slow data link to the robot. Removing the hand from the field of view and later returning the hand to the field of view usually does not confuse the interface software. It is expected that the system can be speeded up and made more reliable with moderate effort. Development of the system so far has required roughly 30 man-hours.

DISCUSSION

The equipment required for the control station is limited to a marked glove, a video camera, and computers. A video camera and computers will probably be present on any manned spaceflight, so the only unique equipment required is the glove. Future efforts may remove the requirement for a glove.

Further development of the vision-based control station is warranted. Logical extensions are to increase the number of parameters that are supported, to increase the speed, and to enable operations with a bare hand rather than a glove. Increasing speed and number of parameters are fairly straightforward. Operating without a glove can probably be accomplished by subtracting each image from the known background image; this would reduce dependence on visual contrast between the hand and the background.

A significant challenge for the vision system is to distinguish the positions of all five fingers, should that be necessary for future dextrous manipulators. A reasonable approach may be to use a color camera and to require the user to wear different colored rings on each finger. This avoids the inconvenience of a glove, yet eases the task of the vision system.

SUMMARY

A vision-based telerobotic control interface has been described. The interface requires little or no special equipment, which is an advantage for space applications. It provides a natural correspondence between user actions and robot motions.

The interface has been partially implemented with little effort. The current implementation has been successfully tested. However, the interface requires further development to improve reliability and speed.

¹DataGlove is a product of VPL Research, Redwood City, CA.

²Sensor Frame is a product of Sensor Frame Corporation, Pittsburgh, PA.

FIFTH CONFERENCE ON ARTIFICIAL INTELLIGENCE FOR SPACE APPLICATIONS

TITLE: THE REAL-TIME LEARNING MECHANISM OF THE SCIENTIFIC RESEARCH
ASSOCIATES ADVANCED ROBOTIC SYSTEM (SRAARS)

AUTHOR: ALEXANDER Y. CHEN

COMPANY: SCIENTIFIC RESEARCH ASSOCIATES, INC.

ADDRESS: 50 NYE RD., P.O. BOX 1058, GLASTONBURY, CT 06033

ABSTRACT

SRAARS is an intelligent robotic system which has autonomous learning capability in geometric reasoning. The system is equipped with one Global Intelligence Center (GIC) and eight Local Intelligence Centers (LICs). It controls mainly sixteen links with fourteen active joints, which constitute two articulated arms, an extensible lower body, a vision system with two CCD cameras and a mobile base. The on-board knowledge-based system supports the learning controller with model representations of both the robot and the working environment. By consecutive verifying and planning procedures, hypothesis-and-test routines and learning-by-analogy paradigm, the system would autonomously build up its own understanding of the relationship between itself (i.e. the robot) and the focused environment for the purposes of collision avoidance, motion analysis and object manipulation. The intelligence of SRAARS presents a valuable technical advantage to implement robotic systems for space exploration and space station operations.

I. INTRODUCTION

While the control engineers are concentrating on the relationship between actions and responses, the artificial intelligence discipline is emphasizing mainly on understanding the processes and the descriptions of the knowledge formation. Learning is one of the major items within the overlapped area of these two disciplines. As mentioned in Reference 2, learning can be divided into a high-level symbolic category and a lower-level numeric variety. However, a complete learning mechanism will employ symbolic knowledge to instruct the numerical control system, and the information of sensor inputs will be processed to update or upgrade the symbolic knowledge base. The integrated structure of both quantitative descriptions and qualitative descriptions stands for a more appropriate interpretation of the human learning function.

In order to maintain the learning procedure as a progressively improving process, the parallel coherency of both the short term memory (STM) and the long term memory (LTM) is necessary. There are several approaches in maintaining this kind of operation, e.g. using neural networks (Ref. 5), or employing schemata (Ref. 6). Furthermore, for a real-time expert controlled system, STM should directly interact with the control scheme and LTM would serve as the supervisor (or teacher) (Ref. 3). Then, the resulting learning mechanism can fulfill the six-step knowledge acquisition procedure: perception, strategy, decision making, execution,

result and evaluation. As shown in Figure 1, this closed-loop operation would eventually develop a certain extent of understanding which can be expressed as the integral of perceptions and the corresponded evaluations. That is,

$$\text{UNDERSTANDING} = \int \text{PERCEPTION} \wedge \text{EVALUATION} \quad (1)$$

The perceptions can be regarded as the qualitative descriptions of the event, and the evaluations would be the quantitative interpretations of the system's states. Therefore, a generic knowledge representation can be expressed as a set of tuples. That is,

$$\text{KNOWLEDGE} = \{(\text{Qualifier}[i], \text{Quantifier}[i]), i = 1, 2, \dots\} \quad (2)$$

For a specific section of knowledge (e.g. automobile repair), the formation may not be unique, but the utilization of the knowledge base would have an objective standard to qualify the specific knowledge representation as an expert system. This concept is adopted as SRAARS' fundamental principle of establishing the real-time learning mechanism.

Not every quantitative representation has its corresponded qualitative description; also, not every qualitative description can be expressed in quantity. However, it is perceived that those situations which can not be depicted as a tuple of qualifier and quantifier are normally handled without explicit knowledge. In other words, they are termed as unreasonable or illogical such that improving the system performance with learning is uncertain. Therefore, excluding those cases will not affect the generality of the learning mechanism. It is then assumed that every piece of meaningful knowledge can be expressed in a tuple of (Qualifier, Quantifier). When the process of understanding reaches the predetermined level, the evaluated system performance would converge to the designated standard and the machine intelligence will be regarded as an expert. Then the system behavior under the expert controller without closed-loop updating will be just as good as that of the feedback closed-loop in Figure 1. For safety reasons, the feedback closed-loop may still need to be maintained for responding to the potentially changing environment. However, the frequency of processing the understanding procedure can be reduced from real-time to on-line supervising.

The main objective of the real-time learning of SRAARS is geometric reasoning. Therefore, the perception is the geometric interpretation. The strategy is to move the robotic system in an effective way such that the unwanted collision can be avoided and the assigned object manipulation can be achieved. Decision making would determine the strategy selection based upon given knowledge of the physical correlation of the environment and the robot. The execution would then carry out the decision in a model-referenced adaptive control. The result would decide the termination of the action; and the evaluation will praise the system performance based upon the chosen criteria such as the difference between the desired design and the actual accomplishment, the execution time and the repeatability, etc. In the next section, these functions will be explained in detail. The third section will introduce the actual implementation plan. A prototype of SRAARS is under development. The anticipated results are discussed in the final conclusion section.

II. THE FRAMEWORK OF SRAARS INTELLIGENCE

SRAARS is a general purpose mobile manipulator. Its real-time learning mechanism can be applied to any specific target domain. The only limitation is the memory size. Three sets of representation elements would require user input for qualitative expression, quantitative description and relational specification, respectively. The knowledge system has a library of inference tools (e.g. forward chaining and backward chaining, induction and deduction) and a set of strategies for heuristics and hypothesizing. The knowledge base is designed with the basic format of Equation 2. The overall learning/control system of SRAARS is illustrated in Figure 2.

The user input set of qualitative expressions would be the entire set of vocabulary which the system can accept and utilize. The current level of the system intelligence does not have the capability of generating any new expressions by itself. The set of quantitative descriptions has two subsets: one contains the symbolic specifications of variables, parameters and constants, such as ranges (e.g. minimum, maximum), types (e.g. integer, real, or boolean), and formats (e.g. scalar, vector or matrix); another is comprised of the corresponded numerical values of each entity specified in the previous subset. Every symbolic entity can represent only one item throughout the entire learning/control process to eliminate unnecessary internal conflict. The third input set of relational specification would provide the initial system knowledge in linking and grouping all the available qualifiers and quantifiers. Some examples of relational specifications are formula or equations among symbolic quantities, rules of thumb among qualitative terms or certain production rules which relate both qualitative and quantitative items.

II.1 KNOWLEDGE UTILIZATION

Most currently available expert systems require a certain extent of human involvement (Ref. 7). Ranging from off-line query format man-machine interface to on-line process advisory, the developed knowledge systems are designated as an auxiliary component in the real world operations, which is because the knowledge programming is still confined to the level of managing the expressiveness of various knowledge representations, and the methodology for actually utilizing the digested information is not available. The knowledge evolution is basically a dynamic process. Since there are quite a few methods to express the understanding of one particular issue, and a major proportion of knowledge of understanding does not have a direct connection to the necessary action. Therefore, knowledge utilization itself becomes a rather ambiguous phenomenon. Fortunately, it is less ambiguous in those occasions of skill learning (Ref. 2). The knowledge utilization of SRAARS will serve two purposes: improving the understanding of the working environment and employing the updated geometric knowledge of both the robot and its surroundings to execute the given assignment.

In the field of robotics, problems such as the reachability analysis of articulated mechanical systems with redundancy, the collision avoidance problem of mobile robots and the object extraction of robot vision are very difficult to solve in a general platform. The human approach of dealing with these problems is through a long period of learning. Therefore, the

knowledge system of SRAARS is regarded as the core solution of mobile manipulation of the robot in general environments.

II.2 STRATEGY GENERATION

The system is initiated with knowledge of models of both the robot and the environment. The strategy at the beginning would be assuming these models are correct and plan the action accordingly to perform the given task. The decision maker then takes this strategy to execute the user commands. Some possible commands are "find object A in location C", "find object B in location D", "insert object A into object B, then place the sub-assembly on the conveyer belt", etc. The user commands would be translated into a sequence of actions for the controller to execute. The controller would employ a model-referenced adaptive scheme to plan the motion trajectory based upon the models and the assignment. Then, the actuating systems will be directed to realize the planned actions.

At the same time, an initial sensing action with default frequency is issued to monitor the robot motion and the changing environment. Therefore, both the vision system and other sensors such as ultrasonic sensors, proximity sensors or laser range finders, are sending data back to the brain. The data processor then converts the data into two formats. A direct conversion of input data to the values of system variables will be sent to the controller to serve as the output feedback of the control law. It will still maintain the data format. Another more complicated conversion will examine the input data with the existing knowledge base. The result is converted into an information format. The result will then be processed through the inference engine to conclude as knowledge verification and validation.

If there is a certain unacceptable inconsistency detected, then other strategy generation will be activated. The activation and the termination of different strategies are embedded in the kernel of the knowledge base which the user will not be able to access. However, users can change the level of acceptable inconsistency which triggers the activation and the termination of certain strategies, and increase or alter the strategy library. There are essentially two types of alternative strategies: heuristic approaches or hypothesis-and-test. The heuristic strategies are particularly suitable for object searching. When a segment of the environment model is detected to be incorrect (e.g. the original model does not have an object in that segment and current information indicates there is something in that area, or the previous model shows that there should be an object in that segment but the current sensor information cannot verify the existence of the object), a heuristic search strategy can be issued to take certain actions to modify the current model.

The hypothesis-and-test strategy can be utilized to resolve the potential reachability problem of the robot. The relationship between the robot model and the environment model can be very delicate when certain compliance activity is required. The articulated robot with many redundant degrees of freedom has a better chance to perform difficult interior operations. However, a general solution may require substantial computational effort (Ref. 4). Employing the knowledge engineering technique with an adequate hypothesis-and-test strategy is a practical

solution to resolve these types of problems. For this particular situation, there may not be any inconsistencies in modeling; instead, the maneuverability of the robot is the main focal point. If the robot has n degrees of freedom, then the projection of n dimensional joint space to the three-dimensional world coordinates present a very complicated combinatorial problem, especially when the topological structure contains tree branches or closed-loops. This type of knowledge system application will be the first step of utilizing artificial intelligence to solve some problems which even human beings cannot provide a definite solution.

III. THE SYSTEM DESIGN OF SRAARS

The realization of SRAARS emphasizes more on the system feasibility than on the utilization of the state-of-the-art microprocessor technology. It is fully aware that an adequate integration of both CISC and RISC architectures, such as Intel 80486 and 80860, Motorola 68040 and 88000 series, may provide a more powerful platform within a miniature form factor for intelligent robot development. However, this technology is not yet matured. It is therefore considered as the system design platform of the next generation of SRAARS. Intel 80386 is selected as the main processor for current SRAARS development. The major bus connection between GIC and LICs is determined to be VME bus.

The Global Intelligence Center is regarded as the brain and the central neural system of SRAARS. It has the following major features:

- Multitasking with virtual 8086 mode;
- Asynchronous processing of memory management, computation, inference and I/O interface;
- Hierarchical interrupt control; and
- Real-time kernel with concurrent programming constructs.

The functional diagram of SRAARS's brain is shown in Figure 3. Each individual modular functionality is listed as follows:

C3BUS:

- Mother-daughter communication handshake
- Command send out to designated LICs
- Receive status reports from LICs
- Network management and control
- Record update and archive

RVR:

- Image coprocessor communication
- Frame buffer management
- Image analysis, including edge detection, motion detection and object extraction

SANE:

- Sensor sampling and data acquisition control
- Sensor/actuator network management
- Device-independent database management
- Input buffer update
- Limited sensor fusion and impulsive reactions

EAN:

- Environment model formulation and update
- Consistency analysis
- Map formulation, revision and interpretation

KBL:

- Knowledge base formulation
- Knowledge utilization including rule extraction, rule-data association, and strategy generation
- Knowledge base/data base integration
- Knowledge input processing

MSM:

- Memory management among various devices (e.g. ROM, RAM, or disks)
- Memory initialization and reboot sequence
- Memory swap, cache and page
- Backup routine
- Memory status update

SPE:

- Model-based control monitoring
- Self-correction and self-calibration
- System power monitoring
- System performance record

MMI:

- Human operator input interrupt handling
- Panel-driven task assignment input
- Task analysis
- Task recording and retrieving
- Task planning
- Task status update

There are eight Local Intelligence Centers. Six of them are employed for controlling the mechanical linkage manipulation. Every one of the six LICs would control two to three degrees of freedom of the robot. The decentralized control systems would carry out the spatial planning commands under AISP (Ref. 1). The seventh LIC is a dedicated machine vision system which would control two CCD cameras and two to three degrees of freedom of motion to manipulate the orientations of cameras. Lower level image processing will be accomplished in this LIC as well. The eighth LIC is utilized to control the mobility of the base which includes the driving, steering, braking and backup operations. Some major features of LICs are listed below:

- Mother-daughter interactive communication
- Local trajectory planning and execution
- Status report back to GIC
- Impulsive response override

The connection between GIC and LICs basically follows the VME specification. GIC and Four LICs, including vision and base control, are located at the mobile base compartment. The rest of LICs will be located at the central portion of the lower body. There is a dedicated cable to

connect the "head", which contains cameras and the corresponded positioning mechanism, to the seventh LIC situated at the base. A general mockup of SRAARS is shown in Figure 4.

IV. THE CONCLUSION

The real-time learning mechanism of SRAARS is one of the innovative ideas which are incorporated into the development of SRAARS. It takes a plain and straightforward approach to integrate the knowledge system into the real world applications. The developed learning capability will move the knowledge engineering into the central arena of operating a complex dynamic system and perform as a decision maker. It would be interesting to see the comparative analysis with other conventional approaches if the explicit general solution may one day become available.

ACKNOWLEDGEMENTS

This research and development is supported by Contract NAS7-1012 of the NASA SBIR Program, from the NASA Resident Office at Jet Propulsion Laboratories, Pasadena, CA.

REFERENCES:

1. Alexander, Y.K. Chen, "AISP-A Robot Intelligence System in Spatial Manipulation," at ROBEXS '89, The Annual Workshop on Robotics and Expert Systems, Palo Alto, CA, August 2-4, 1989.
2. Anderson, R.L., "A Robot Ping-Pong Player: Experiment in Real-Time Intelligence Control," The MIT Press, Cambridge, MA, 1988.
3. Fu, K.-S., "Learning Control Systems," Reprinted in IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. PAMI-8, No. 3, May 1986, pp. 327-342.
4. Goldenburg, A.A., Benhabib, B. and Fenton, R.G., "A Complete Generalized Solution to the Inverse Kinematics of Robots," IEEE J. Robotics and Automation, Vol. RA-1, No. 1, pp. 14-20, March 1985.
5. Grossberg, S., "Neural Networks and Natural Intelligence," Bradford Books, MIT Press, Cambridge, MA, 1988.
6. Hanson, A.R. and Riseman, E.M., "VISIONS: A Computer Vision System for Interpretation of Natural Scenes," in Computer Vision Systems, (Hanson, A.R. and Riseman, E.M. eds.), Academic Press, New York, NY, 1978, pp. 303-334.
7. Hayes-Roth, F., Waterman, D.A. and Lenat, D.B. eds., "Building Expert Systems," Addison-Wesley Publishing Co., Reading, MA, 1983.

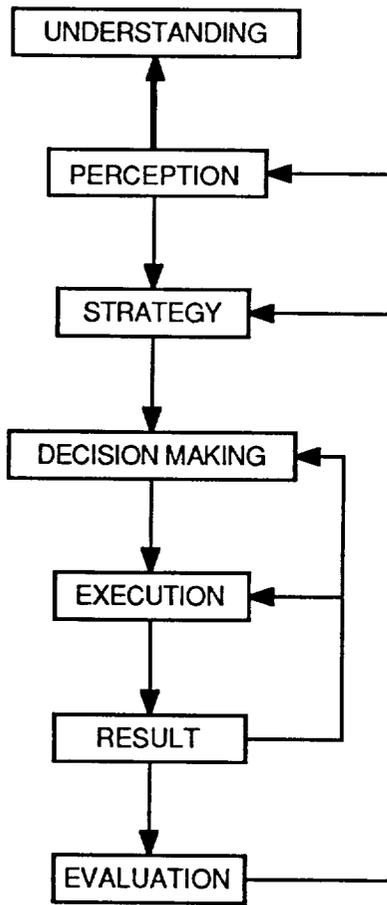


Figure 1. The Evolution Process of Understanding.

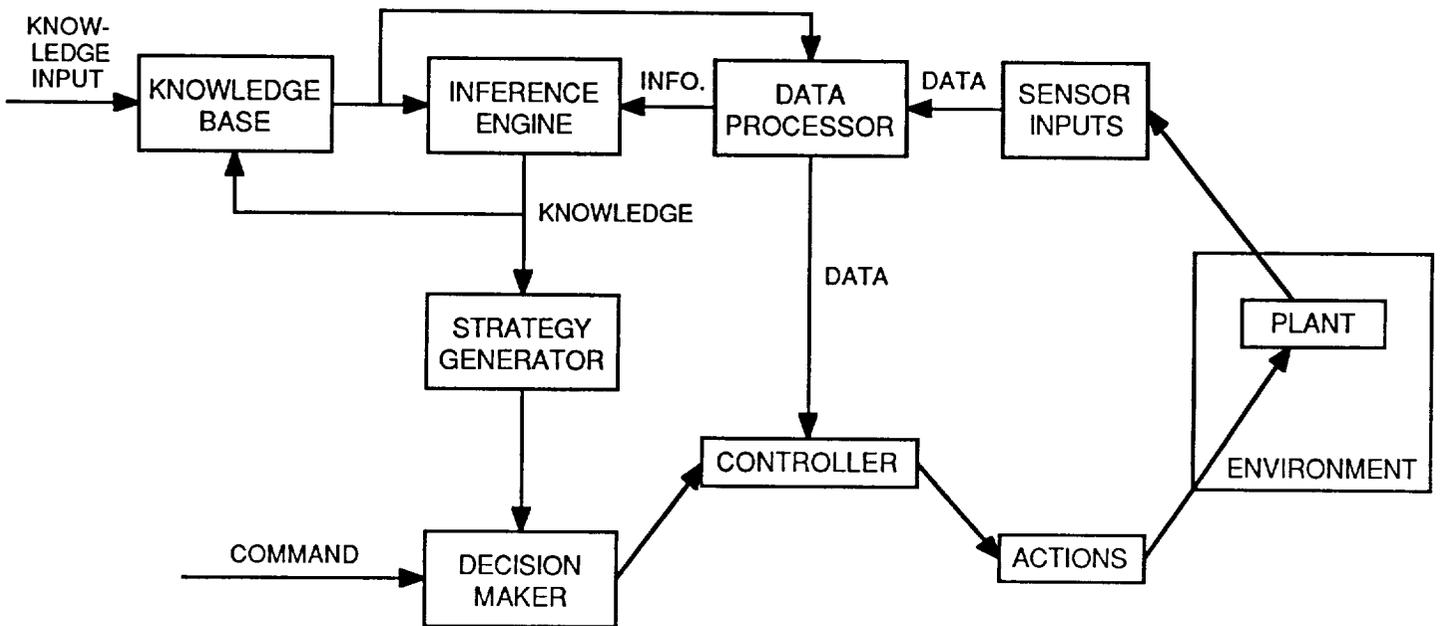


Figure 2. The Functional Diagram of the Learning Control System of SRAARS.

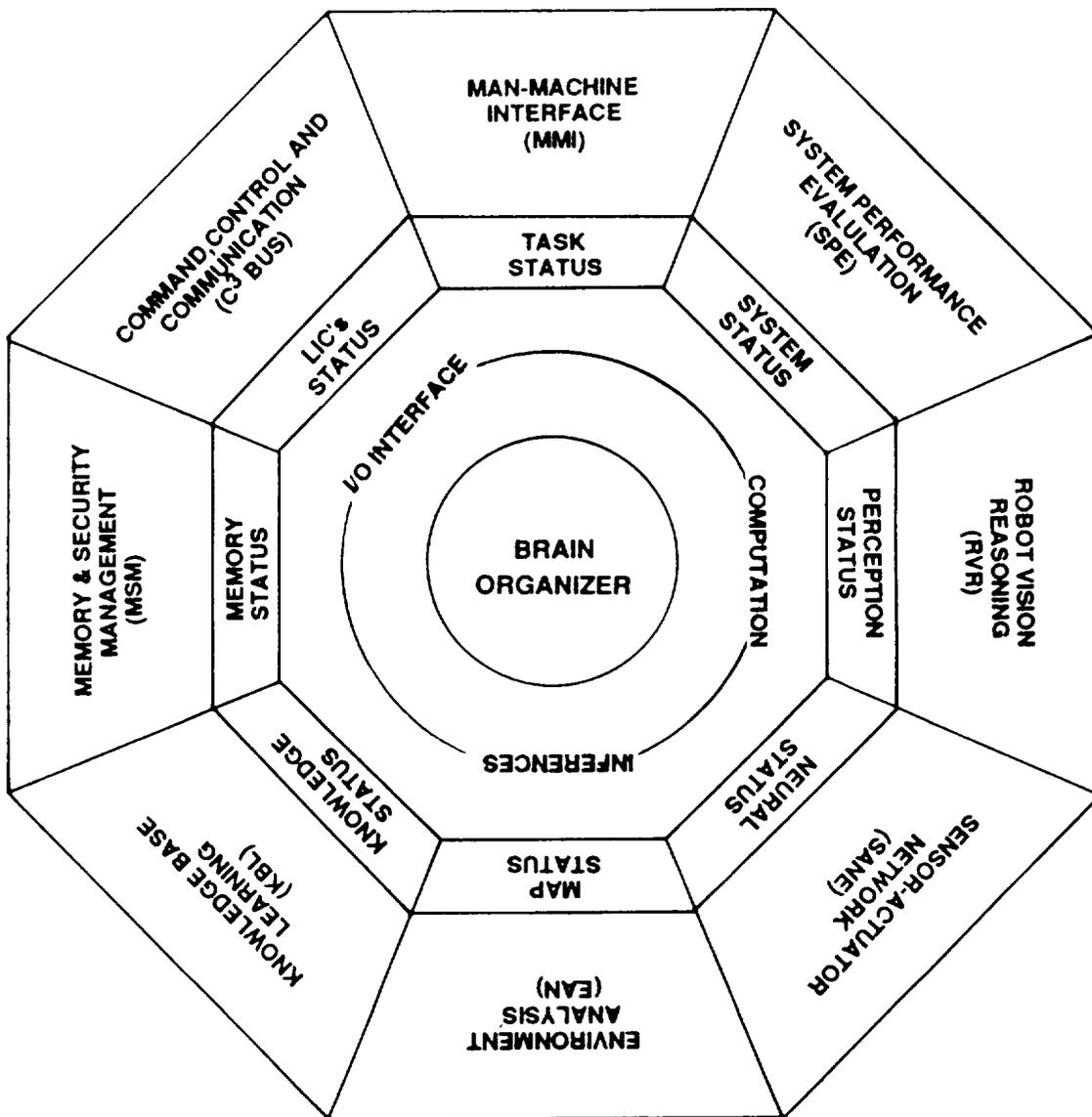


Figure 3. Functional Diagram of SRAARS's Brain.

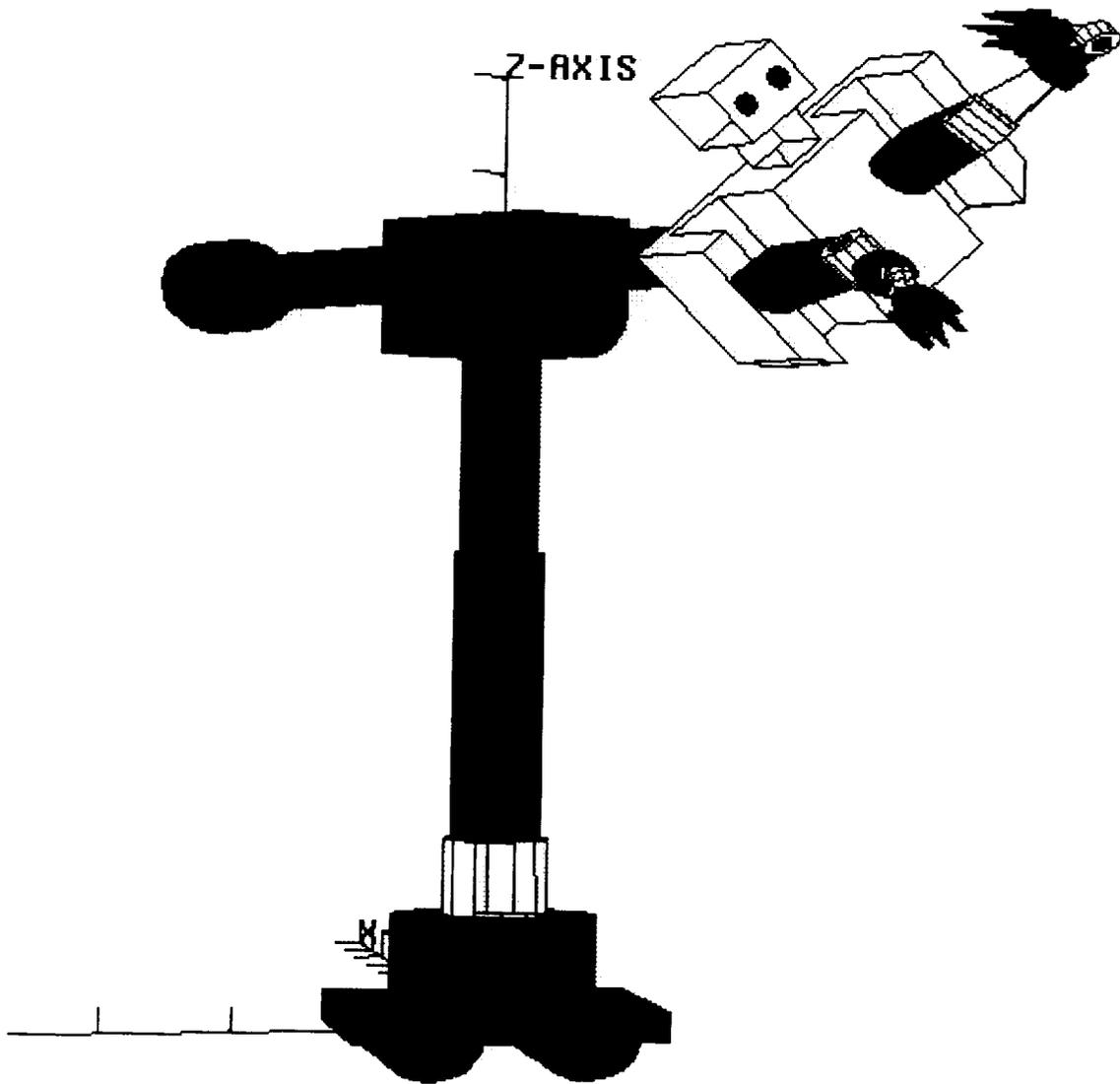


Figure 4. SRAARS Mockup

A Diagnosis System Using Object-Oriented Fault Tree Models

David L. Iverson
NASA Ames Research Center
Mail Stop 244-4
Moffett Field, CA 94035
iverson@pluto.arc.nasa.gov

and

F.A. Patterson-Hine
NASA Ames Research Center
Mail Stop 244-4
Moffett Field, CA 94035
aph@pluto.arc.nasa.gov

Abstract

Spaceborne computing systems must provide reliable, continuous operation for extended periods. Due to weight, power, and volume constraints, these systems must manage resources very effectively. A fault diagnosis algorithm is described which enables fast and flexible diagnoses in the dynamic distributed computing environments planned for future space missions. The algorithm uses a knowledge base that is easily changed and updated to reflect current system status. Augmented fault trees represented in an object-oriented form provide deep system knowledge that is easy to access and revise as a system changes. Given such a fault tree, a set of failure events that have occurred, and a set of failure events that have not occurred, this diagnosis system uses forward and backward chaining to propagate causal and temporal information about other failure events in the system being diagnosed. Once the system has established temporal and causal constraints, it reasons backward from heuristically selected failure events to find a set of basic failure events which are a likely cause of the occurrence of the top failure event in the fault tree. The diagnosis system has been implemented in Common LISP using Flavors.

Introduction

Most artificial intelligence diagnosis systems developed to date fall into one of two categories: rule-based systems or model-based systems. Rule-based systems, such as MYCIN [1], encode expert knowledge about the diagnosis problem as declarative rules. These systems have been quite successful, but it is difficult to maintain consistency when updating or adding to the rulebase of such systems. Model-based diagnosis systems usually simulate the system being diagnosed and find faults by comparing the simulation results with actual data. The simulations are usually quite slow and the diagnosis problem could become quite complex when multiple faults are present.

When a diagnosis system is used in a dynamic environment, such as the distributed computer system planned for use on Space Station Freedom, it must execute quickly and its knowledge base must be easily updated. Representing system knowledge as object-oriented fault trees provides both features. Changing values in these fault trees is easily accomplished by changing an object's instance variables and using well defined procedures to update related information in the fault tree. Also, reasoning based on knowledge represented in a fault tree is faster than running a system simulation and comparing results. The diagnosis system described here performs its task by reasoning with knowledge contained in an object-oriented fault tree. It is well suited for use in a changing environment since a fault tree can be updated during normal operation so that it accurately reflects the current system status. When a fault occurs, the diagnosis system will have up-to-date information for performing its task.

Object-Oriented Fault Tree Representation

Fault tree analysis can be described as an analytical technique where an undesired state of a system is specified and the system is analyzed, in the context of its environment, to find all tenable ways this undesired state could occur. The resulting information can be represented as a tree structure with the original undesired event (or failure) at the root, the possible causes of that event as the root's children, the causes of those events as their children, and so on. Analysis stops at any given branch of the tree when the event described by the leaf node is fine-grained enough to satisfy the needs of the analyzer. These leaf nodes are called basic events and frequently correspond to failures which are relatively easy to repair or overcome.

All non-leaf nodes of a basic fault tree can be thought of as logic gates representing a logical AND or OR. An AND gate signifies that all the child events of that node must occur before the event represented by the node will occur. An OR gate means if at least one of the child events occurs, the parent event will occur. Sometimes a NOP gate is used when a node has only one child. A NOP gate is just like an OR gate with only one child event. See Fig. 1 for illustrations of fault tree AND and OR gates. Fig. 4 shows a complete fault tree.

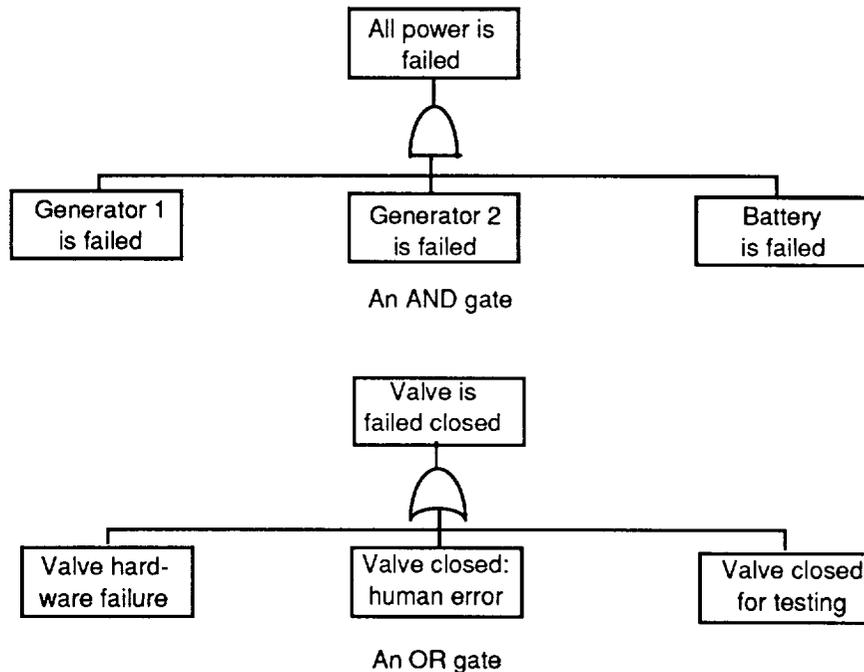


Figure 1: Examples of Fault Tree gates
(From the Fault Tree Handbook [4])

Frequently other information will be associated with each fault tree node. For instance, a node might also contain the probability of occurrence of its associated failure event, or how long it takes between the occurrence of a child event and the occurrence of its parent event. In these cases, the fault tree is called an augmented fault tree [2].

Patterson-Hine [3] has developed an efficient object-oriented fault tree representation and an evaluation procedure which permits dynamic updating and rapid recalculation of values associated with fault tree events. In this representation, each event node in a fault tree is described by an

object with instance variables containing information about the node's parents, children, type, and other details about the failure event represented by that node.

The information in the fault trees used by the diagnosis system includes event importances, C-factors and time intervals. The *importance* of an event is a measure of how critical that event is to the occurrence of the top event of the fault tree. It reflects the relative contribution of that event and all events in the subtree below it to the occurrence of the top event in the fault tree. The *C-factor* associated with a failure event in a fault tree is a heuristic measure of the likelihood that the occurrence of the parent fault of that event was caused by that event rather than by one of its siblings. The *time interval* of an event under an OR gate is an estimate of how much time will elapse from the moment that failure event occurs until its parent failure event occurs. In the case of a child event under an AND gate, the *time interval* measures the time between the moment when all of the child events have occurred and the occurrence of the parent event. The top event (root node) of a fault tree does not have values for importance or C-factor. See Fig. 2 for an illustration of some of the objects in an object oriented representation of the fault tree nodes shown in Fig. 1.

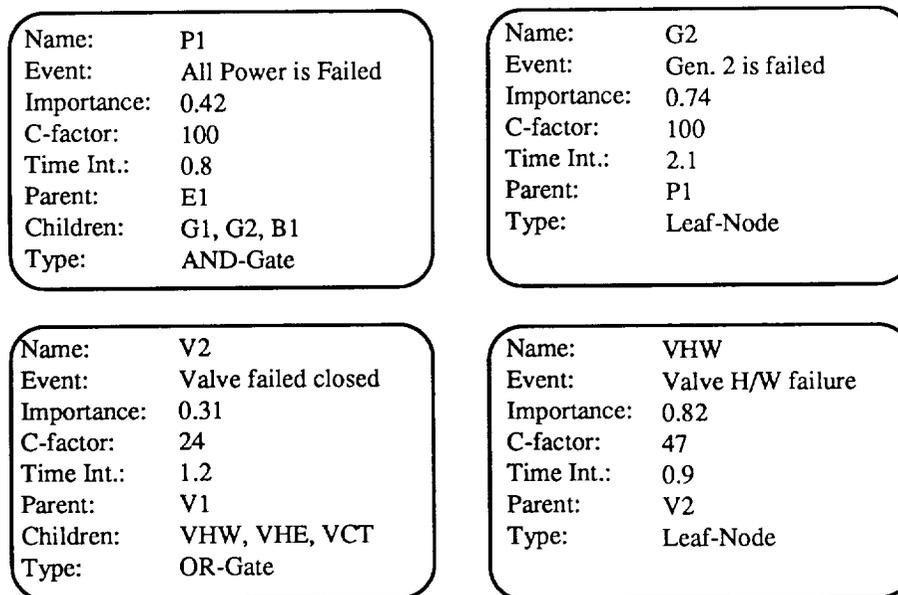


Figure 2: Fault Tree Object Representation

In a dynamic environment, such as a distributed computer system, the probability of occurrence or the importance of a given event may change as the system is reconfigured. Object-oriented representation of the fault tree allows these changes to be made easily and their consequences to be calculated quickly. More traditional knowledge representations, such as if-then rules, do not facilitate updates nearly as well.

The Diagnosis Algorithm

This diagnosis system is based on the failure cause identification phase of the diagnostic system described by Narayanan and Viswanadham [2]. Their system has been enhanced in this implementation by replacing the knowledge base of if-then rules with the object-oriented fault tree representation. This allows the system to perform its task much faster and facilitates dynamic updating of the knowledge base. Accessing the information contained in the objects is more

efficient than performing a lookup operation on an indexed rule base. Also, Patterson-Hine's object-oriented fault tree evaluation algorithm makes dynamic updating viable.

The diagnosis system is given information about the system being diagnosed in the form of alarms. Alarms can be thought of as lights on a panel monitoring the system. A *normal alarm* is a green light on the panel that indicates the failure event it is monitoring has not occurred. An *abnormal alarm* is a red light on the panel indicating that the failure event has occurred. Each possible alarm corresponds to a node in the fault tree. If it is known that the failure event represented by a node has not occurred, that event is placed in the normal alarms set. If it is known that an event has occurred, that event is placed in the abnormal alarms set. Any information about which failure events have and have not occurred is very helpful for diagnosis and can speed up the diagnosis process considerably.

The diagnoses produced by this system are sets of basic failure events that causally explain the occurrence of the top failure event. The diagnosis process is initiated by specifying the failure to be diagnosed, the estimated time of occurrence of this failure, the current set of normal alarms and the time that each normal alarm was last confirmed, a set of abnormal alarms with estimated failure times, and the root node of the fault tree representing the system to be diagnosed.

The diagnosis begins by inferring all failure events that must have occurred and those that could not have occurred based on the information in the normal and abnormal alarm sets. The alarm sets are updated accordingly. The system uses the alarm sets to guide its search of the diagnosis space. It does not consider those portions of the diagnosis space with diagnoses containing sets of basic failure events that would cause the occurrence of a failure in the normal alarms set. Also, those portions of the search space with diagnoses containing abnormal alarms are searched early in the diagnosis process.

The system also checks possible diagnoses for temporal and causal consistency. The time of occurrence information provided for each alarm is used to propagate temporal constraints throughout the fault tree. In order for a diagnosis to be accepted, the occurrence times of the suspected failure events must have a logical causal ordering. In reference 2, Narayanan and Viswanadham give a thorough explanation of the use of temporal constraints.

While it is updating the alarm sets, the system builds a set of starting points corresponding to some of the confirmed failure events. The starting points are chosen in such a way that the combined diagnoses of the events in the starting failures set will be sufficient to explain the occurrence of the top level failure event.

Finally, the system performs backward chaining from the selected starting points to find a set of basic events that were a likely cause of the top failure event. The backward reasoning process first decomposes the starting point failures into sub-failures. Then the sub-failures are further decomposed. At each decomposition, a set of heuristics is used to choose which branches of the fault tree are likely to produce a good diagnosis. The heuristics make decisions based on the contents of the alarm sets, the temporal information associated with the events, the overall importance of each failure event, and the C-factors of the events. This decomposition process continues until basic faults are reached. If the basic faults in the proposed diagnosis do not violate any temporal or causal constraints, they are returned as a diagnosis for the top failure event. If any constraints are violated, the algorithm backtracks and searches other branches of the fault tree for possible diagnoses. The basic algorithm and heuristic descriptions can be found in Narayanan and Viswanadham [2]. This implementation has modified those algorithms to use the object-oriented fault tree knowledge representation instead of the rules suggested by Narayanan and Viswanadham, but the basic diagnostic procedures are the same as those stated in that paper.

Diagnosis Examples

The following examples show diagnoses of the simple adder circuit shown in Fig. 3. The top level failure event for the adder fault tree (see Fig. 4) is incorrect adder output. In this tree all

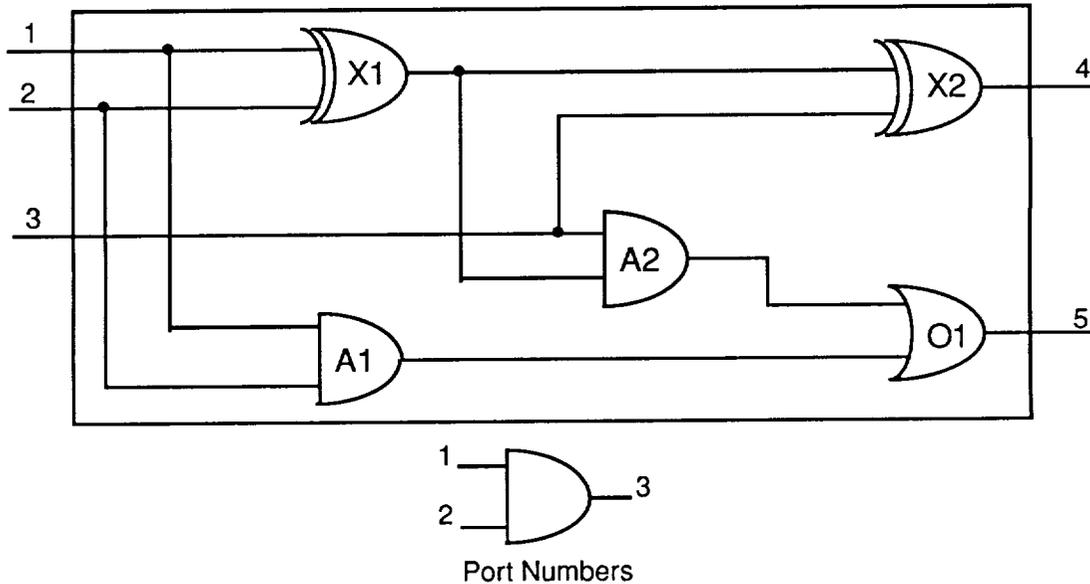


Figure 3: Simple Adder Circuit

basic events are seen as equally important (if any failure occurs, the system fails), and the fault propagation time intervals are all the same except for operator input error. Also, the contribution of a failed gate was heuristically judged to be slightly more important than the contribution of errors propagated from other sources in the circuit (hence the higher C-factor). This loosely accounts for facts such as if one input to an AND gate is zero, the validity of the other input is irrelevant. This also results in diagnoses favoring gates nearer to the circuit output. Note that the diagnosis system does not always return all possible causative faults for the top level failure, just those that seem very likely.

Example 1:

```
(find-failures '(incorrect-result adder) ; Top Level Event w/
1 ; Time of Occurrence
'(((incorrect-operator-input adder) 1) ; Normal Alarms
((incorrect-port-input adder port1) 1)
((incorrect-port-input adder port2) 1)
((incorrect-port-input adder port3) 1)
)
'(((incorrect-port-output adder port4) .98)) ; Abnormal Alarm
'adder ; Root of Fault Tree
```

Diagnosis: (((FAULTY-GATE X2) 0.96))

The top event is an incorrect adder result. The normal alarms set indicates that the operator input was correct and arrived at the input ports without any problems. The abnormal alarms set reports that we discovered an incorrect output value on port four 0.02 seconds before diagnosis began. The diagnosis is that Gate X2 is faulty since it connects directly to output port 4 and, due

to the C-factors, gate failure is more likely to cause bad output than propagated errors. The numbers included in the diagnoses are the latest times that the suspect faults could have occurred in order to cause the top level failure.

Example 2:

```
(find-failures '(incorrect-result adder) ; Top Level Event w/
  1 ; Time of Occurrence
  '(((incorrect-operator-input adder) 1) ; Normal Alarms
    ((incorrect-port-input adder port1) 1)
    ((incorrect-port-input adder port2) 1)
    ((incorrect-port-input adder port3) 1)
    ((faulty-gate X2) 0.96)
  )
  '(((incorrect-port-output adder port4) .98) ; Abnormal Alarms
    ((incorrect-port-input O1 Port2) .97))
  'adder ; Root of Fault Tree
```

Diagnosis: (((FAULTY-GATE A1) 0.95) ((FAULTY-GATE X1) 0.94))

This is like Example 1, but (Faulty-Gate X2) has been added to the set of normal alarms, indicating that the X2 failure did not occur, and the failure event (incorrect-port-input O1 Port2) was added to the abnormal alarms list. Having X2 in normal alarms sent the system a little deeper into the fault tree to find that gate X1, which feeds through X2 out to port 4, was faulty. (Faulty-Gate A1) was also added to the diagnosis to account for the incorrect input to gate O1.

Conclusions

This diagnosis system based on fault tree models has several advantages over other diagnosis systems. It does not rely on a single fault assumption in its reasoning. In fact, most diagnoses will include many basic faults. It deals with temporal parameters and uses them to effectively reduce the diagnosis search space. Probabilistic reasoning is used to a certain extent in selecting rules and search paths. The inclusion of event importance factors allows the system to consider the most heavily weighted search paths first. The use of C-factors includes some expert knowledge in the diagnosis procedure. The diagnosis technique is not domain specific. If a fault tree can be constructed for a given system, this diagnoser can diagnose that system.

One shortcoming of this system is that the diagnosis is only as good as the fault tree model. If an incomplete, undetailed, or inaccurate fault tree is given to this system, the resulting diagnoses will be equivalently incomplete, undetailed, and inaccurate. This is in contrast to a diagnoser which models the system as a simulation. The trade off here is that the simulation model will probably be domain specific and incapable of expressing as rich a diagnosis environment as a fault tree. Also, the simulations would have difficulty including probabilistic and heuristic knowledge in their diagnosis, and diagnosis using simulation is usually quite slow.

Future Work

There are plans to integrate this diagnosis system into a fault tolerant distributed system. The overall goal of this work is to reduce the amount of redundancy needed for fault tolerant spaceborne computation. Distributed systems, such as those planned for Space Station Freedom, will be characterized with fault trees and Markov diagrams. As the distributed system runs, the

system knowledge base will be updated to reflect the current status of the system. When faults are detected during execution, the diagnosis system will be used to determine the basic causes of the faults. Then the system load will be redistributed to avoid the faulty components and the computation will continue. This work will be done on the advanced architecture testbed and fault tolerant computing testbed at NASA-Ames Research Center.

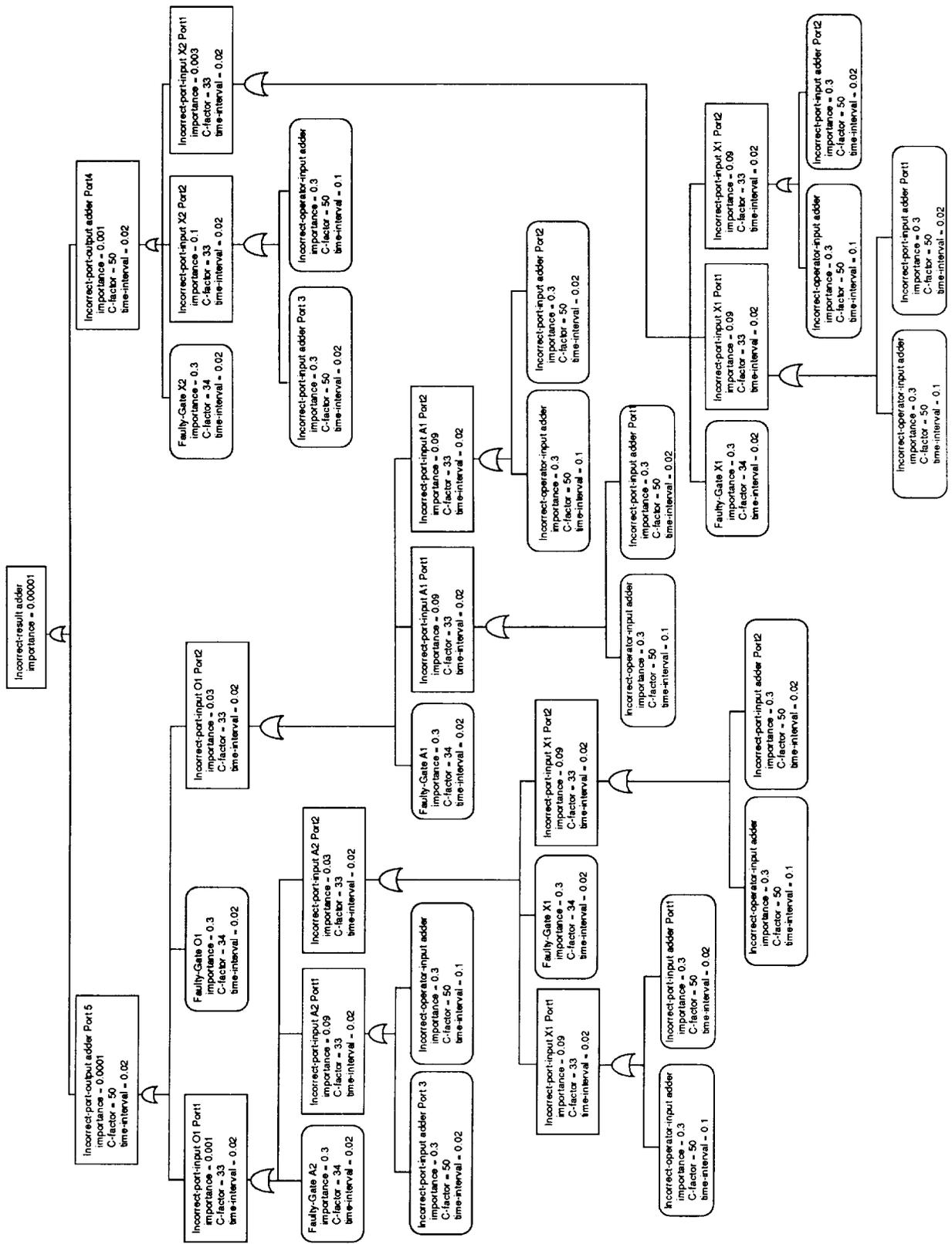


Figure 4: Fault tree for simple adder circuit shown in Fig. 3.

References

1. Buchanan, Bruce G., and Edward H. Shortliffe, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, Addison-Wesley, Menlo Park, CA, 1984.
2. Narayanan, N.H., and N. Viswanadham, "A Methodology for Knowledge Acquisition and Reasoning in Failure Analysis of Systems," *IEEE Transactions on Systems, Man, and Cybernetics*, pp. 274-288, March/April 1987.
3. Patterson-Hine, F.A., Object-Oriented Programming Applied to the Evaluation of Reliability Fault Trees, Ph.D. dissertation for The Univ. of Texas at Austin, May 1988.
4. Vesely, W.E., F.F. Goldberg, N.H. Roberts, and D.F. Haasl, Fault Tree Handbook, NUREG-0492, U.S. Nuclear Regulatory Commission, Washington, D.C., 1981.

A MODEL FOR A SPACE SHUTTLE SAFING AND FAILURE-DETECTION EXPERT**Daphna Zeilingold and John Hoey****Rockwell International Corporation
Space Transportation Systems Division
12214 Lakewood Blvd., Downey, CA 90241****ABSTRACT**

The safing and failure-detection expert (SAFE) is a prototype for a malfunction detection, diagnosis, and safing system for the atmospheric revitalization subsystem (ARS) in the Space Shuttle orbiter. SAFE, whose knowledge was extracted from expert-provided heuristics and documented procedures, automatically manages all phases of failure handling: detection, diagnosis, testing procedures, and recovery instructions. The SAFE architecture allows it to handle correctly sensor failures and multiple malfunctions. Since SAFE is highly interactive, it was used as a test bed for the evaluation of various advanced human-computer interface (HCI) techniques. The use of such expert systems in the next generation of space vehicles would increase their reliability and autonomy to levels not achievable before.

INTRODUCTION

The fault detection, isolation, and recovery (FDIR) process on board the orbiter is lengthy and tedious (Figure 1). The crew is surrounded by diverse inputs: gauges, displays, warning lights, alarms, caution and warning messages, and ground control communications. If an anomaly occurs, the crew must follow flow chart-like malfunction procedures (essentially fault trees) to isolate the problem and reconfigure the vehicle to a safe state. The crew must locate the appropriate malfunction procedures in a voluminous manual, then find the cockpit switches that must be manipulated—and do it all in a timely manner in an emergency.

Automation of this process clearly would have many benefits. It would speed up fault isolation and make it more consistent and reliable; it would reduce the crew's work load and alleviate the loss-of-efficiency problem on longer missions; and it could, as confidence in the system is gained, reduce training and ground support requirements.

Beyond its immediate benefits, such a system would introduce into space vehicle management a new technology with the potential for substantially increasing vehicle autonomy—autonomy that will be essential when space flights such as the Mars mission are undertaken. Because of the communication time lag, complete responsibility for vehicle "health" will have to be assumed by on-board functions, and mission length and complexity will make automation mandatory.

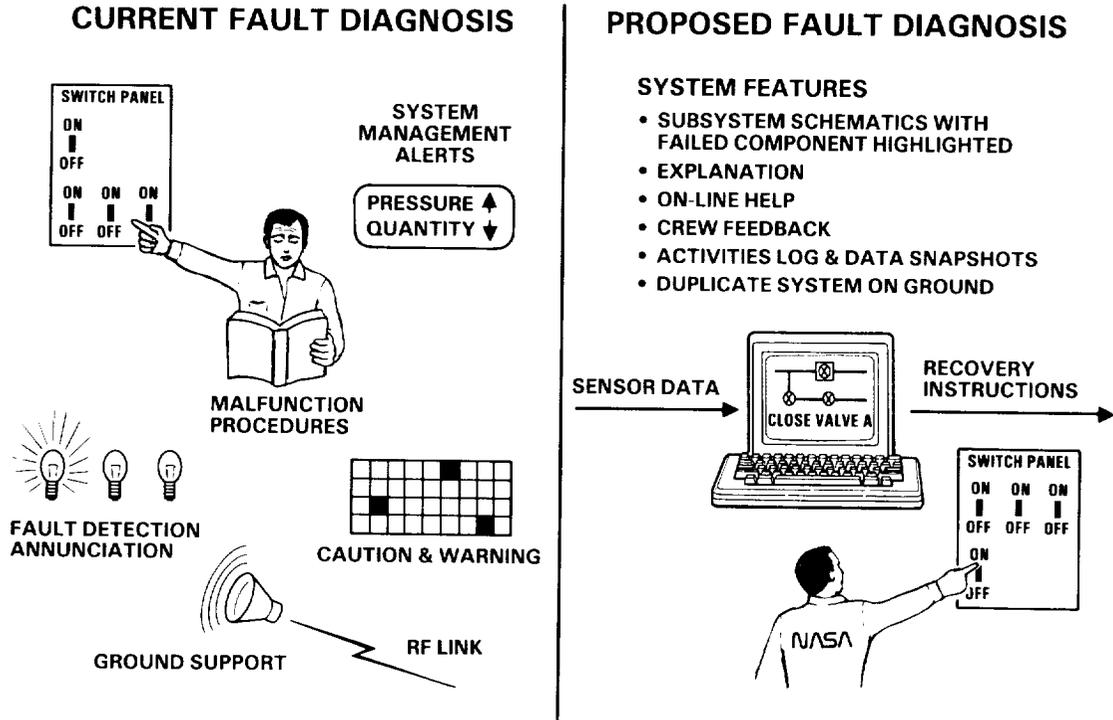


Figure 1. Comparison of Current and Proposed Fault Diagnosis

The SAFE project's objective was twofold: to investigate the use of expert system technology for on-board malfunction diagnosis through the development and prototyping of the software architecture; to experiment with various user interface techniques that would provide easy and effective crew interaction.

SAFE ARCHITECTURE

The paradigm used by SAFE for diagnosis is the "shallow-reasoning" expert system, which employs expert heuristics that map sensor signatures to component malfunctions. Its advantages are that it is fast, conceptually easy to understand, and lends itself efficiently to a rule-based structure. It reflects the way diagnosis is now done: a mixture of malfunction procedures and the expertise of the subsystem people on the ground who advise the crew. A disadvantage, which requires further research, is that such a system cannot detect an unanticipated failure, since it has no understanding on any level of how the modeled system works. (An attempt to do diagnosis with a model-based paradigm was deferred because of severe performance problems.)

As shown in Figure 2, components of the SAFE system are the knowledge base (rules and facts), the user interface (UI), and the ARS simulator (SIM) and the simulator interface (SI). The inference engine is OPS83, which is a forward chainer—i.e., data-driven. The user interface is written in Ada and communicates with OPS83 via VAX mailboxes.

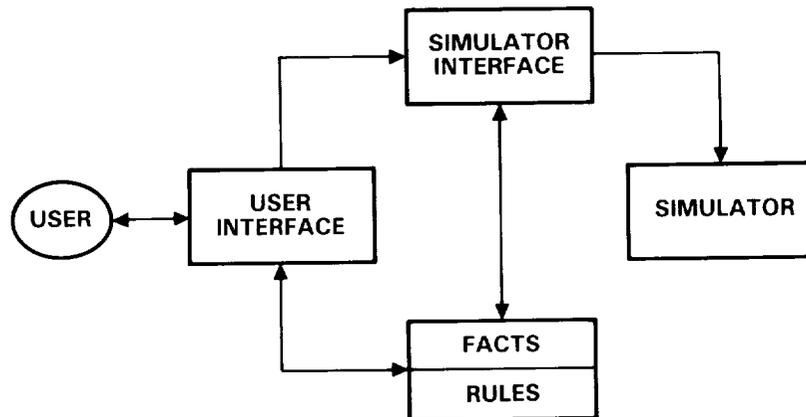


Figure 2. SAFE Overall Architecture

ARS Model

The system model used by SAFE represents a simplified view of the ARS of the Shuttle orbiter environmental control and life support system (ECLSS). ARS was chosen because of its criticality and because it is both complex enough to provide a variety of malfunctions and simple enough to be easily understood by knowledge engineers.

The system maintains a breathable atmosphere, with total pressure about 14.7 psi and partial oxygen pressure between 2.9 and 3.4 psi. The main components of the model are the oxygen and nitrogen supply, check valve, sensors and controller for the partial pressure of oxygen (ppO₂), and the control valve and its actuator. When the ppO₂ exceeds 3.4 psi, it is detected by the ppO₂ sensors, and the ppO₂ controller causes the control valve actuator to switch to the OPEN position. This opens the control valve, and nitrogen flows into the cabin regulator. The nitrogen applies reverse pressure to the check valve, stopping the oxygen flow. Crew breathing depletes the existing oxygen, and eventually the partial pressure drops below 2.9 psi. The controller valve is then closed and the nitrogen flow stops, enabling oxygen to enter the cabin (Figure 3).

The failures that can be detected by the existing prototype are in the ppO₂ sensors and controller, the control valve, the oxygen and cabin regulators, the oxygen pressure sensor, and the cabin pressure sensor.

SAFE Operation

As long as no failure is detected, the user interface displays the sensor values on a user-oriented display. For prototype verification purposes, a software simulator for the ARS was written by using the procedural capabilities of OPS83. The simulator supplies the sensor data. Twelve malfunctions have been incorporated into the simulator, and the demonstrator can choose and inject one through the simulator interface (refer to the simulator section).

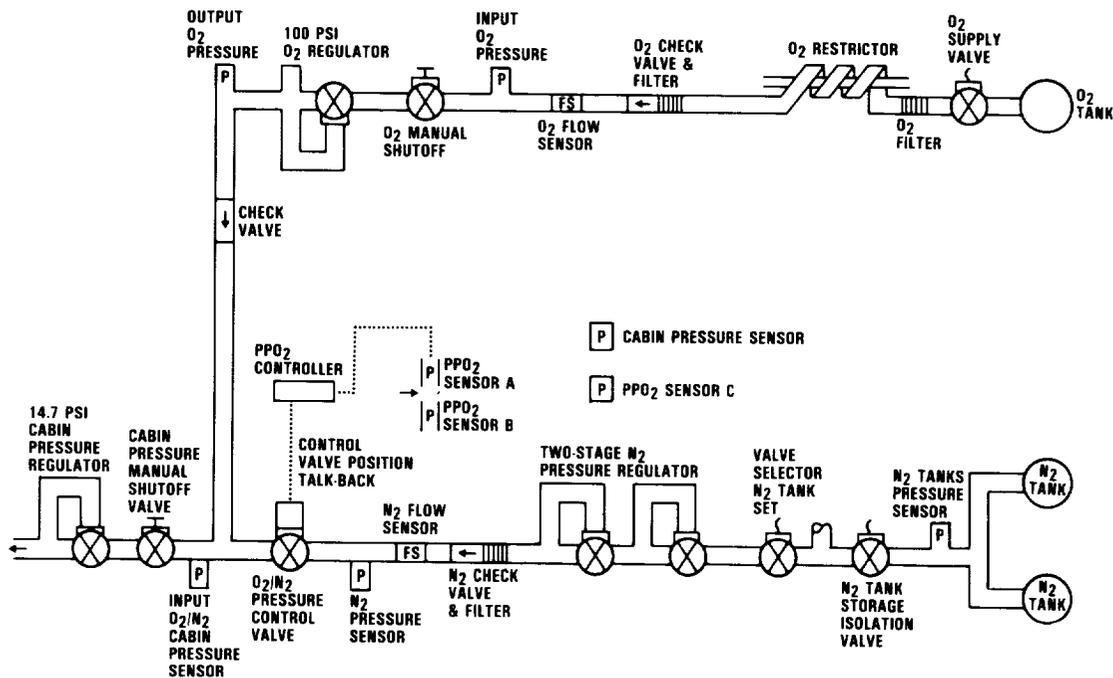


Figure 3. ECLSS O₂/N₂ Pressurization System

Once the expert system has detected a malfunction, it notifies the user, instructing him on how to perform a safing procedure to restore the system to an operational state. If, however, the symptoms are insufficient to uniquely identify the failure, the user can choose to run tests to isolate the malfunction further. The appropriate steps for a test procedure are then displayed. The expert system receives information like acknowledgments of actions taken or requests for explanations through the user interface, which runs as a separate process from the expert system. The expert system can continue to monitor sensor data while the user interface is engaged in man-machine dialog.

SAFE Diagnosis Mechanism

A complete and well-defined architecture for a shallow-reasoning diagnosis and safing system now exists. The entire process can be described as a cycle through the phases illustrated in Figure 4. The diagnosis activity is separated into sensor and nonsensor device failure detection phases so that sensor problems can be found before the readings are relied upon to diagnose other failures. The process involves the following steps:

- The sensor diagnosis rules are enabled. If a sensor is determined to be faulty or is suspected of being faulty, it is so marked. This stage can generate a diagnosis for a failed sensor.

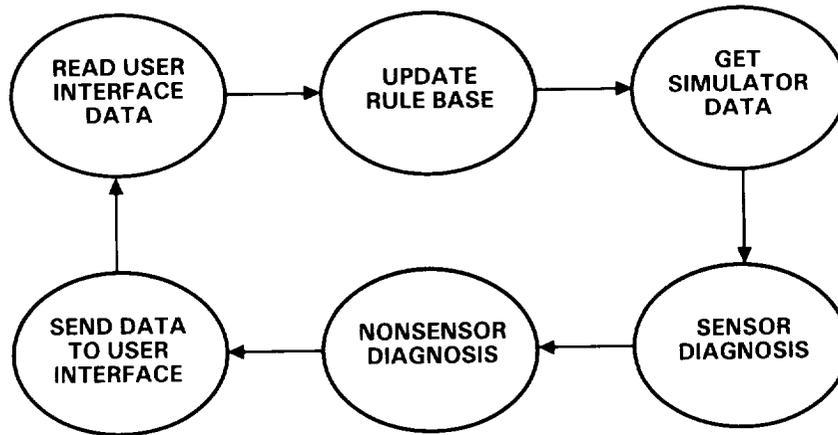


Figure 4. Major Phases of Rule-Based Architecture

- Next, the nonsensor diagnosis rules are enabled. Any rules that base their diagnosis on readings from a sensor previously marked as possibly faulty are not enabled, so no diagnosis is made on the basis of unreliable sensor data. This stage can generate a diagnosis for a failed component.
- A diagnosis can be “final,” meaning that the system has made as detailed an isolation process as can be expected, or it can be “intermediate,” meaning that tests can narrow it further. The diagnosis then lists all the suspected components and their potential failure mode.
- If the diagnosis is not a final one and the user signals his interest in additional tests, test rules are enabled to create a test procedure. The test procedure is a series of operational instructions to the crew, the purpose of which is to reconfigure the subsystem in such a manner as to permit further isolation of the fault.
- Once the test is completed, the resultant sensor data, coupled with knowledge of the prior diagnosis, enable the system to determine which component really failed.
- If the diagnosis is as specific as possible or if the user chooses not to run any tests, safing rules generate the appropriate safing procedure, which is a series of instructions on how to reconfigure the system to ensure safe vehicle operation, with emphasis given to a safe return home.
- The expert system creates the test and safing procedures dynamically, going from the current system state to the desired one (e.g., it will not instruct to close a closed switch). It also takes into account the flight history (e.g., if the backup system were activated due to a previous failure, there would be no backup system left and the emergency supply system would have to be used).

- The user has to acknowledge the completion of every step of the testing or safing procedure.
- When all safing actions are completed, the system suspends the diagnosis process to permit all sensor values to converge to their nominal ranges. Otherwise, the failure that just occurred might be reannounced, since typically the sensor values would be out of range for a while. This step is called “system stabilization.” The approach is too simplistic; should another failure occur, the user would not know about it until the system is stabilized. Other solutions should be investigated.
- A mechanism for handling certain multiple malfunction situations is introduced. The malfunctions are given two severity ratings. Throughout the diagnosis process, all diagnosis rules for failures of similar or less criticality than that of the current diagnosis (i.e., the diagnosis being handled) are disabled. If a more severe failure is detected, the current diagnosis is replaced by the new one and the user is instructed to take care of that one first. If, after the safing, the “old” failure is still valid, it influences the sensor readings and eventually is rediagnosed by the expert system. In many cases, a switch is made to the backup system, and failure of a component of the now-inactive system does not manifest itself. However, this approach works only for failures whose signatures are totally independent of each other.

The knowledge base also contains a representation of the systems using OPS83 memory elements. Employing this model, the expert system keeps track of the system’s state. It is separate from the simulator’s model, and only the sensor information is copied from the simulator to that model via the simulator interface. Thus the rule base has no access path to the failure information in the simulator.

User Interface

The user interacts with SAFE and the simulator via a multilevel menu structure. Communication is effected through the selection of menu and support icons using touch-panel technology.

The user is able to monitor the ARS by viewing either a performance display or a schematic display. The performance display provides a high-level picture of the system status by displaying five color-coded dials that show the status of five critical parameters, by the numerical values indicated as well as by the color of the needles as they swing into the various safety levels on the dial faces. Also, several less critical parameters are represented in tabular form.

The user may also view the status of the same parameters in a more hardware-oriented manner by selecting a schematic view of the system. In this mode, the data appear alongside the components of the relevant subsystem (in this prototype, the ARS), allowing the user to view the physical relationships among the various components.

The user can choose any of three system logs by selecting the appropriate icon: the Status Log, which contains messages relevant to the status of the entire system; the Event Log, which logs any change of state of the ARS subsystem; and the Message Log, which records all communications between the system and the user. The logs are time-stamped and, when relevant, color-coded to indicate the severity of the message. Between the graphic displays and the different logs, the user has an opportunity to choose the level of detail that suits his needs. If he is particularly interested in the ARS performance, he can look at the schematic view; but if he is busy with other mission tasks, he may wish to see only the information in the Status Log.

When a failure is detected by the expert system, the icons necessary to handle the situation appear on the screen, a color-coded message appears in the message window at the bottom of the screen, and the DECTalk device annunciates the message. If the message is of low severity, it is displayed in amber and annunciated once. However, if it is of high severity, it is displayed in red and annunciated until the user acknowledges the message. At this point, the user may do one of two things: safe the system or, if possible and time permits, test the system to isolate the cause of the malfunction.

When the decision is made to safe the system or to perform a test before safing, the user is guided every step of the way via the DECTalk, printed messages, and schematic displays of the relevant cockpit panels with highlighted switches. If at any point the user wants to know the motivation for the actions being recommended, an explanation icon is available. The ensuing explanation is printed in the message window but is neither annunciated by the DECTalk nor recorded in the Message Log.

The communication scheme employed by the user interface to interact with the expert system and SI is based on message passing. The first field of each message is a type field that guides the message's interpretation by its receiver. This allows the message format to be flexible and extensible.

Simulator and Simulator Interface

The simulation model, though simplistic, yields results sufficient to provide a good test bed for the prototype. It produces the values of 13 sensors, emulating the gradual changes and fluctuations that characterize flows and pressure measurements.

When a demonstrator chooses to inject a failure into the system, the failure information is transmitted from the user interface to the simulator interface, where it is converted into the relevant simulator parameters. The simulator iterates constantly, creating sensor values and logging them in the knowledge base, where the expert system can monitor them. Once the simulator creates the right signature (which could take a number of cycles, since the simulator gradually converges on the right sensor values), the proper rules generate the failure diagnosis. The expert system and the simulator interface are in constant communication with the user interface via the VAX mailboxes.

Since the simulator is not a real part of the diagnosis system (existing only for testing purposes), it was decided to completely isolate it from the other components. On the other hand, it was desirable to separate the simulator from the details involved in its communication with the rest of SAFE. This approach allows the communication functions to be isolated into a single module—the simulator interface module—that must be rewritten in order to interface with real hardware. The module is responsible for converting relevant user inputs into data the simulator can manipulate and reporting the simulated sensor readings to the expert system. The expert system has no direct access to the simulator, so it is easy to verify that it gets no data it would not receive under real circumstances.

Host Hardware and Software

The knowledge base, simulator, and simulator interface are implemented in OPS83, a language targeted for rule writing. It has good pattern-matching capabilities, allows user customization of the inference engine, and, because it is compiled, runs at a fairly high speed. Having the simulator also written in OPS83, with its procedural capabilities, permits fast and easy interaction between the simulator and the knowledge base.

The user interface is written in Ada. It runs on a Tektronix 4128 high-resolution graphics work station connected to a VAX 11/785. An Elographics touch panel allows the user to communicate with the expert system via an icon-driven menu. This serves as the sole input device. A Digital Equipment Corporation DECTalk voice synthesis unit announces all messages that appear on the screen.

SUMMARY AND CONCLUSIONS

The architecture described in this paper could be used to automate the diagnosis and safing process for any subsystem of the orbiter by replacing the safing and diagnosis rules. Given a good and cooperative expert, this would be a relatively straightforward process. It has some basic capabilities for handling complex situations. Faulty sensor detection, done before the nonsensor devices diagnosis, precludes detection based on erroneous data. The rating and failure-interruption scheme is a first attempt to address the issues of multiple malfunctions detection. The creation of dynamic test and safing procedures enables the expert system to save time and minimize errors that could occur when redundant actions are possible. Dynamic information exchange with the user interface gives the system flexibility and room for expansion.

Many aspects of expert system technology application to fault detection, isolation, and recovery require more investigation. A primary area of research is that of cooperative expert systems that monitor all vehicle subsystems and exchange information. Such an architecture must be developed before the complexity of the whole vehicle can be mastered.

Some diagnosis capabilities are still lacking. Among these are better handling of multiple failures, mainly the inclusion of failures whose signatures interfere with each other; refinement of the postsafing monitoring process; intermittent failure diagnosis; and recognition of unexpected anomalous conditions that have not been encoded explicitly, at least to the extent of notifying the user that the vehicle is not operating correctly and that the problem cannot be diagnosed by the expert system.

The SAFE prototype has demonstrated the feasibility and viability of the technology beyond just Space Shuttle applications. The same concepts can be applied to any on-board automated FDIR system. This technology is destined to become a key component in the design of next-generation space vehicles.

**OBJECT ORIENTED FAULT DIAGNOSIS SYSTEM
FOR SPACE SHUTTLE MAIN ENGINE REDLINES**

John Rogers
Senior Research Associate

Saroj Kumar Mohapatra.
Graduate Research Assistant

Johnson Research Center
University Of Alabama In Huntsville

ABSTRACT

A great deal of attention has recently been given to Artificial Intelligence research in the area of computer aided diagnostics. Due to the dynamic and complex nature of space shuttle red-line parameters, a research effort is under way at the Johnson Research Center on the campus of UAH to develop a real time diagnostic tool that will employ historical and engineering rulebases as well as sensor validity checking. The capability of AI software development tools (KEE & G2) will be explored by applying object oriented programming techniques in accomplishing the diagnostic evaluation.

What is an Expert System?

In its simplest sense an "Expert System" is a system which can handle any domain limited situation as efficiently as a human expert would handle it. The keywords here are domain limited and human. Here is a simplified example. It might be easy to conceptually perceive a system that handles the breaking mechanism in an automobile. Now suppose these brakes are hydraulically activated as are some of the brakes in large trucks. If the system suddenly lost pump pressure due to some type of power system failure, the system should be expected to determine the element in its system (namely the pump) was malfunctioning, but probably would not be able to totally diagnosis the problem as a power failure. The power system is a separate entity in the automobile and therefore does not fall into the hydraulic systems limited domain. Theoretically the domain of an expert system may be as large as the designer desires; however, the larger the system domain, the larger the knowledge base requirements and the more complex the inference engine.

The second keyword, human, details the performance requirements of the system. The system must be able to handle a situation that exists within its limited domain as efficiently as a well trained expert in the field. Though it can not be has not yet begun to reach its size or pattern matching potential. Also, since the system is currently operating as a stand-alone system and no sensor input is available, simulation and data file supplied values are driving the system. Output is restricted to operator notification and suggested corrective procedures by the system. Scan Intervals are well within one second, yet far from the ultimate goal. These limitations are being slowly reduced and eventually it is hoped they will be removed entirely.

expected to determine the faulty power supply, a properly designed expert system should immediately identify the pump as the source of the localized problem. Of course, the system is not doing anything that a human can not do. But for large problems that involve many parameters the system, if properly designed, can reach the conclusion much faster and free the human expert to be accomplishing other tasks.

CREATING AN EXPERT SYSTEM

An expert system has three major modules. These are the systems interface with its environment, the knowledge base of rules and information governing the system, and the user interface.

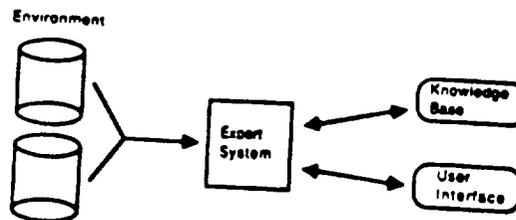


Figure 1

Each of these modules plays an important role in the operational efficiency of the overall system. If the interface between the environment and the system is inadequate the system will be unable to properly monitor the environment's condition. If the knowledge base is not complete or contains erroneous information the system will reach improper conclusions. And finally the system must be designed so it is easy for the individual to use. Each of these topics will now be presented.

Interface between the System and the Environment

Depending upon the type of expert system under development the interface between the environment and the system may take on different forms. For example, if the system is being designed to maintain certain operating parameters in a particular piece of computer software, the interface may include a communications protocol between the two machines (if they are in fact located on separate machines). Or possibly for a mechanical system, the interface between the mechanism and the expert system monitoring it may be a connection to an analog converter that takes in raw sensor voltages and converts it into meaningful data. Any combination of these forms is possible. In different systems throughout industry a variety of interfaces have been designed to fit the needs of the application.

Regardless of the type of interface, the primary objective is to have as complete a communication between the environment and the system as possible. Erroneous results would be produced by a system that has the potential of functioning properly due to simply bad input data. Whatever needs to be included in the interface to insure this communication clarity must be included. Obviously this is a critical necessity for the system and must be satisfied in order for it to monitor the environment correctly.

It is the job of the knowledge engineer to acquire a complete set of facts and rules that govern the environment. This is accomplished in a variety of ways. First, many processes follow a given set of engineering and natural laws. The applicable formulations of these laws can be

incorporated into the system. Previous or historical data can be interpreted and stored. And learned and experienced individuals can be interviewed to gain from their knowledge.

The interview method is the most difficult to perform, but often the most beneficial. It is easy to see how valuable it is to have experience with a certain type of problem. Many times a process can be completed rather routinely by an individual, but it is time consuming and takes the individual away from other much needed tasks. This is a perfect example of where the interview method would readily apply. However, the familiarity with the problem can become a mixed blessing. Crucial steps in the performance of a task become routine to the expert and might be omitted in the performance description. Therefore leaving the system designer with an inadequate instruction set.

Knowledge Base

Perhaps the most difficult task facing the developer of an expert system is constructing the knowledge base. The knowledge base is a collection of facts and associated rules (sometimes called the rule base) that are connected with the environment of the expert system. These rules and facts are based on information gathered by a process called Knowledge Engineering.

Most tasks that humans perform are complex in nature, yet they seem quite simple to the individual. For example, consider the everyday activities that an ordinary individual might find themselves doing such as brushing their teeth, getting dressed, or preparing a meal. Certainly most of us know how to perform each of these simple tasks quickly and easily; however, it is not quite as simple a problem to describe or explain all of the steps that are involved. If one of the steps is omitted the total process is likely to fail. This knowledge engineer must familiarize himself with the environment that is being monitored. Only then can a successful dialog between himself and the experts occur. It is his responsibility to question these experts in a manner such as to gain a complete set of facts and rules governing the specific environment.

User Interface

The user interface is the individuals gateway to the expert system. One of the most often heard complaint about any software package is the inadequacy of the user interface. A user interface should be as simple to **use as possible**, yet have the flexibility to allow the operator the ability to handle virtually any situation.

There are many different formats for the user interface. It can be menu driven or allow for command input lines. It might include mouse sensitive items, help documentation, and meaningful error messages for the user. Any or all of these items can help to make a quality user interface. However, remember the most important aspect is usability.

Current Application - S.S.M.E. Redline Diagnostics

The current application involves the monitoring and diagnostic analysis of a group of flight parameters for the Space Shuttle's main engine known as the redlines. These parameters include the High Pressure Oxidizer Preburner discharge intermediate seal purge pressure, the High Pressure Oxidizer Turbopump turbine discharge temperature and secondary seal cavity pressure, the High Pressure Fuel Turbopump turbine discharge pressure, High Pressure Fuel Preburner coolant liner pressure, and the Preburner shutdown purge pressure.

This system is divided into three subsystems: a monitoring system, a rule based diagnostic system, and a historical data base of past performance. Each plays an important role in emulating an expert in the identification and evaluation of possible operating problems.

The monitoring system is similar to those in place in many applications around the globe. It receives sensor data that it compares to a **given** set of parameter limits. If the data falls within the acceptable range for each of the redline parameters, then operations continue unaffected. However, if the data does not remain confined between the operational limits, flags are set which indicate a problem has occurred. At the completion of a given cycle of data entry, these flags are examined. If they indicate possible problems the diagnosis procedure is initiated.

The diagnostic system is a rule based system that forward chains or background chains through a set of rules and hopefully arrives at a possible conclusion for the situation. For the SSME redlines these rules consist of engineering and interparameter relationships which must be maintained. As a conclusion the system may either initiate an attempt to correct the situation and/or report the probable causes and suggested ways to proceed to the operator.

The diagnostic system should also have a built-in mechanism for validity checking on data. This is accomplished in this system in two ways. The rule base has certain rules which define physical laws. If according to these rules all but one **dependant** parameters abide by these laws or if data trends for parameters like pressure and temperature exhibit sporadic almost non-continuous behaviour then there exists a high probability that the sensors are not functioning properly.

The third sub-system involves pattern matching current data to past experiences. This gives the system a type of historical prospective on the current events and may indicate the cause of the current situation based on the causes of the similar historical event. This sub-system is initiated only if the diagnostic subsystem was unable to arrive at a plausible conclusion. If initiated, however, the current events are compared to a data base of past events and a ranked list of possible situations along with a weight factor that describes the degree of parameter matching is returned.

Prototype Development

The original system was prototyped using a software development tool named KEE (Knowledge Engineering Environment) which is a product by Intellicorp. It was menu driven and used graphical displays to relate the information to the user. KEE proved to be a good starting place for the early development. However, time is of primary importance in this system. Each scan interval for the system is desired to be less than 60 microseconds. Therefore, an alternate method had to be explored. G2, a real-time expert system, software development tool developed by Gensym, was selected for the second development.

What is G2

G2 is an advanced tool for developing real time expert systems. Real time expert systems are programs that can respond intelligently to events as they occur. The need for such systems is evident in the process and manufacturing evident in the process and manufacturing industries, telecommunications, medical care, aerospace, robotics, and so on. At the heart of G2 is its ability to do the following:

.G2 can scan an application (like an human operator) and can focus on key areas when it detects potential problems or opportunities.

.G2 can control events in a continuously changing application. This is possible because of a real time inference engine that can maintain validity intervals and history for variables.

.G2 can respond to events when they occur (without having to continually poll sensors, for example.)

.G2 allows the application experts to create prototypes rapidly using structured natural language in an intuitive, graphically oriented environment.

Idea of developing a expert system using G2

To use G2, the developer first **describes** each class of object in the application, what it looks like, and what its attributes are. After describing the classes of objects that are found in the application, the developer can create a model of the application by placing objects on a workspace and connecting them to show their relationships. Associated with each object is a table that describes the object. G2 automatically creates this description from the definition of the object class. After drawing the schematic, the developer writes rules that indicate how to respond and what to conclude from changing conditions within the application. The developer enters these rules and all other statements within G2, in structured, natural language using a context-sensitive editor that guides him through each part of the grammar.

When the knowledgebase is running, G2's real time inference engine uses the rules, together with data that it receives from the data server (the sensor, G2 simulator, and other external sources) to infer how to respond to conditions.

The data server attribute indicates where G2 should go to get values for the variables. The inference engine scans key rules at rates associated with each rule. It focuses on key objects by trying rules associated with the objects. The inference engine invokes rules of a particular category for a particular class of objects. It backward chains to other rules to find values and forward chains to rules when values are found.

The knowledgebase uses the G2 simulator to simulate values for sensors and other variables while the knowledge base is running. It can solve algebraic difference and first order differential equations.

The developer also can create operator controls like check boxes, radio buttons, and action buttons that an operator will be able to use to enter values or to give instructions to G2. The developer can create displays like graphs, dials, meters or readout tables.

The development of a knowledge base usually proceeds incrementally. To develop a full-sized application, he proceeds in stages, at each state refining the prototype, adding a little more to the knowledge base and testing the result. After the knowledge base is built, the developer can interface to an application using "Gensym Standard Interface." The expert system can then receive values from sensors or other sources, set the values of setpoints, write to databases, and so on.

Example of - G2 Redline Implementation

The project required implementing eight parameters: Preburners S.D., HPOP (High pressure oxidizer pump), HPOT (High pressure oxidizer turbopump), HPFT (High pressure fuel turbopump), HPFP (High presusre fuel pump), LPOP (low pressure oxidizer turbopump), LPFP (Low pressure fuel turbopump) and POGO. After creating a workspace those redline parameters are implemented using different kinds of objects(Figure 1). A superior class object is defined so that redline parameters can inherit its attributes whenever they need.

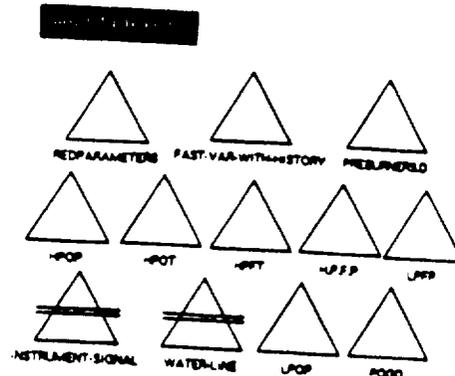


Figure 2

G2 has the capabilities to create tables (Figure 3) for each object For example in Figure 3, HPOT is the object and the table has user restrictions, where the user can describe the restrictions needed for HPOT; HPOT is the class for which also the superior class is redline parameters. Attributes specific to the class HPOT are discharge-temp (a quantitative variable), sealactivity pressure (a quantitative variable), OK (a logical variable), and high (a logical variable). The capabilities and restrictions of HPOT are "none", change is "none", and menu option is "final menu choice." In this case no attributes are inherited, so inherited attributes get the value "none." Default settings for variables are also "none." Notice the attribute stubs in Figure 3. There is a water line output-port located at left 20; a connection input-port located at right 20. To use the connection between the instances of objects the user has to define an object called water-line or instrument line. So as shown on Figure 3 the instrument signal and water-line are defined. The user can define an instance out of the objects. The last four attributes describe the icon which is used to display the instance of the object HPOT and can modified to the designer's specifications.

Notes	OK, and note that this is one of 2 default items named legal
User restrictions	none
Class	HPOT
Superior class	redparameters
Attributes specific to class	discharge-temp is given by a quantitative-variable; sealactivity-pressure is given by a quantitative-variable; ok is given by a logical-variable; high is given by a logical-variable
Capabilities and restrictions	none
Change	none
Menu option	a final menu choice
Inherited attributes	none
Default settings for variables	none
Ports	a water-line output-port located at left 20; a connection input-port located at right 20
icon height	50
icon width	40
icon description	area (0, 0) (20, 40) (20, 0) area (20, 0) (20, 40) (40, 0) area (40, 0) (40, 40) (20, 40) outline (10, 20) (40, 20) (20, 30) (40, 30)
Color	black

Figure 3

After creating the instances of objects, the user can see the table for that instance by selecting it with the mouse, that is automatically created by G2 like Figure 4. For each attribute in

the table, simulation details can be written on a specific subtable. The user can get a value for any attribute using the G2 simulator, the inference engine, or from outside files and sensors. After considering where to get values for all the attributes of the instances, the user can write rules on the workspaces attached to object definitions, the instances, and G2 itself like the example given below for HPOT. Each rule has its own table and G2 scans keyrules at rates associated with each rule. For example in the figure 4.1, the scan interval is one second. The scan interval provides a way to regularly invoke a rule to monitor an event. The value of the scan interval tells G2 how often to invoke the rule. The user can follow rule firing path since everytime G2 fires a rule using forward chaining or backward chaining it highlights that rule.

Options	do not forward chain, breadth first backward chain
Notes	OK
User restrictions	none
Names	none
Tracing and breakpoints	default
Data type	quantity
Last recorded value	no value
Validity interval	indefinite
Formula	
Simulation details	discrete-state, with initial value 0, and has specific subtable
Initial value for simulation	0
Data server	G2 simulator
Default update interval	1 second
History keeping spec	keep history

Figure 4

Options	invocable via backward chaining, invocable via forward chaining, may cause data seeking, may cause forward chaining
Notes	OK
User restrictions	none
Names	none
Tracing and breakpoints	default
if the dischargetemp of hpot-tp >= 23 then inform the operator that "This is the upper limit for dischargetemp of hpot" and rotate hpot-tp by 90 degrees	
Scan interval	1 second
Focal classes	none
Focal objects	none
Categories	none
Rule priority	6
Depth first backward chaining precedence	1
Timeout for rule completion	use default

Figure 4.1

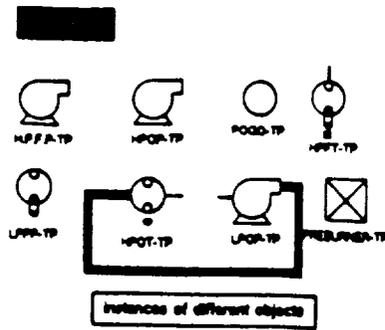


Figure 4.2

Implementation of Intelligent Fault Diagnosis Using G2

As described before to create any instance, an object should be defined. So for the S.S.M.E project in order to describe the project and to create icons which will have its own workspaces, the utility icon, the documentatio-details and KB -details are defined (Figure 5). An icon library is created as shown on the Figure 6, so that it will help the user to extend the system whenever he wants. As shown on the workspace (definitions) on Figure 7, the icons are defined from documentation-detail and KB-detail objects. On the subworkspace of "about-objects" we have successfully implemented eight redline parameters (Figure 2). Then as described before the instances for all these parameters are created by defining the attributes on the subtables attached to them (Figure 3). To give a nice idea about the instances of objects, a new workspace is created and all the instances of redline parameters objects are defined on that workspace (Figure 4.2). Values for those instances of the objects are obtained using G2 simulator, inference engine and data files simulating sensors. As described before, each instance has its own subtable, which has its own specific subtable where we defined our simulation formula as the abstract shown on Figure 8 for HPOT redline parameter.

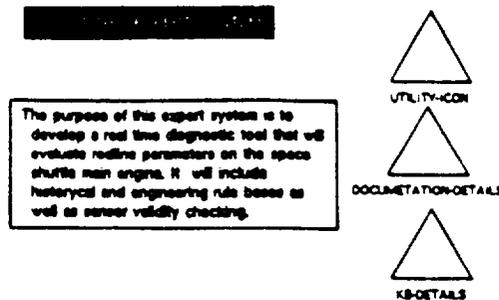


Figure 5

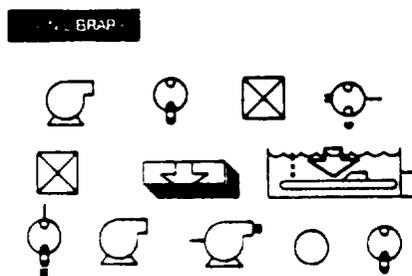


Figure 6

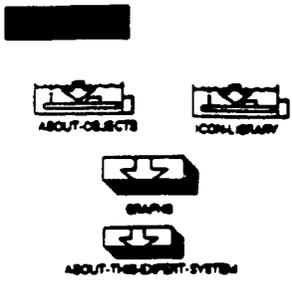


Figure. 7

Information for the display of hpot-tp	
Notes	OK
User restrictions	none
Time increment for update	none
Simulation formula	state variable next value = (8 * (current time / 2)) + 21, with initial value 0
History keeping spec	keep history

Figure 8

After making sure that all the redline parameters are getting values, rules are implemented. The rule groups are divided into three parts: local rules, global rules and chained rules. Local rules are implemented on the workspace attached to object definitions. Global rules relate two or more things at a time and can be fired (Figure 9). Chained rules first try to get a conclusion from global or local rules and then try to find another solution for that conclusion (Figure 10).

The rule base is traversed after each intermediate conclusion is reached until all possible paths have been taken. Sometimes logical variables are used for some redline parameters like low, OK, and high. The low, OK, and high variables have values attached with them. These are the limitations for a particular instance. For example if the attribute dischargetemp of HPOT is greater than high or low or OK then after G2 fires the rule the operator can get the message. We have implemented graphs as shown on Figure 11 to display the result of realtime systems by keeping a history.

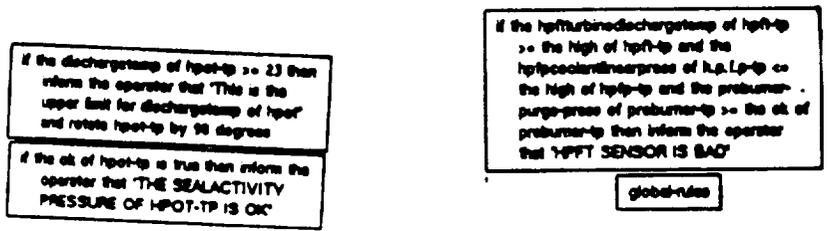


Figure 9

ORIGINAL PAGE IS
OF POOR QUALITY

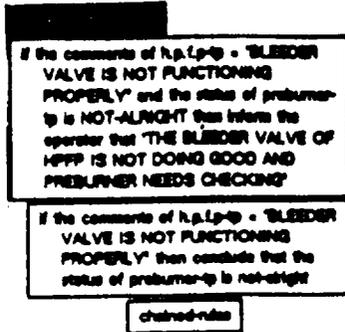


Figure 10

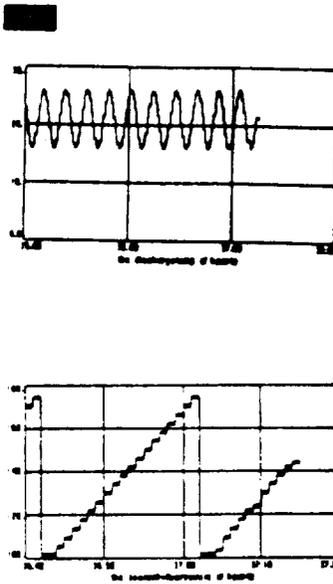


Figure 11

ORIGINAL PAGE IS
OF POOR QUALITY

Through the message board of G2 the diagnostic system communicates to the operator. When the diagnostic system fires the rules different kinds of messages appear on the message board as shown on the Figure below

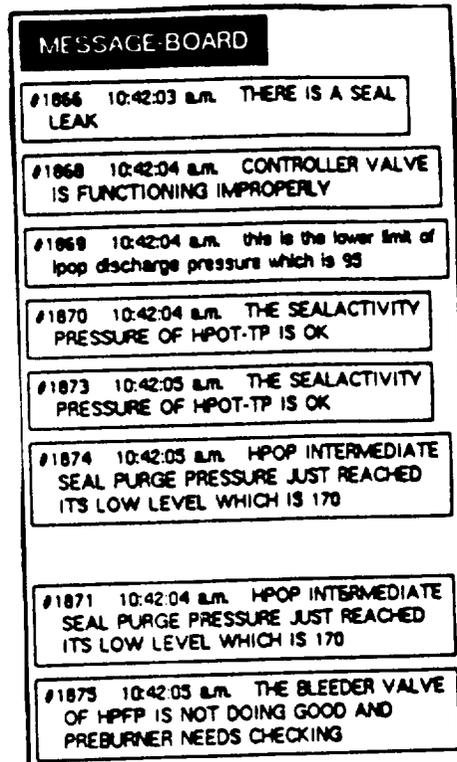


Figure 12

Currently the system contains approximately twenty-five rules which attempt to bind the eight redline parameters. Historical data has been difficult to obtain and consequently that data base has not yet begun to reach its size or pattern matching potential. Also, since the system is currently operating as a stand-alone system and no sensor input is available, simulation and data file supplied values are driving the system. Output is restricted to operator notification and suggested corrective procedures by the system. Scan Intervals are well within one second, yet far from the ultimate goal. These limitations are being slowly reduced and eventually it is hoped they will be removed entirely.

Closing Note

The S. S. M. E. real-time diagnostic system is a good start toward a viable tool. Currently, even using G2, the system is much too slow and the rule and historical data bases are too limited for it to ever be considered for an online control system. However, the system does show potential as an analysis aid for engineers working with the S. S. M. E.

The next phase of the systems development will address these problems by increasing the number of historical database entries as well as continued exploration into methods of speeding up the programs execution. The primary objective of the project is to prove the feasibility of the concept of an extended diagnostic system to S. S. M. E. In this respect the program is succeeding.

REFERENCES

1. Ali, M.,Schrnhorst, D.A."Sensor-based Fault Diagnosis in a Flight Expert System." Proceedings of the Second Conference on Artificial Intelligence Applications, Miami Beach, FL, December 11-13, 1985.
2. Firebaugh, Morris W.; Artificial Intelligence, "A Knowledge Based Approach." Published by PWS-Kent, Boston.
3. Fox, Mark S., Lowenfeld, Simon and Kleinosky, Pamela "Techniques For Sensor-based Diagnosis," Carnegie-Mellon University, The Robotics Institute Technical Report, 1983.
4. "G2 user's manual" Version 1.1, 1988
5. Harmon, Paul and King,David "Expert Systems," Artificial Intelligence in Business; edited by Theron Shrene, 1985.
6. Palmer, Michael T., Abbott,Kathy H., Schutte,Paul C. and Ricks, Wendell R., "Implementation of a Research Prototype On-board Fault Monitoring and Diagnosis System," AIAA Conference of Computers in Aerospace, Massachusetts, 1987.
7. Scharnhorst, Dean A."On the role of Artificial Intelligence In SSS Computer- aided diagnosis ", The University Of Tennessee Space Institute, Knowledge Engineering Laboratory

AN APPLICATION GENERATOR FOR RAPID PROTOTYPING OF ADA REAL-TIME CONTROL SOFTWARE

Jim Johnson

Haik Biglari

Boeing Aerospace & Electronics, 499 Boeing Blvd., Huntsville, AL

Larry Lehman

Integrated Systems Inc, 2500 Mission College Blvd., Santa Clara, CA

Abstract – The need to increase engineering productivity and decrease software life cycle costs in real-time system development establishes a motivation for a method of rapid prototyping. The design by iterative rapid prototyping technique is described. A tool which facilitates such a design methodology for the generation of embedded control software is described.

Introduction

The software crisis

Due to the increasing complexity and size of software projects there is currently a software crisis. Software is on the critical path of project development time lines. It is usually late and over budget and often doesn't perform the necessary functions. One of the primary reasons for the crisis is that software development is a labor intensive and error prone task. In an effort to overcome these problems many automation tools have come into being to assist the programmer. In the area of real-time control software, these tools must assist hardware and software engineers as both are involved in the crisis.

The need for prototyping

Designers are faced with a dilemma. On one hand they need a plan prior to beginning design work and on the other hand they need some initial design analysis for the establishment of a reasonable and adequate plan. The two design philosophies in conflict are the "plan it" approach and the "do it" approach. Both approaches have considerable merit.

In the plan it approach the need for adequate preparation and specification is emphasized. Great pains are taken to ensure a fully specified and documented plan prior to the onset of preliminary design. The plan it approach attempts to avoid inadequate planning and the resulting failure which can be very expensive. The do it approach is to do a prototype design and learn from the mistakes of that initial design prior to the actual system design. The do it approach provides insight into design feasibility. It eliminates the costly and wasteful time spent designing systems on paper without the foggiest notion of whether or not a design will meet requirements. It is desired to combine the strong points of both approaches while avoiding the pitfalls of each.

On a large long term software project, changes in objectives (requirements) are inevitable, so as Brooks states, prepare for them.[1] During preliminary design, changes in development strategy and technique are typically frequent. According to Brooks, one should always have a prototype or pilot model from which to learn. "The throw-one-away concept is itself just an acceptance of the fact that as one learns, he changes the design." Plan to throw the first one away, or at least to modify it significantly.

Rapid prototyping in iterative design

On a large project it is not possible to prototype the entire project due to time and economic constraints. However it is still desirable to be able to use the prototype approach. The prototype method best for large projects is the iterative prototype method. Iterative prototyping implies many prototypes. This requires a tool capable of generating rapid prototypes for evaluation of incremental changes. The prototyping tool should be based on natural language of the intended user. In the control engineering domain this language would be based on block diagrams. A tool which provides such capabilities is the Application Generator (AG). The AG solution to the design dilemma fulfills the needs of adequate planning and design prototypes in a design methodology henceforth called "design by iterative rapid prototyping."

Copyright © The Boeing Company, 1989, All Rights Reserved.

PUBLISHED WITH PERMISSION

The Application Generator

The Application Generator is an alternative to conventional application software development. It reduces the software life cycle cost associated with application software generation activity. The principle behind the AG is that process control engineers must perceive control concepts and strategies that require control of several processes within a particular plant simultaneously. To achieve this (see Figure 1,) the process

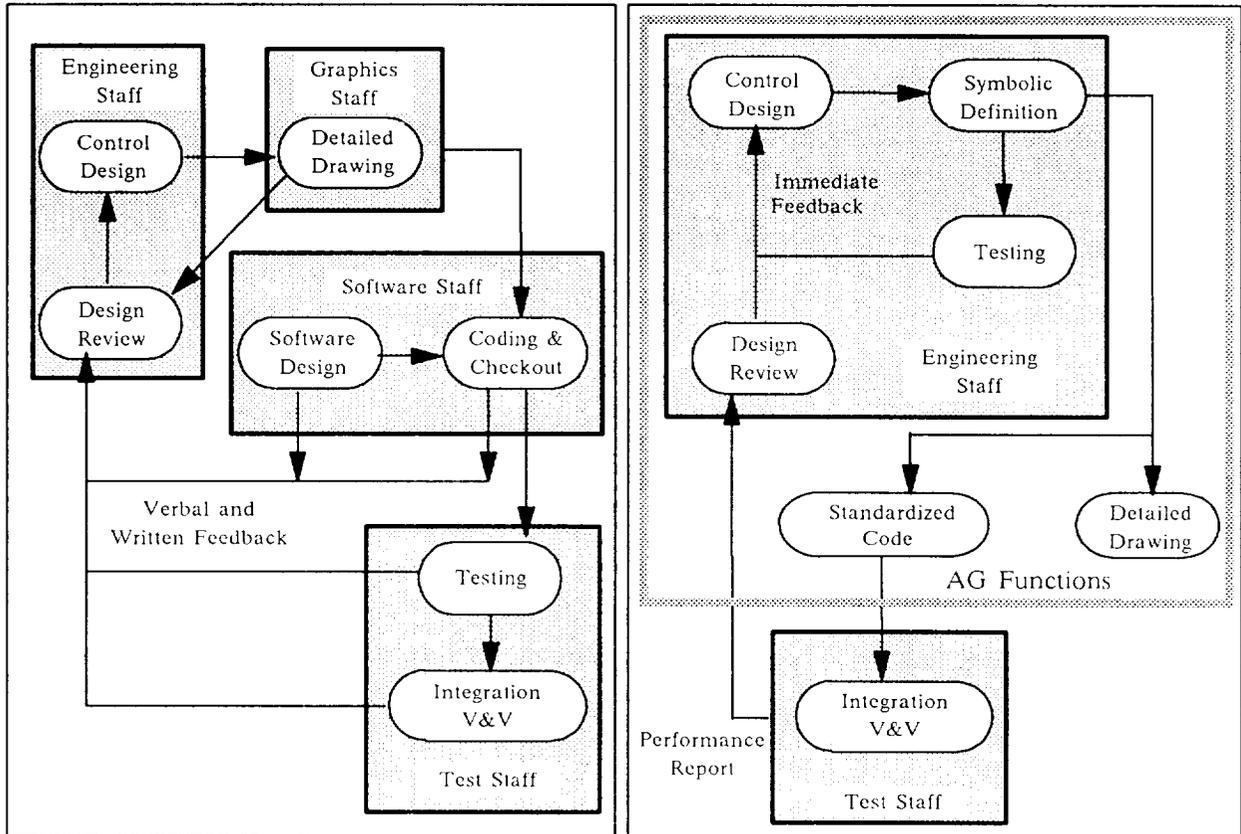


Figure 1 – Conventional Control Algorithm Development And Implementation

Figure 2 – Control Algorithm Development And Implementation Using the AG

control engineer draws spatial and temporal diagrams representing the processes to be controlled and derives the necessary control algorithm for each particular mode of operation. Conventionally, software engineers take this information and convert it to sequential codes suitable for a digital computer. Process control engineers evaluate the performance, take corrective action, and improve the control strategies. The result is submitted to software engineers and the cycle continues until a desired system response is obtained. The AG simplifies this procedure significantly by allowing the process control engineers to enter their control specifications directly into a computer and obtain analysis and simulation responses that they may use to modify their algorithms (see Figure 2).[2]

Ada Software Prototyping

Motivation

Development of prototype real-time software can be nearly as difficult and expensive as development of production software. Costs, debug time and tedious algorithm coding of embedded control systems are software development problems which can be addressed by an automated prototyping tool. Some of the features which should be included in the tool to make it an effective prototyping tools are a graphical interface, a means for providing interactive animation and automated generation of real time control software. The graphical interface should be a block diagram editor with icons familiar to control engineers. The modification, interconnection and manipulation of these blocks should be straightforward in a user friendly environment.

Reducing Software Life Cycle Costs

Conventional software development is modeled using the waterfall type phased development. The conventional software life cycle is shown in Figure 3. With the Application Generator, the software life cycle is

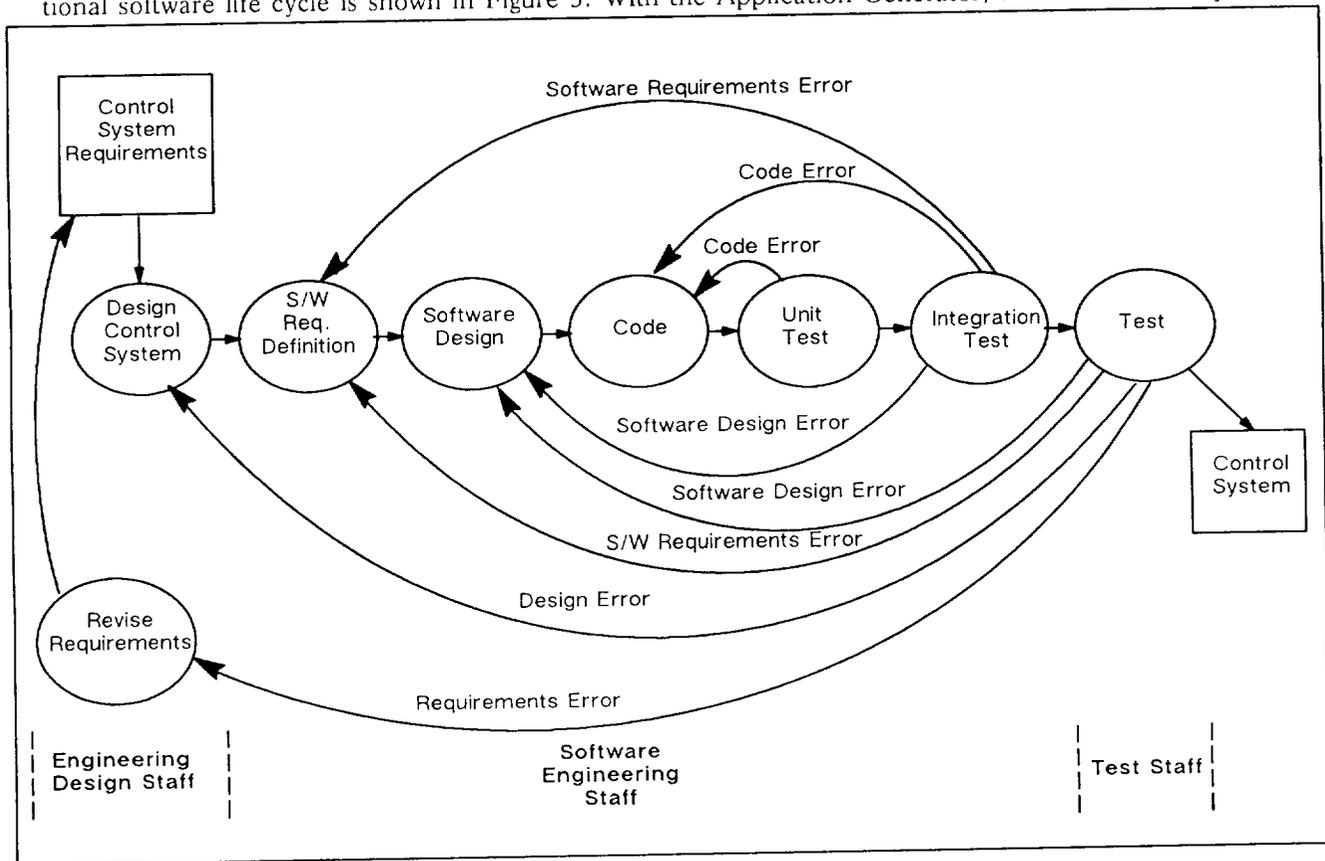


Figure 3 - Conventional Control System Software Development

modified as shown in Figure 4. The AG life cycle is modified by introduction of iterative rapid prototyping and an integrated toolset. Figure 4 displays how requirement and design errors can be discovered earlier in the life cycle and handled inside the AG process.

The relative cost to make a change increases throughout the lifecycle.[3] The overall life cycle is unchanged, but now there is tighter feedback, faster iteration. The design engineering includes concept, design, off-line simulation and prototyping. The software engineering includes prototyping, implementation and maintenance. The prototyping phase is included in both the engineering design and software engineering activities.

How the AG is used in the proof of concept, design, off-line simulation, prototyping, implementation and maintenance of control law development is detailed below:

a. Concept

Proof of concept can be demonstrated in the modeling phase of design using the AG. Requirements specifications can be evaluated for feasibility as the models are simulated. The model serves as a baseline for development of control strategies.

b. Design

Interactive graphical design using standard control engineering block diagrams facilitates an efficient and effective means to expedite design specifications and implementations.

c. Simulation

Simulation of the modeled plant and control provides immediate feedback to the designer. It provides an indication of whether design goals can be achieved.

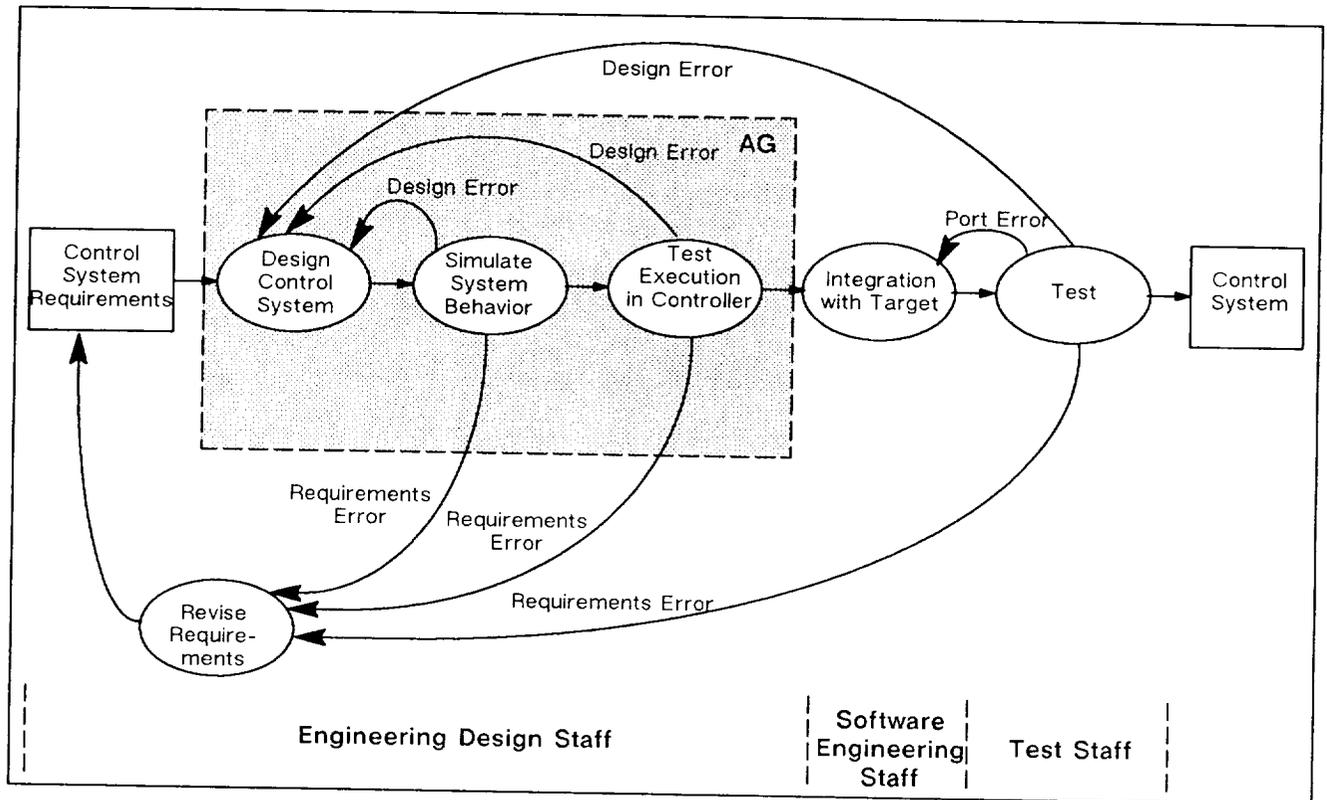


Figure 4 AG-Based Control System Software Development

d. Prototype

Rapid generation of prototype software enables the control engineer to verify design goals. It supplies the ability to test various control strategies, and it allows evaluation of alternatives and options to determine the best design.

e. Implementation

Because the AG controller emulates the target controller, implementation is simply a matter of determining porting and final integration issues. This can be performed once and used by all subsystems developers.

f. Maintenance

Maintenance is accomplished at the control block diagram level, which means that control diagram documents are automatically available and that they are maintained consistent with the control software. Also the control engineer has high visibility into the design maintenance.

Eliminating Software Coding Errors

The later in the life cycle that an error is detected, the more expensive it is to find and repair. It must be determined if the bug is in the software or in the control design. "Due to the introduction of bugs, program maintenance requires far more system test per statement written than any other programming." [1] Some methods to mitigate bug introduction in the implementation of designs are to find ways to use fewer people, fewer interfaces, and greater uniformity.

Automating Algorithm Coding

It is well understood that high level languages and self documenting techniques can be used to reduce errors induced by changes.[1] The control diagram level input is a self documenting high level language. There is a risk of error introduction in the translation of control diagrams to software code. This risk is minimized by automating the coding of algorithms directly from the block diagrams.

Variation among programmers in productivity [3] is a big problem in trying to estimate costs, but variation among subcontractors in coding style and software design philosophy can be a nightmare for software integration. On large programming projects, consistency of code from subcontractors is a major concern for a prime contractor responsible for integration. Code generated by the same tool, an Application Generator, is uniform across all subcontractors in terms of style, format and software design methodology.

System bugs arise from mismatched assumptions made by various people. [1] The assumptions and requirements are more highly visible at the block diagram level than when embedded in the syntax of a programming language.

All too often a software engineer is frustrated in attempts to debug an apparent software error when the actual problem is a design error. But in the numerous steps of translation from design drawings, to final coding, the error source is obscured.

Many design tools which speed up design work already exist. New features which are needed include a prototyping capability, a friendly user interface, high performance, deterministic behavior, and an environment to support increased levels of fidelity between the simulation model and the target code.

Project Description – Space Station Freedom

Background

Boeing's Space Station Freedom role is to develop the pressurized living, working, and storage areas under contract to NASA's Marshall Space Flight Center (MSFC). Onboard systems contained in these elements include thermal control, environmental control and life support, internal audio and video, and experiment facilities. The onboard Data Management System (DMS), which includes processors, networks, workstations, operating systems, network operating systems, data base management system, and a user interface language will be supplied to Boeing by NASA. However, Boeing must develop and test application software for the onboard systems before the DMS components are available. An interim development system is required to rapidly design and control the onboard systems. The Application Generator (AG) concept was selected for the development system because of its productivity potential and usability by control systems engineers. [2]

Requirements For Application Generator Tools

Large systems such as Space Station Freedom have numerous processes that require a large number of controllers to control and monitor the entire system. Such control is best achieved if a local area network is utilized that is capable of having many controller nodes. It is also necessary to have programmability for the controllers so that future needs can be accommodated with minimum cost and effort.

To further reduce systems operation costs, commonality must be maximized. One method of achieving commonality is to establish a set of standard control system icons. A data table corresponding to the selection and organization of these icons in a control design can be used by a common code generator. Commonality can also be maximized by having a common run time kernel. At the workstation level the process and the control algorithms are defined as a data structure which is used to generate high level source code. This source code is cross-compiled, linked with the run time kernel, and down-loaded to the controller's RAM. This provides an environment that supports all phases of development and testing (see Figure 5).

Four categories of requirements are detailed in the following paragraphs. They are software requirements, user interface requirements, integrated toolset requirements, and hardware requirements.

Software Requirements

The three main software requirements are design simulation, rapid prototyping capability, and generation of reusable software modules.

a. Design Simulation

Simulation of the system with Ada executing on the AG controller provides an easy means of testing software design.

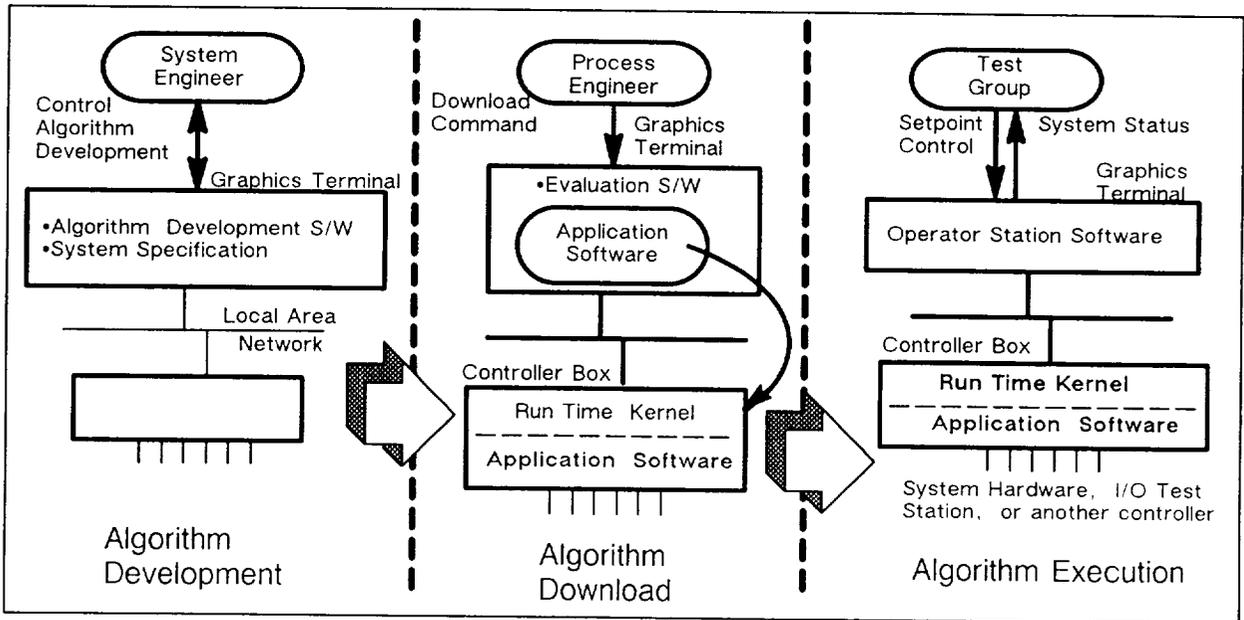


Figure 5 - AG Software Generation Phases

b. Rapid prototyping

The process of rapid prototyping makes requirement specification, design, simulation, validation, and hardware/software trade-off analysis an integral part of the design and development process. [4] The standard waterfall type development alternative treats these tasks as afterthoughts, as if they were simply by-products of the coding phase.

c. Reusable Software

AG supplied control diagram blocks are a form of reusable software. Blocks can be grouped together into user designed super blocks. These super blocks can also be cataloged making them available for reuse.

User Interface Requirements

A rapid prototyping environment requires a user friendly graphical interface and animation capabilities as described below:

a. Graphical Interface

The AG workstation software uses multiple windows, pull-down menus, and mouse input for operator interaction. The major components of the workstation software are the interactive animation and graphical programming environment utilities.

b. Graphical Programming Environment

Using standard control engineering diagram icons available in the software, control engineers are able to input control law algorithms efficiently.

c. Interactive Animation

To test the functionality of the real-time control software in a typical set of environmental and user inputs, it is desirable to have an interactive animation capability. (see Figure 6). Note that the interactive animation ties in directly to the control system on the AG-100 controller. Interactive animation is used in simulation, software prototyping and hardware prototyping. The interactive animation schematic contains icons which represent actual hardware and its associated sensors and actuators.

Integrated Toolset

It is advantageous to have an integrated toolset which resides on one workstation. Some of the benefits are that there is only one set of workstation environment software to learn, data transfers between various tools are local to a workstation and all applicable data sets are resident on the local workstation. The types of tools which comprise the toolset are listed below.

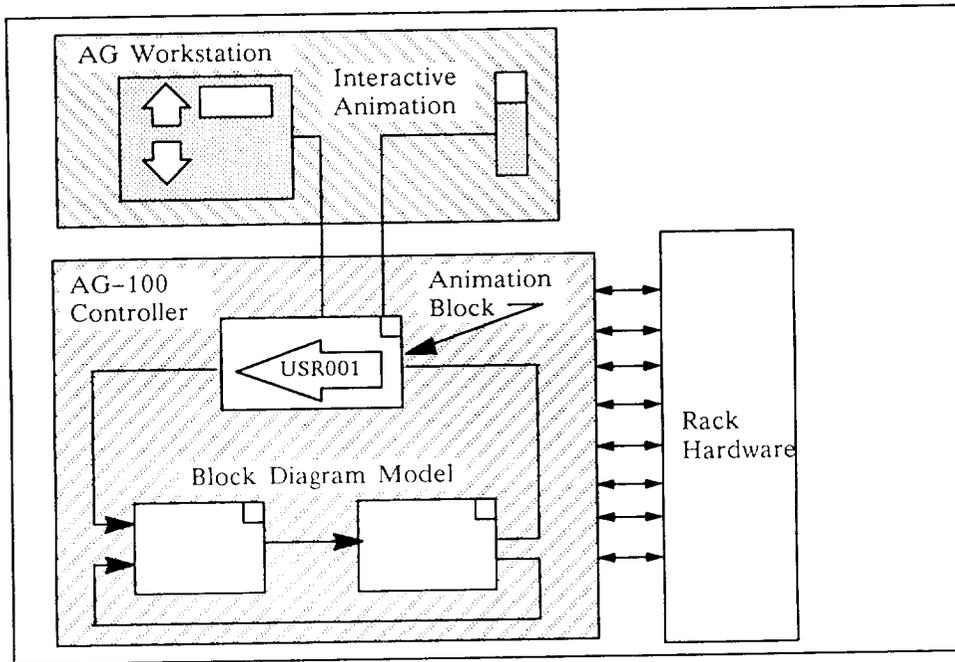


Figure 6 - Rack Hardware Test

- a. Design and Simulation Tools
- b. Software Support Tools
- c. Documentation Tools
- d. Project Management Tools
- e. Data Dictionary
- f. Interface To Other Engineering Databases

Hardware Requirements

The AG hardware components are the workstation, the controller, the optically isolated I/O modules, and the local area networks.

- a. Workstation

The AG workstation is a VAXstation 3100 graphics workstation with 19" color monitor using a DEC-window user interface. It has 8 Mbytes of memory, 208 Mbytes of disk storage and thinwire/thickwire Ethernet network capabilities.
- b. Programmable Controller

An embedded data processor (EDP) emulation using 80386 and 80186 message processor is provided by the AG controller. The programmable controllers operate in real-time and have multi-tasking capability, allowing them to run several different rate control algorithms simultaneously.
- c. Optically Isolated I/O Module Rack

A low bandwidth noise suppression, noise tolerant I/O capability is necessary for the Space Station Freedom environment.
- d. Local Area Networks

There are two communications networks, the Ethernet for Host/Workstation/Controller communications and the Token Bus for Controller/Supervisor communications. The Ethernet has a 10 megabit/second data rate to facilitate a common communication medium used to connect workstations, controllers, and user-provided equipment.

Using The AG On Space Station Freedom

The AG is used to aid in the integration of low level code modules into rack level hardware and system level software. AG generated code has not yet been approved as actual flight code, so current use of AG code is restricted to prototyping of hardware and software.

Real Time Embedded Control Processor

The Application Generator includes hardware and software to support the embedded controller. The tools which support the embedded controller are integrated with the tools used to develop control algorithms allowing for rapid iterative prototyping.

For real time systems there is the special need for Ada tasking, and schedulers. The task of generating real time embedded control software is complicated by the extra implementation steps of cross compilation and downloading as well as the more primitive debugging tools common to embedded controllers.

Flight Hardware Transition

It is proposed that the AG controller serve as a prototype for the flight controller. As components of the flight hardware such as flight qualified I/O modules become available they will be inserted into the AG controller. This will help to provide a smooth transition from the AG controller to the actual flight controller.

The most challenging tasks expected during this transition are converting from AG runtime kernel to the EDP operating system, the I/O module fidelity, and the speed of the I/O. Errors discovered here are the porting errors (See Figure 4). These are the only class of errors in an AG based control system software development which do not result in changes at the control diagram level. Note that correction of the porting errors does not require a redesign of the control system or the re-coding and unit testing of control algorithms.

System Design

The architecture of software implementation is common across all subsystems. Control engineers use simulation models and software engineers use configuration, implementation and customizing tools. Configuration management of models is implemented in a code management system similar to the DEC code management system, CMS.

The hierarchy of control diagram blocks into super blocks greatly facilitates the design of large systems on the AG. Merging of control algorithms is more readily accomplished at the block diagram level than at the code level.

Integration In a Large Scale Development

AG generated software must communicate with higher level software used to manage an entire Space Station Freedom element, such as the Habitation element or Laboratory element(see Figure 7). The

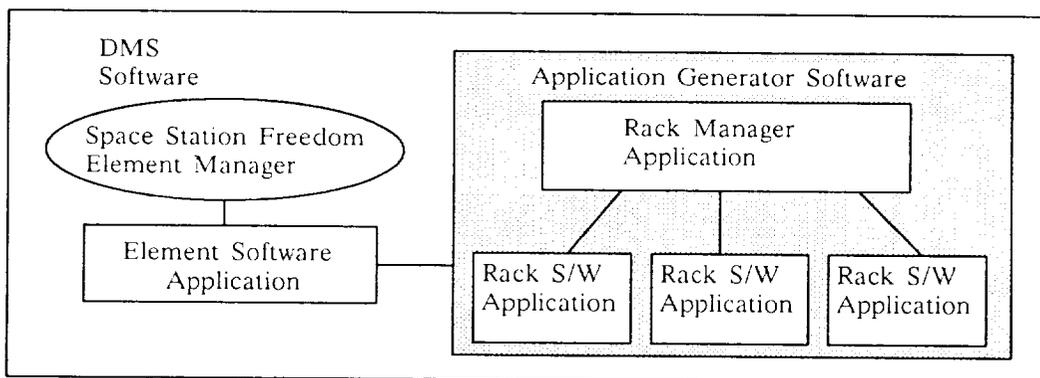


Figure 7 - AG Interface to Space Station Freedom Element

high level interfaces are modeled on the AG in the form of messages to and from the low level AG control algorithms. The AG is limited to development of software in individual racks where each rack typically has one controller. Software at a higher level to control an entire Space Station Freedom element, is not developed on the AG.

Training

A user friendly workstation environment helps to mitigate the complexity of a sophisticated tool such as the AG. Formal training accelerates the learning process. A major objective of training is to encourage the control engineer to become familiar with all phases of design and implementation using the AG. This is consistent with a primary objective of the AG, which is to keep the control engineer in the loop throughout all phases of development.

Summary

Interactive rapid prototyping is a key element in the software life cycle. It allows completely new control software to be developed rapidly so that alternative control strategies can be evaluated in a cost effective manner. It also provides the ability to address system integration and performance issues early in the development cycle.

The Application Generator provides a flexible environment for development of robust process control systems. It is based on automatic code generation directly from control block diagrams. A highly intuitive and interactive environment is used to specify mathematical models of the hardware and implement necessary control strategies.

Acknowledgments

We thank the numerous people who encouraged and supported us and specifically we extend our appreciation to Bill Walker and Roger Williams for their valuable comments and suggestions.

References

- [1] Brooks, F.P., Jr: *The Mythical Man-Month*, Addison-Wesley, 1982
- [2] Boeing, *Application Generator Statement of Work*, D683-10094-1
- [3] Fairley, R.: *Software Engineering Concepts*, McGraw-Hill, 1985
- [4] Tanik, M.M. and Yeh, R.T. "Rapid Prototyping in Software Development" *IEEE Trans Comp* vol c. 22 no 5 , May 89

AN ENGINEERING APPROACH TO AUTOMATIC PROGRAMMING

by

STUART H. RUBIN *

(3T4U5FH@CMUVM)

Department of Computer Science

Central Michigan University, Mt. Pleasant, MI 48859

ABSTRACT

An exploratory study of the automatic generation and optimization of symbolic programs using DECOM - a prototypical requirement specification model implemented in pure LISP was undertaken. It was concluded, on the basis of this study, that symbolic processing languages such as LISP can support a style of programming based upon formal transformation and dependent upon the expression of constraints in an object-oriented environment. Such languages can represent all aspects of the software generation process (including heuristic algorithms for effecting parallel search) as dynamic processes since data and program are represented in a uniform format.

Keywords - Constraint-based programming, object-oriented programming, software automation, transformation

1. INTRODUCTION

Software is currently the major cost in information processing systems. It is estimated that information processing systems will account for 13% of the U.S. GNP in 1990 [9]. Higher level languages are necessary in order to obtain significant improvements in the software automation and support process. They also provide substantial decreases in the time and cost of software development, as well as provide major reductions in the cost and time for maintenance and modification of software. Moreover, higher level languages make the management of the software development activity easier and represent a step in the direction of automatic programming.

* Funding for this project, carried out at NOSC, was provided by the Office of Naval Technology (ONT) Summer Postdoctoral Fellowship Program, Projects Office, ASEE, 11 Dupont Circle, Suite 200, Washington, DC 20036 U.S.A.

2. WHY PURE LISP?

There are three good reasons for choosing a functional language like LISP: firstly, functional programs are invariably much shorter, more abstract, and easier to understand than their procedural language counterparts; secondly, pure functional programs are amenable to formal analysis and manipulation, and thirdly they are naturally amenable to implementation on a parallel machine [4]. In addition, functional programs describe the transformation of input values to output values - making it possible to establish properties about them and to transform them into more efficient forms through the apparatus of conventional mathematics.

LISP is the oldest and most widely used symbolic language [13]. In it, a list can contain different types of objects. LISP is more flexible than statically typed languages like PASCAL and C because it supports dynamic typing. In LISP, function calls, control structures, and built-in operators have the same syntax - facilitating extensibility. Moreover, LISP macro expansion is performed by user-defined functions, thus letting an arbitrary computation compute the result of the expansion [13]. Hence, it follows that LISP is an excellent language for implementing a transformational synthesizer.

Pure LISP is a universal orthogonal subset of LISP composed of basic functions for constructing pairs, lists, and numbers; namely CAR, CDR, CONS, EQ, and ATOM. It also incorporates the control structures using COND, recursion, and functional composition (including some means for function definition). In fact, the pure dialect requires list structures containing only atoms and sublists - without numbers or property lists.

Pure LISP is declarative in nature. Thus, it helps to avoid unnecessary sequentiality in a specification, which in turn facilitates introduction of parallelism [2]. This is because the order of evaluating the arguments of a multiple-variable lambda-expression is not defined. Hence, such lambda-expressions are a source of parallelism. Moreover, the referential transparency of the language (i.e., variables are bound to expressions) eliminates the need to access complicated data flow analyses and the rules of Church's lambda-calculus can be used as the basis for transformations that manipulate it [2].

Graham has implemented a database system which stores information in the form of LISP lists and responds to queries about the information it has stored [5]. Rubin has characterized *learning* as a process for the compression of information in order to yield knowledge (i.e., theory formation and revision) [12]. Hence, the approach to higher level languages advocated herein extends to higher level data and knowledge bases too. It follows that expert systems (and hence their explanation facilities, the knowledge acquisition bottleneck, and control problems) stand to be enormously and favorably impacted by this technology, since their operation depends upon effectively interfacing with one or more knowledge bases.

3. REUSABLE PROGRAM SPECIFICATIONS

The DOD has invested \$300 million in the STARS (Software Technology for Adaptable, Reliable Systems) project to investigate software reuse [10]. NASA recognizes that the United States needs a flight-research facility dedicated to rapid avionics prototyping. The Agency is now developing the Ames-Dryden facility to meet that need through reusable software [3].

A higher form of software reuse is needed to overcome the limitations of code reuse. Software reuse becomes more feasible if program specifications are reused instead of

program code. Hence, program specifications should be formally defined in order that they may undergo automatic and formal correctness-preserving transformations. Note that program specifications conveniently serve the purpose of verification and testing. Finally, Sellis et. al. note that the scale of a transformational system is an important design consideration since future expert database systems will contain knowledge bases of significant size which makes main memory insufficient and the use of a database system a necessity [12].

4. AN ALGEBRAIC APPROACH TO FUNCTIONAL SYNTHESIS

The algebraic transformation method is based upon a collection of theorems which state generic equivalences, i.e., semantic equalities, between classes of functions. Then in a functional program, expressions may be rewritten by more efficient, equivalent expressions which are given by one of these theorems. In this way, the process of transformation becomes that of the identification and application of instances of theorems, and the algebraic approach is therefore particularly conducive to mechanization [4]. Optimization is thus a consequence of some underlying analysis which establishes theorems equating an 'original', user-defined function with a more efficient version. As a general definition of an algebraic approach to specification transformation, suppose that the user has defined a pair of abstract data types \mathbb{A} , \mathbb{B} , and the corresponding concrete pair \mathbb{A}' , \mathbb{B}' which provide realizations of \mathbb{A} , \mathbb{B} respectively. Then, given any function $f: \mathbb{A} \rightarrow \mathbb{B}$, it is desired to synthesize a corresponding function, say $f': \mathbb{A}' \rightarrow \mathbb{B}'$, which performs operations on objects of type \mathbb{A}' which are isomorphic to the operations performed by f on corresponding objects of type \mathbb{A} . The function f' is then the concrete, implementation version of f that was sought.

Many functions $f': \mathbb{A}' \rightarrow \mathbb{B}'$ corresponding to $f: \mathbb{A} \rightarrow \mathbb{B}$ supplied by the programmer may be synthesized by process of algebraic transformation [4]. Each abstract or complex transformation within a system results in a new lower level subsystem.

Functional synthesis may be applied not only to generating programs but also to constructing other complex objects such as relational database implementations of first-order logic queries, VLSI circuit designs, and detailed plans for robotic vehicles to achieve a set of military reconnaissance goals [8]. Hence, it follows that the pursuit of transformative synthesizers has the potential for very broad impact.

5. RESULTS WITH THE DECOM SYSTEM

The DECOM or program decomposition system is intended to minimize the occurrence of software bugs through the use of a top-down structured approach to software reuse. The current version uses a subset of Common LISP as its implementation language. Note that DECOM, version 1, while only a prototype, serves to illustrate the potential of the concept of software reuse through knowledge-based design in an object-oriented environment. It also serves as a model for human learning through the use of function(al) composition.

To begin with, consider the programmer working in an object-oriented environment (i.e., without loss of generality). Let

$$((\text{FUNC}) (((\text{IN}_1) (\text{OUT}_1)) ((\text{IN}_2) (\text{OUT}_2)) \dots ((\text{IN}_n) (\text{OUT}_n)))) \quad (1)$$

define an arbitrary LISP function which satisfies all of the specified distinct I/O constraints (i.e., at least one pair required). DECOM will take such a specification and through the use of knowledge-based heuristic search and user assistance define a function(al), FUNC, such that it satisfies all specifications.

FUNC may be viewed as a procedural knowledge source representing the compressed declarative information contained in all of its constraining I/O pairs. Moreover, the I/O pairs may be viewed as production rules. It then follows that DECOM functions as a fully general rule-inducing system having demonstrable/provable convergence properties. It is worth noting that if FUNC is defined to be a functional, then a knowledge base segment of optimizing transforms may be inductively generated (and tested). Naturally, these functionals will be maintained as fixed points with respect to the contents of the appropriate optimizing knowledge base segment. Different knowledge base segments are represented by different sublists - that's part of the overall beauty of the scheme.

First, DECOM searches the existing knowledge base segment, shown by (2) below, for an exact match of the I/O specification pairs (where $m <$ the number of concurrent processors). If the knowledge base segment is empty, then proceed to the next step.

```

((FUNC1) ((IN1,1) (OUT1,1)) ((IN1,2) (OUT1,2)) ... ((IN1,n1) (OUT1,n1))) (2)
((FUNC2) ((IN2,1) (OUT2,1)) ((IN2,2) (OUT2,2)) ... ((IN2,n2) (OUT2,n2)))
((FUNC3) ((IN3,1) (OUT3,1)) ((IN3,2) (OUT3,2)) ... ((IN3,n3) (OUT3,n3)))
. . . . .
. . . . .
. . . . .
((FUNCm) ((INm,1) (OUTm,1)) ((INm,2) (OUTm,2)) ... ((INm,nm) (OUTm,nm)))

```

If an exact match of the I/O specifications is found, then $FUNC_i$ is returned as the desired LISP function. If however the knowledge base segment is empty, then the user is asked to specify a reduction(s) (if necessary) and proceed with the component derivation as described above - storing their interrelations in the knowledge base in the form of a "macro"-function(al).

If an exact match cannot be found, then a heuristic means-ends analysis attempts to locate the closest match. The heuristic (a dynamic object) is a search function(s) saved in a knowledge base segment. Note that more than one 'closest' match may be explored in parallel. Alternatively, if no I/O specification pairs in the knowledge base satisfy the defined matching metric, then the case is handled as though the knowledge base were empty.

Now, for each closest match found above, a function $FUNC_i$ is searched for such that it distinctly maps each of the given inputs to a corresponding input, $IN_{k,n}$, where the single function $FUNC_k$ is known by the knowledge base. That is, the knowledge base attempts to map the specification by process of forward composition (i.e., state-space heuristic search).

Much like a genetic approach [6,7], the current approach entails the use of combinatoric search. However, it surpasses the capabilities of a genetic approach in that the powerful technique of means-ends-analysis is fully utilized. Besides, it should be noted, lest the reader struggle with the question as to which approach to take, that genetic algorithms can

be embedded within DECOM. Again, each given specification pair must be distinctly mapped onto the same FUNC_k if the composition is to be successful (extraneous specification pairs are ignored as in the proof of the program form of the *parametization* or *s-m-n* theorem which underpins most of computability theory [1]). Furthermore, this mapping must be effected by the same FUNC_i (which itself may be a specified composition). Then, the desired function (3) is given by the forward composition $f_k \circ f_i$

$$(\text{FUNC}_k (\text{FUNC}_i (\text{IN}))) \quad (3)$$

The use of composition may be extended to an arbitrary level, $f_m \circ f_{m-1} \circ \dots \circ f_2 \circ f_1$, subject to the number of available concurrent processors. The application of optimization rules can prune the search tree and/or compress the result.

The mapping of outputs is analogous to the case for inputs - except that here, the desired function (4) is given by the backward or inverse composition $f_k \circ f_i^{-1}$ (i.e., goal-driven heuristic search)

$$(\text{FUNC}_k (\text{FUNC}_i^{-1} (\text{OUT}))) \quad (4)$$

where FUNC_i^{-1} maps the outputs as described above for the case of the inputs. Note that

$$(\text{FUNC}_i^{-1} (\text{OUT})) = (\text{FUNC}_k (\text{FUNC}_i (\text{IN}))) \quad (5)$$

Moreover, there is no reason that an inverse composition, $f_m^{-1} \circ f_{m-1}^{-1} \circ \dots \circ f_2^{-1} \circ f_1^{-1}$, cannot be combined with a forward composition for greater efficiency (i.e., bidirectional heuristic search).

New functions (i.e., functions defined by composition as per above) are saved in the knowledge base if and only if they have accepted optimization or have been manually specified due to failure, for whatever reason, to be the result of composition. (Frequently referenced functions should be copied, in expanded form, into a cache.) This is not unlike case-based reasoning, since the larger the knowledge base, the more likely the matching metric is to succeed. Also, it is clear that the matching metric should be a dynamic object(s), saved in the knowledge base, although this aspect has not yet been explored.

Note that erroneous functions may be pulled from the knowledge base at any time - independent of any other functions. They can subsequently be re-synthesized from the (updated) specifications. Hence, the DECOM system, like a neural net or even a DNA program, exhibits a capability for self-repair.

IN and OUT can specify LISP functions since again LISP makes no syntactic distinction between program and data. It follows that FUNC can serve as a functional - mapping LISP functions, meta-functions, or even entire knowledge base segments, and so on. This is vitally important to the efficient working of the described transformational synthesizer (even on a connection machine) because functions carried as specifications define optimizing rewrite rules. Hence, it is generally more efficient to maintain them in a separate knowledge base segment consisting of fixed-point functionals. Note that function(al)s can be recursively defined using a push-down stack of pending tasks.

6. AN INTRODUCTION TO OPTIMIZING TRANSFORMS

One of the key results pertaining to optimizing transforms is that their effects often enable subsequent optimizations. To see this, first consider the following abstract function sequence (6) and the three associated Type 0 rewrite rules:

FUNC: UVWXYZUVW (6)
R1: VW --> X
R2: XX --> Z
R3: Z?Z --> Z

A derivation sequence (7) is given by:

UVWXYZUVW .R1-> UXXYZUX .R2-> UYZUX .R3-> UZUX (7)

Note that optimizing rewrite rules are saved as fixed points with respect to the segment in which they reside. That is, the i^{th} segment is such that for all contained rules, there does not exist a contracting rule, R_j , whose antecedent matches any of the patterns found in the i^{th} segment - itself excluded. Note that the use of the term "fixed point" here applies only so far as a one-step derivation is concerned. It does not contradict the undecidability of the minimalization problem [1].

The question arises as to how many different ways the optimizing rules can be applied and with what result. The above example provided no branching in the derivation tree. However, this is obviously a special case. In general, given Type 0 (i.e., universal program or context sensitive contracting) rewrite rules, a derivation can be arbitrarily long and include multiple applications of the same rewrite rules. What this means in a practical context is that abstract program specifications can, in general, derive an arbitrary number of concrete programs. Providing additional specifications may limit this number if the functions defining sequence is altered with respect to the applicable rewrite rules as a result.

Hence, it becomes necessary, in general, to provide an agenda mechanism to order the potential application of the rewrite rules. This agenda is represented in the form of a meta-rulebase segment. An initial sample meta-rulebase segment (8) for the given rulebase follows:

MR1: U --> R1 (8)
MR2: UX --> R2
MR3: UZ --> R3

Meta-rules are treated the same as ordinary rules and thus are saved as fixed points. Hence, (8) is saved as follows:

MR1: U --> R1 (9)
MR2: APPLY (R1) || X --> R2
MR3: APPLY (R1) || Z --> R3

The principal advantage of the fixed point format is that it is more readily amenable to parametrization (such as substituting R_k for R_1 above), or in the general case, transformation. This advantage applies to rule, meta-rule, ..., meta^n -rulebase segments alike. Note that the Type 0 characterization of the rewrite rules implies the absence of hierarchy in the meta^n -rulebase, $n = 0, 1, \dots$. Hence, the distinction between rules and meta-rules is merely an illusion which is well-adapted to the purpose of illustration.

Finally, the concern relates to the acquisition of all manner of rules. A recursive model of EBL [11], although not yet implemented, is proposed which induces (meta) rules from their specifications. This idea is consistent with the methodology presented in this paper and will be formally analyzed in forthcoming works. Specifications are optimization constraints which are discovered in retrospect (such as through the use of backtracking). Good (meta) rules tend to reinforce the discovery mechanism; bad (meta) rules achieve the opposite effect. Again, backtracking is but one discovery mechanism - another is heuristic search. The key point, at least at this level of discussion, is that all effective process are given a uniform representation within the system and hence are equally subject to inductive extension.

7. A SIMPLE EXAMPLE

The above exposition will be concretized here by way of a relatively trivial example serving to illustrate the main points made above. To begin with, assume the existence of the following knowledge base segment:

$$\begin{aligned} &(((\text{CAR} (\text{LAMBDA} (X))) (\text{NIL} \text{NIL}) ('(A) A) ('(A B) A)) && (10) \\ &((\text{CDR} (\text{LAMBDA} (X))) (\text{NIL} \text{NIL}) ('(A) \text{NIL}) ('(A B) (B)))) \end{aligned}$$

Notice that the constraints are ordered - in this case in order of nondecreasing sublist length - in order to facilitate the search and match process. Also, while the number of constraint pairs has been set at three for each function, it is recalled that the only requirement is that at least one constraint pair be defined for each function - with each function allowed arbitrarily more.

Next, a pair of constraints are specified and the sought after function is initialized to the NIL value:

$$(\text{NIL} (\text{NIL} \text{NIL}) ('(A B) B)) \quad (11)$$

Now, the knowledge base segment is heuristically searched in a forward direction (the heuristics may reside in a distinct segment) for a function which distinctly maps each input list in the I/O pairs of the unspecified function to the corresponding input lists of a single function residing in the appropriate knowledge base segment. That is, the image under the operation of the applied function will constitute a suitable preimage under the operation of some known function(al) in the relevant knowledge base segment. In the current instance, the images under the operation of the CDR function are NIL and (B), and the preimages under the operation of composition with the CAR function are NIL and (A) respectively. Hence, the following subgoal is attained:

$$((\text{CDR} (\text{LAMBDA} (X))) (\text{NIL} \text{NIL}) ('(A B) (B))) \quad (12)$$

Next, the above process is iterated where the images under the operation of the CAR function are found to be NIL and B - satisfying all constraints. Hence, the following is the attained goal as desired:

$$((\text{DEFUN} \text{HEADTAIL} (X) (\text{CAR} (\text{CDR} X))) (\text{NIL} \text{NIL}) ('(A B) B)) \quad (13)$$

Note that all LISP errors are interpreted by convention to be the special atom - NIL, or equivalently, the empty list - ().

8. CONCLUSIONS

It follows from experience with the DECOM system that function(al)s can be automatically induced in an extensible coherent environment through the use of a technique for programming by example. The DECOM system also advances the suggestion that AI and distributed computation are interdependent. These fields are unified through the use of the LISP symbolic language - a representational vehicle where the data and the program have the same list structure. Other languages may be employed if translated into a suitable symbolic representation. That is, all programming constructs may be placed in bijective correspondence with pure LISP constructs.

The use of constraint-based transformation in an object-oriented programming environment promises to allow for the inductive extension of data and knowledge. It is claimed that only then can a system for automatic programming - that is, one capable of learning - be engineered. This claim follows from the evolutionary approach being equally applicable to all effective processes within the system.

ACKNOWLEDGEMENTS

The author would like to express his gratitude to the ASEE, Irwin Goodman, NOSC, Code 421, Alan Gordon, NOSC, Code 013, John H. Holland, Univ. of Mich., Linwood Sutton, NOSC, Code 411, and Robert Wasilausky, NOSC, Code 411 for their time and respective contributions which served to guide the preparation of this paper. A note of appreciation is also accorded to my colleagues at CMU and my parents and brother.

REFERENCES

1. Arbib, M.A., *A Programming Approach to Computability*, Springer-Verlag, New York, NY, 1982.
2. Biggerstaff, T.J. and Perlis, A.J. (eds.), *Software Reusability Volume I*, Addison-Wesley Publishing Co., New York, New York, 1989.
3. Duke, E.L., Brumbaugh, R.W., and Disbrow, J.D., "A Rapid Prototyping Facility for Flight Research in Advanced Systems Concepts," *Computer*, Vol. 22, No. 5, May 1989, pp. 61-66.
4. Field, A.J. and Harrison, P.G., *Functional Programming*, Addison-Wesley Publishing Company, Inc., Menlo Park, CA, 1988.
5. Graham, P., "A LISP Query Compiler," *AI Expert*, Vol. 4, No. 6, June 1989, pp. 21-26.
6. Holland, J.H., "Adaptive Algorithms for Discovering and Using General Patterns in Growing Knowledge-Bases," *International Journal of Policy Analysis and Information Systems*, Vol. 4, No. 3, Plenum Press, New York, 1980, pp. 245-268.
7. Holland, J.H., "Genetic Algorithms and Classifier Systems: Foundations and Future Directions," *Genetic Algorithms and their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, L. Erlbaum Associates, Hillsdale, New Jersey, 1987, pp. 82-89.
8. Linden, T.A. and Markosian, L.Z., *Transformational Synthesis Using REFINE*, Tech. Report GH4-116847, Reasoning Systems, Inc., Palo Alto, CA, 1988.
9. Markosian, L., Abraido-Fandino, L., and Katzman, S. *Knowledge-Based Software Engineering Using REFINE*, Tech. Report, Reasoning Systems, Inc., Palo Alto, CA, 1988.

10. McClure, C., *CASE is Software Automation*, Prentice Hall, Englewood Cliffs, NJ, 1989.
11. Mitchell, T.M., Keller, R.M., and Kedar-Cabelli, S.T., "Explanation-Based Generalization: A Unifying View," *Machine Learning*, Vol. 1, No. 1, 1986, pp. 47-80.
12. Rubin, S.H., "Modeling High-Level Knowledge: A Survey," Submitted to *AI Review*, June 1989.
13. Zorn, B., Ho, K., Larus, J., Semenzato, L., and Hilfinger, P., "Multiprocessing Extensions in SPUR LISP," *IEEE Software*, Vol. 6, No. 4, July 1989, pp. 41-49.

AUTOMATED EXTRACTION OF KNOWLEDGE FOR MODEL-BASED DIAGNOSTICS

Avelino J. Gonzalez Harley R. Myler
Massood Towhidnejad Frederic D. McKenzie Robin R. Kladke

Department of Computer Engineering
University of Central Florida
Orlando, FL 32816-0450

ABSTRACT

The concept of accessing CAD design databases and extracting a process model automatically is investigated as a possible source for the generation of knowledge bases for model-based reasoning systems. The resulting system, referred to as Automated Knowledge Generation (AKG), uses an object-oriented programming structure and constraint techniques as well as internal database of component descriptions to generate a frame-based structure that describes the model. The procedure has been designed to be general enough to be easily coupled to CAD systems that feature a database capable of providing label and connectivity data from the drawn system. The AKG system is capable of defining knowledge bases in formats required by various model-based reasoning tools.

1.0 INTRODUCTION

The process of knowledge acquisition has been an impeding factor in the growth of knowledge-based systems. For this reason, research in automating the process has attracted the interest of a number of investigators around the world. Although significant progress has taken place (Marcus 89), a significant difficulty has been that the knowledge required for more traditional rule-based systems is of extensive and complex domains and generally found only in the minds of human experts.

The emergence of model-based reasoning techniques in control and diagnosis of electrical, mechanical, and/or process systems has opened an avenue of opportunity in the area of automated knowledge acquisition. The knowledge required in such systems is actually a model representation of the system to be analyzed. This knowledge is not in the form of explicit rules and is extractable from schematic drawings of the target system. When such drawings exist in electronic media such as a Computer-Aided Design (CAD) system, the automation of the knowledge acquisition process simplifies.

In general, CAD databases do not provide all the information necessary to generate a complete knowledge base. Additionally, the lack of constraints placed upon the draftsman doing the drawing requires that acquisition system be able to understand the intent of the process system model and thus make estimates of what the draftsman intends to represent. This process is no

different than that followed by a human process engineer trying to carry out the same task.

This topic is under investigation at the University of Central Florida Department of Computer Engineering. In a three year project funded by NASA-Kennedy Space Center, an objective was set to develop a system capable of generating knowledge bases from CAD databases with minimal human interaction. The prototype system is called the Automated Knowledge Generator (AKG) and is the topic of this paper.

2.0 THE AKG SYSTEM

An Object-Oriented Programming (OOP) approach using the Symbolics Genera 7 LISP machine environment has been taken in the development of AKG. Each component of the target system described in the CAD database is represented as an object within AKG. This approach is intended to model the physical system as closely as possible by representing components as an organized set of discrete objects capable of communication with external processes. In addition, OOP encourages modularity of design, thus making development, modification and enhancement of the system much simpler. The AKG system is divided into eight modules as shown in Figure 1.

The AKG process can be divided to two major tasks. 1) the capture of information which resides in the CAD database, and the creation of an internal model 2) the resolution process, which include the verification of captured knowledge and the generation of missing information.

2.1 Knowledge Acquisition from CAD

At start up, a CAD-generated description of the target system is obtained through the ACCESS module. This module communicates with the computer hosting the CAD system and downloads two files, COMPOC.DAT and TOFROMC.DAT, that must be formatted by the CAD database system. ACCESS uses a command file that contains the unique communication configurations required by the host as well as appropriate database query instructions needed to format the data files. The COMPOC.DAT file contains component details made up of a unique identifier, nomenclature, and possibly other descriptive information such as operating range and units. The TOFROMC.DAT file contains structure data which describes the process component interconnectivity in the system being modelled. The SPAWN module then uses information from the COMPOC.DAT file to create unique component objects within the AKG environment. The CONSTRAINT GENERATOR module sends connectivity information to each of these component objects. The connectivity structure imposed represents an initial constraint set on the system.

Once the CONSTRAINT GENERATOR completes its process, all the available information has been collected from CAD and an internal model is established. This internal model lacks information

regarding the functionality of the source system it represents. In order to accomplish a complete knowledge acquisition, additional modules are called upon to generate the function data. Generation of the function data is termed resolution and is the primary knowledge generation process.

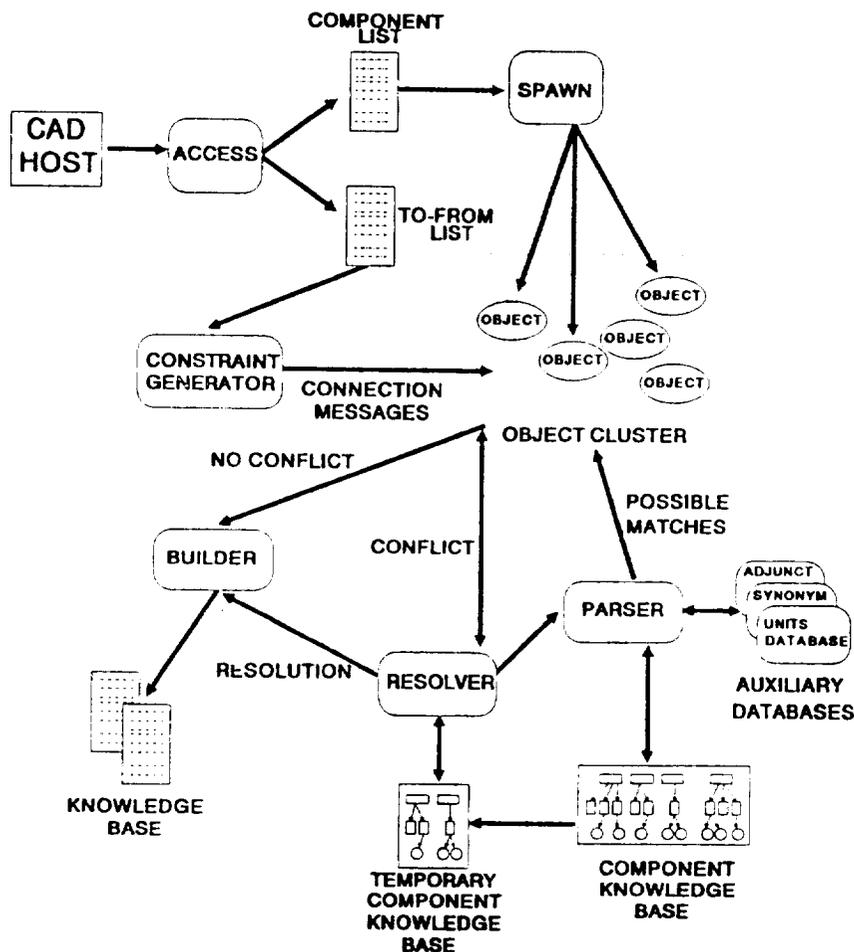


Figure 1. A graphical representation of the AKG process

2.2 The Resolution Process

In order to accomplish the resolution process, AKG uses the PARSER, COMPONENT KNOWLEDGE BASE, and RESOLVER modules.

2.2.1 PARSER

The PARSER provides the first level of identification of the components in the source system. PARSER uses several string matching heuristics (Kladke 89) to search through the COMPONENT

KNOWLEDGE BASE (CKB) in order to find one or more possible matches for each component and the label supplied from the CAD system. PARSER utilizes an internal confidence factor to rank the possible matches. A match confidence of one hundred identifies a perfect match between the source system component label and one found in the CKB. This process is a form of concept learning (Rendell 1987) because a search is made for a measure of graded class inclusion that is consistent with experience, the known CKB objects.

2.2.2 COMPONENT KNOWLEDGE BASE

The descriptive representations of components in the CAD system are not as complete as would be required for the proper operation of a diagnostic and control system. A major deficit to the completeness of some component descriptions, for example, is the lack of output functions. To complete component frames and to further resolution of the flow inconsistencies that exist in the connectivity of the CAD representation, more information is needed. An easily accessible database of generic-type components with a description of their functions and other significant data is the link to complete resolution of the source system.

The role of the COMPONENT KNOWLEDGE BASE is to provide the information necessary to complete the functional description of a component. This information includes the output function, parameters that affect the output function, and parameters that affect the performance of a component such as tolerance and delay. Descriptions of generic components that resemble a particular component in name and nature are stored in a hierarchical internal database. By determining the generic component which best fits the name and nature (analog-component, digital-component, etc.) of the specific component, the vital information known to the generic component such as output function can then be inherited by the specific or instance level component of the internal model to further enrich its own description. It is at this point that the component frame may be complete enough for use with a reasoning tool. More complete component frames also lead to better opportunities to resolve flow inconsistencies. The quantity and quality of information inherited depends on the degree of accuracy of the match. Generic components in the CKB are stored as frames and, when accessed, are spawned into internal objects. As an object, each component possesses its own identity and function.

The conceptual structure of the CKB is a list of top level generic components in the knowledge base that access more successively descriptive components. Upper level components constitute types of devices and have information that govern the accepted behavior of these device types. This information is carried through to the children of these upper level devices as a result of inheritance during the spawning process.

The process of CKB access can be broken into the following stages:

1. The path of a component (i.e., the generic component to be accessed along with its ancestors to the top level) is given as an argument to the access function.
2. The access function retrieves the generic component from storage and allows inheritance from its parents.
3. Once retrieved, the generic component is spawned into the AKG world the AKG world as a generic comp
4. A list of the children of this component is returned.
5. These children are used by the PARSER to further add to the depth of the path to be accessed.
6. Each accessed component is then noted within a global list that serves as a temporary component knowledge base for later use by RESOLVER.

An editor is provided that allows direct user modification of the CKB. This utility has many features including editing of both actual storage frames of generic components and spawned generic components.

An extension to the CKB is the implementation of a constraint representation scheme which will encompass process knowledge for generic components. AKG uses the criterion that process and control system components must have similar, and sometimes identical, properties. The idea is to interrelate components that belong to certain process system classes (such as electrical, pneumatic, flow, etc.). For example, one never connects a logic gate to a pressure valve. The CKB provides these properties as constraints of the components. This knowledge base contains general domain knowledge concerning component details and system aspects of process control. Such information will not only include standard values for tolerance, delay and transfer function for each generic component represented, but also will include constraints indicating which components may be validly connected. The availability of process knowledge allows the primary constraint propagation mechanism (Resolver) to further identify and select the best generic component and transfer function for a specific CAD component.

2.2.3 RESOLVER

The Resolver examines components in the system to establish an initial confidence factor for each. Each slot in the internal object cluster is assigned a weight (e.g., the OUTPUT-FUNCTION and NOMENCLATURE slots have the weight of 20, and the RANGE and TOLERANCE slots have the weight of 5). These weights are based on the amount of importance a particular slot has in determining

the final identification/operation of the system. The initial CF for each component is computed by summing the weights of those slots that are filled directly from the CAD database. This value represents the level of information that a component has about itself with respect to all other objects in the system. A global threshold for the confidence factor is established by the user which, when exceeded, flags a component as ready for conversion into a knowledge-base frame. If a component's confidence factor does not reach the minimum threshold due to lack of information, the RESOLVER module is called to deduce the correct identification from the CKB. The confidence factors at each object are not independent. This is a significant difference from the way CF's are used in rule-based systems. No single CF, CF cluster or CF sequence can dominate the final outcome of the resolution process.

The RESOLVER calls PARSER with the list of inadequately identified components. Upon completion, PARSER adds a list to the POSSIBLE-MATCH slot of each component flavor for which a match was found. This list includes the component matched within the CKB and a parse confidence factor that reflects the certainty of the match. The RESOLVER searches the temporary CKB which is produced during the parsing process as a result of accessing the components in the CKB (see section 2.2.2), for the match with the highest parse confidence.

Once this component is found in the temporary CKB, the RESOLVER attempts to verify the match between the component in the system and the generic component from the CKB. This is accomplished by comparing the slot values (i.e., values for UNITS, RANGE, allowed/possible upstream (INPUTS) and downstream (OUTPUTS) components, etc.) for the component and the generic component. If a match is confirmed, the RESOLVER supplies the information missing from the system component with the information contained in the generic component from the CKB. The act of adding information to a component flavor causes an immediate increase in the confidence factor of that component.

If a match between the component and its best possible match can not be supported, the RESOLVER will attempt to match against the remaining components in the list of POSSIBLE-MATCHES. If still unsuccessful in finding a match, the RESOLVER attempts to match the component with the parents of the possible matches, starting again at the best match. As it was discussed in section 2.2.2. the parent of a generic component in CKB is a more general form of its children. In this case the RESOLVER relaxes the constraints on the possible match. If a match is found between the component and the parent of a possible match, it would be advisable to try to find a match between the component and the parent's alternative children (i.e., siblings to the possible match). Therefore, in this situation the RESOLVER again would tighten the constraints on the possible match.

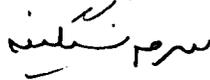
AKG compares, using the relaxation algorithm and the CKB, the validity of the system component connections. When a component is flagged as valid, AKG is then able to assign a function to it that is consistent with the target reasoning system. This approach to conflict resolution using the reasoning mechanism of constraint propagation raises the AKG system well above the capability level of a simple translator.

In summary, once the RESOLVER is called, all the components in the system are examined and the components with the highest confidence factors are marked. Based on the information (i.e., constraints) in these marked components, the propagation of confidence proceeds beginning with neighboring components. The propagation of confidence factors is global in the system and continues until all the components' confidence factors change less than some preassigned rate of convergence. At that time, the system's confidence factor is considered settled. The RESOLVER then scans all the components in the system and flags the components with confidence factors below the user-defined threshold. As a last resort, the RESOLVER asks the user to supply new information and confidence factors for these flagged components. This resolution process repeats until all the components' confidence factors exceed the threshold value.

3.0 TRANSLATION VERSUS INTELLIGENT INTERPRETATION

The following example identifies the difference between translation and interpretation using the conversion of a sentence from one language to another.

The original sentence (in Persian):



The literal (English) translation:

My head is heavy.

However, the correct interpretation of the sentence is:

I have a hangover.

The AKG system provides many advantages over a direct translation approach. A knowledge base translator is capable only of uncritically reformatting information explicit within its input data. An intelligent interpreter, however, is able to extend and correct input by inferring missing values and resolving conflicts. This ability is necessary for automated knowledge generation in the presence of sparse data such as that available from a CAD system.

The AKG prototype took as its testbed a demonstration circuit for purging pneumatic systems called the "Purge Demo." The knowledge base for this system had been manually constructed

by NASA for verification using the Knowledge-Based Autonomous Test Engineer (KATE). Early work using a translator (Thomas 87) had indicated that translation is not sufficient for the resolution of CAD data into a knowledge base. A test of the AKG system was thus to autonomously produce a knowledge base which would closely approximate its human-generated counterpart.

The results of both studies are listed in Table 1. A description of the KATE slots depicted in the table may be found elsewhere [Cornell 87, Gonzalez et al. 88]. Note that the translation approach was found to be unable to provide any values for some KATE slots and it predicted a relatively low potential capability to fill others. In each of these cases the AKG intelligent interpretation approach is found to be superior. The component information of the CKB coupled with heuristic driven parsing will enable slots AN-ELEMENT-OF (AEO), TOLERANCE, DELAY, and STATUS (transfer function) to be filled at least 75% of time. It is estimated that the process information coupled with the TO-FROM list will allow identification of 90% to 100% of the SOURCE-PATH, IN-PATH-OF, SOURCE, and SINK slots.

Slots	Translation Results			AKG Results		
	Filled Auto.	Percent	Est. of Pot. Cap.	Filled Auto.	Percent	Est. of Pot. Cap.
aio	52/52	100%	100%	26/26*	100%	100%
aeo	NA	NA	NA	14/14*	100%	> 75%
nomenc1.	0/52	0%	100%	13/13*	100%	100%
source-path	8/52	15%	50%	25/25#	100%	> 90%
in-path	52/52	100%	> 90%	35/35	100%	> 90%
source	2/2	100%	> 80%	NA	NA	100%
tolerance	NA	NA	NA	2/2*	100%	75%
delay	0/3	0%	0%	3/3*	100%	75%
status	0/52	0%	50%	6/6*	100%	75%
units	23/23	100%	100%	23/23	100%	100%
range	16/16	100%	100%	15/15	100%	100%
sinks	2/2	100%	80%	NA	NA	100%

Notes: (*) Filled with the help of the component database
 (#) Need special operators to get this result.

Table 1. Comparison of Results.

4.0 CONCLUSIONS

This paper has discussed the structure and the operation of the Automated Knowledge Generator (AKG) system. It has been shown that a simple translator would not be sufficient to generate a viable knowledge base for a diagnostic system. An intelligent interpreter such as AKG is needed in order to accomplish the task of automatic knowledge acquisition from CAD databases. Work on the AKG system is continuing with work focussing on using CAD descriptions from a number of varied sources. These include Shuttle Ground Support subsystems, power generation systems, and Advanced Launch System processes.

ACKNOWLEDGEMENTS

The AKG research project is supported by NASA, Kennedy Space Center under contract NAG-10-0043.

REFERENCES

- [Cornell 87] M. Cornell, "The KATE Shell - An implementation of Model-Based Control, Monitor and Diagnosis", Proceedings of the First Workshop on Space Operations Automation and Robotics, Houston, Texas, 1987.
- [Gonzalez 88] A. J. Gonzalez, H. R. Myler, B. C. Owen, and M. Towhidnejad, "Automated Generation of Knowledge from CAD Design Data Bases", Proceedings of the First Florida Artificial Intelligence Research Symposium, Orlando, Florida, 1988.
- [Gonzalez 89] A. J. Gonzalez, H. R. Myler, and M. Towhidnejad, "Automated Knowledge Generation in Support of Shuttle Ground Operations", Proceedings from the Artificial Intelligence in Government Conference, Washington, D.C., 1989.
- [Kladke 89] R. Kladke, "A Mega-heuristic Approach to the Problem of Component Identification in Automated Knowledge Generation", Masters Thesis, University of Central Florida, November, 1989.
- [Marcus 89] S. Marcus and J. McDermott, "SALT: A Knowledge Acquisition Language for Proposed-and-revised Systems" Artificial Intelligence, Vol. 39. No. 1. 1989. pp. 1-37.
- [Rendell 85] L. Rendell, "Substantial Constructive Induction Using Layered Information Compression: Tractable Feature Formation in Search", Proceedings of the Ninth International Joint Conference on Artificial Intelligence, Vol. 1., Los Angeles, CA, 1985.
- [Thomas 87] S. J. Thomas, "Automated Construction of a Knowledge Base from Computer Aided Design Data", NASA Internal technical report, Artificial Intelligence Section, 1987.



Derivation of Sorting Programs

Joseph Varghese
QTC Horizon Labs
8700 SW Creekside Pl., Suite D
Beaverton, OR 97005

Rasih Loganantharaj
The Center for Advanced Computer Studies
USL, P.O.Box 44330
Lafayette, LA 70504

Abstract

Program synthesis for critical applications has become a viable alternative to program verification. We use nested resolution and its extension to synthesize a set of sorting programs from their first order logic specifications. We have successfully synthesized a set of sorting programs, such as, naive sort, merge sort, and insertion sort, starting from the same set of specifications.

1 Introduction

The important phases of a software life cycle include requirement acquisition, development of algorithms, implementation, verification and maintenance. Usually, the execution performance is an expected requirement in a software development process. Unfortunately, the verification and the maintenance of programs are the time consuming and the frustrating aspects of software engineering. The verification can not be wavered for the programs used for critical applications such as, military, space, and nuclear plants. As a consequence, synthesis of programs from specifications, an alternative way of developing correct programs, is becoming popular.

There are three basic approaches for program synthesis: theorem proving [5, 6, 8], program transformation [1, 2] and problem solving [3]. In the theorem proving approach, a target program is constructed incrementally at each step of the proof whereas in the transformational approach, inference rules and transformation rules are applied to the specifications and to the derived sentences until the target program is realized. Synthesis systems based on problem solving methods are inflexible as compared to the other two methods. However, they tend to be very effective in the domain in which they operate

In this paper, we do not concern ourselves with the problem acquisition phase of automatic programming. Specification acquisition and subsequent refinement is a research problem in its own right. Assuming that the program is specified in first-order predicate logic, we describe the derivation of logic programs for sorting. In section 2, we provide a brief review of nested

resolution and its application to program synthesis. In Section 3, we describe the specification and the derivation of sorting programs, and it is followed by a summary and discussion.

2 A review of nested resolution and its application to program synthesis

We start with some notations. Let $F[P]$ denote a well-formed formula (wff) containing one or more occurrences of a sub-wff P . Then, a new wff obtained by replacing all occurrences of P by Q is denoted by $F[P/Q]$.

We give an informal definition of polarity. For a rigorous definition the reader may refer to [7, 5]. A sub-wff P has a positive (negative) polarity in $F[P]$ if and only if (iff) P occurs within an even (odd) number of explicit or implicit negations. The positive polarity and the negative polarity are written as $F[P^+]$ and $F[P^-]$, respectively. If P occurs within an equivalence connective or within the if clause of an if-then-else connective, then P has a positive-negative polarity and is written as $F[P^\pm]$.

2.1 Inference Rules

We [4] have proposed nested resolution [9] and its extension for logic program synthesis from first-order specifications. The nested resolution is a variation of nonclausal resolutions. The reader may refer to [9] for more details. Inference rules are applied to a pair of statements: a statement to be transformed which we call a *transformee*, and a statement used for transformation which we call a *transformer*. The transformer may be an axiom, a transformation rule or a lemma. The transformee is initially an axiom from the specification set, and subsequently, it may be the result of an earlier transformation or a lemma. In every transformation, a sub-wff of the transformee is replaced by another sub-wff that is determined by the transformer.

$$\frac{F[P^+] \quad G[P']}{F[P^+\theta/G\theta[P'\theta/true]]} \qquad \frac{F[P^-] \quad G[P']}{F[P^-\theta/\neg G\theta[P'\theta/false]]}$$

Where θ is the most general unifier (m.g.u.) of P and P' . That is $P\theta = P'\theta$. Here the wffs F and G are the transformee and the transformer, respectively.

Let us consider an example to explain the inference rule.

$$\frac{P(X, Y) \wedge Q(Y, Z) \rightarrow R(X, Z) \quad S(X', Y') \rightarrow P(X', Y')}{\neg(S(X, Y) \rightarrow false) \wedge Q(Y, Z) \rightarrow R(X, Z)}$$

Where θ is $\{X'/X, Y'/Y\}$. The expression can be simplified to $S(X, Y) \wedge Q(Y, Z) \rightarrow R(X, Z)$

2.1.1 Some special cases

Here we describe some special cases of nested resolution. These rules are handy when deriving programs by hand. To use these rules, polarities of the transformer and the transformee should be followed strictly.

$$\begin{array}{ccc}
 \frac{F[P^+]}{P'^-} & \frac{F[P^-]}{P'^+} & \frac{F[P]}{P' \leftrightarrow Q'} \\
 \hline
 F[P^+\theta/false] & F[P^-\theta/true] & F[P\theta/Q'\theta]
 \end{array}$$

where θ is the m.g.u. of P and P' .

2.2 Inference rules in the presence of explicit quantifications

In refutation proof procedures, an existential quantifier is replaced by either a Skolem constant or a Skolem function. Replacing an existentially quantified variable by a Skolem constant or a Skolem function is not acceptable in transformational program synthesis methods [4] because we will lose some valuable information in the course of that replacement. To overcome the problem, we extend the nested resolution to handle quantified wffs. To avoid inadvertent problems during unification, all variables in both the transformer and the transformee are renamed at each step. The following condition that checks for possible scoping violations must be satisfied when existentially quantified variables are unified.

Condition QS: (Quantified variable Substitution)

- An existentially quantified variable, say X , within the scope of a universally quantified variable, say Y , cannot be unified to the same universally quantified variable. (That is, X cannot be unified with Y . This is usually detected by occur check in Skolemized version of the quantified wffs)
- Two existentially quantified variables cannot be unified.

Example

$$\begin{array}{l}
 \forall X \exists Y P(X, Y) \\
 P(X', X') \rightarrow Q(X')
 \end{array}$$

X' is unified to X but we cannot unify X with Y since it violates the QS-condition.

The extension to the nested resolution for quantified wffs are given as following:

1. If the transformer is quantifier free and the transformee has an existentially quantified variable then the nested resolution is applied in the same way as it is applied to the quantifier free case, provided that the condition QS is not violated during unification. Consider an example

$$\frac{\exists Y \forall X P(X, Y) \vee R(X, Y) \quad P(X', Y') \rightarrow Q(X', Y')}{\exists Y \forall X (true \rightarrow Q(X, Y)) \vee R(X, Y)}$$

which simplifies to $\exists Y \forall X Q(X, Y) \vee R(X, Y)$

2. If the transformee is quantifier free and the transformer has an existentially quantified variable then the nested resolution is applied in the same way as it is applied to the quantifier free case, provided that the condition QS is not violated. Consider an example.

$$\frac{P(X, Y, Y) \rightarrow Q(X, Y, Y) \quad \forall X' \exists Y' \forall Z' P(X', Y', Z') \vee R(X', Y', Z')}{\forall X \exists Y (\neg(false \vee R(X', Y', Y')) \rightarrow Q(X', Y', Y'))}$$

which simplifies to $\forall X \exists Y R(X, Y, Y) \vee Q(X, Y, Y)$

3. When the transformer and the transformee have existential quantified variables, the extension to the nested resolution becomes complicated. Since, such case is not common in program synthesis, it is not considered here.

Transformation Rules

Transformation rules are usually second-order wffs which have variable predicates. These rules are used to simplify derived sentences or specifications. We provide some of the transformation rules used in this paper.

$$\begin{aligned} P &\leftrightarrow P \\ P_1 \vee P_2 \vee P_3 &\leftarrow P_3 \end{aligned}$$

2.3 Organization of Derivations

As indicated earlier, the specification consists of a set of statements in first-order logic. The synthesis system transforms these statements into a set of Horn clauses that constitute an executable program. At each step of the derivation, the transformee and the transformer statements interact to produce a result. Initially, the transformee is one of the statements from the specification set; later the transformee is one of the intermediate results of the derivation. The transformer can be a statement from the specification set, an intermediate result, a transformation rule or a simplification rule. Simplification rules may have predicate variables, in which case higher-order unification is assumed. In our derivations all the transformees and the transformers are shown at the left and the right hand sides respectively. The sub-wff of

the transformee to be transformed is underlined while the sub-wff of the transformer that is used for transformation is overlined. After the nested resolution is applied to each transformee and transformer pair, the resulting wff is simplified and only the simplified wff is shown in the derivation.

2.4 Controlling the Inference

Logic program synthesis may be viewed as a process that creates executable Horn clauses for each predicate appearing in the specification. This view forms the basis of our strategy and provides a mean for detecting missing knowledge in the specification. We arrange the predicates appearing in the specification in the order in which the executable Horn procedures are derived. The derivation starts with the first predicate and continues till the end of the list. Once we have derived all the executable Horn clauses for all the predicates in the list, the synthesis completes successfully.

It is well known that all the first order sentences cannot be transformed into Horn Logic. However, a procedure which is not in Horn Logic can be transformed into an executable Horn clause form either by introducing recursion or by interpreting negation as failure. This is why we were able to transform the first order specifications into an executable Horn clauses.

We use the following procedure to control the derivation.

1. For each Predicate P appearing in the specification do the following.
 - (a) For each, if half of the definition of P, do the following:
 - i. If the body has a universal quantifier, select a literal within the scope of the quantifier such that there exists a transformation that will enable us to apply induction and hence introduce the recursion. Introduction of recursion will usually transform a non-Horn clause into a Horn clause. Then establish the base case for the induction using ground terms of the body.
 - ii. Check whether the Horn clause is executable. If not, transform the literals of the body until an executable Horn clause form is obtained.

From the *if and only if* definition of a predicate P, we can easily obtain the if half of the definition. That is, from $P \leftrightarrow \text{body}$ we can get $P \leftarrow \text{body}$. If we have disjunctive literals as the head of the *if half*, then interpreting negation as failure, we can obtain the if half of P. That is, from $P \vee Q \leftarrow \text{body}$ we obtain $P \leftarrow \text{not}Q, \text{body}$.

3 Specification and Derivation of Sorting Programs

In this section we provide specifications for sorting program and derive different sorting programs starting from the same specifications. Let us define a relation $\text{sort}(x,y)$ which holds when y is a sorted permutation of x . The corresponding specifications are

$$\text{sort}(x, y) \leftrightarrow \text{perm}(x, y), \text{ordered}(y)$$

$$\begin{aligned} perm(x, y) &\leftrightarrow \forall u \exists z (occurs(u, z, x) \leftrightarrow occurs(u, z, y)) \\ ordered(y) &\leftrightarrow \forall u \forall v (precedes(u, v, y) \rightarrow u \leq v) \end{aligned}$$

The second statement is interpreted as stating that y is a permutation of x , if for every element u , x and y contain exactly the same number of occurrences of u . The third statement specifies the *ordered* relation. y is ordered if and only if, for every two elements u and v in y , if u precedes v in the list, then u is less than or equal to v in magnitude.

The following statements specify the *occurs* relation.

$$\begin{aligned} occurs(u, z, nil) &\leftrightarrow z = 0 \\ (occurs(u, z, x) \leftrightarrow occurs(u, z_1, x_1), occurs(u, z_2, x_2), z_1 + z_2 = z) \\ &\leftarrow union(x_1, x_2, x) \end{aligned}$$

According to the first statement, the empty list contains no occurrences of u and according the second statement, if x can be split up into two subsets x_1 and x_2 , then the total number of occurrences of u will remain the same. An element u precedes an element v in the list x if it occurs before v in x . The *precedes* relation is specified as

$$\begin{aligned} \neg precedes(u, v, nil) \\ \neg precedes(u, v, x.nil) \\ (precedes(u, v, x) \leftrightarrow precedes(u, v, x_1) \vee precedes(u, v, x_2) \vee (u \in x_1, v \in x_2)) \\ \leftarrow append(x_1, x_2, x) \end{aligned}$$

The first two statements indicate when the *precedes* relation cannot hold. In the third statement, the list is broken down into two sublists and the relation recursively applied to these sublists. If u precedes v in x , then it must precede it in either of the sublists if both u and v are in that sublist. Otherwise, u is in the first sublist and v is in the second sublist.

We have used the relations *union* and *append* and have not indicated how these are defined and how they differ. The relation *union* is not the same as the union operation on sets. The result z of a *union* operation on lists x and y may contain duplicate elements. Thus the relation *union*($a.nil, a.nil, a.nil$) does not hold whereas *union*($a.nil, a.nil, a.a.nil$) does. The relation *append* is the familiar list append relation. The difference between *union* and *append* is that *append* respects the order of the elements of the appended lists, whereas *union*(x, y, z) just says that z is a permutation of the result of appending x and y . We do not explicitly specify these familiar relations, but instead use their properties which are listed below.

$$\begin{aligned} append(nil, y, y) \\ append(u.x, y, u.z) \leftarrow append(x, y, z) \\ union(x, y, z) \leftarrow append(x, y, z) \\ union(x, y, z) \leftarrow union(y, x, z) \\ union(u.x, y, u.z) \leftarrow union(x, y, z) \end{aligned}$$

From these properties, we can easily derive the following statements as lemmas.

$$append(u.nil, y, u.y)$$

$union(u.nil, y, u.y)$
 $union(nil, y, y)$
 $union(x, v.y, v.z) \leftarrow union(x, y, z)$

We are now ready to tackle program derivations. We begin with programs and lemmas for *perm* and *ordered*.

1. $perm(x, y) \leftrightarrow \forall u \exists z (\underline{occurs(u, z, x)}_{(q\pm)} \leftrightarrow occurs(u, z, y))$
 $\overline{occurs(u, z, nil)} \leftrightarrow z = 0$
2. $perm(nil, y) \leftrightarrow \forall u \exists z (z = 0 \leftrightarrow \underline{occurs(u, z, y)}_{(q\pm)})$
 $\overline{occurs(u, z, nil)} \leftrightarrow z = 0$
3. $perm(nil, nil) \leftrightarrow \forall u \exists z (z = 0 \leftrightarrow z = 0)$
4. $perm(nil, nil)$

This forms a Horn clause for the trivial case when the input list is empty. The following useful lemma on *perm(x, y)* is assumed. For the derivation of this lemma see [10].

1. $perm(x, y) \leftarrow union(x_1, x_2, x), perm(x_1, y_1), perm(x_2, y_2), union(y_1, y_2, y)$

With appropriate procedures for *union*, this can be used as a Horn clause procedure for *perm*. We now derive a few lemmas.

1. $perm(x, y) \leftarrow \forall u \exists z (\underline{occurs(u, z, x) \leftrightarrow occurs(u, z, y)}_{(q-)})$
 $\overline{P \leftrightarrow P}$
2. $perm(x, x)$

This lemma comes in handy when we want to eliminate extra terms by unifying them. The following lemma is used when attempting to unify elements within lists.

1. $perm(x, y) \leftarrow \underline{union(x_1, x_2, x)}_{(-)},$
 $perm(x_1, y_1), perm(x_2, y_2), \underline{union(y_1, y_2, y)}_{(-)}$
 $\overline{union(u.nil, y, u.y)}$
2. $perm(u.x, v.y) \leftarrow \underline{perm(u.nil, v.nil)}_{(-)}, perm(x, y)$
 $\overline{perm(x, x)}$
3. $perm(u.x, u.y) \leftarrow perm(x, y)$

Other results that we use are

$(u \in x \leftrightarrow u \in y) \leftarrow perm(x, y)$
 $perm(x, y) \leftarrow perm(x, l), perm(l, y)$

These statements cannot be derived from our specifications for *perm*. They can be derived if we use different specifications, but then the other derivations become more difficult. We prefer

2. $ordered(y) \leftarrow ordered(y_1), ordered(y_2), lessall(y_1, y_2), append(y_1, y_2, y)$
3. $lessall(x, y) \leftrightarrow \forall u \forall v ((u \in x, v \in y) \rightarrow u \leq v)$

The relation $lessall(x, y)$ holds if all the elements in list x are less than or equal to all elements in list y . We will derive programs and lemmas for $lessall$ shortly.

1. $ordered(y) \leftarrow ordered(y_1), ordered(y_2),$
 $\forall u \forall v (u \in y_1, u \in y_2 \rightarrow u \leq v), \underline{append(y_1, y_2, y)}_{(-)}$
 $\overline{append(u.nil, v, u.v)}$
2. $ordered(x.y_2) \leftarrow ordered(x.nil), ordered(y_2),$
 $\forall u \forall v (u \in x.nil, v \in y_2 \rightarrow u \leq v)$
3. $ordered(x.y_2) \leftarrow ordered(x.nil), ordered(y_2),$
 $\underline{\forall v (v \in y_2 \rightarrow x \leq v)}$
 formation of procedure $lessall'$
4. $ordered(x.y_2) \leftarrow \underline{ordered(x.nil)}_{(-)}, ordered(y_2), lessall'(x, y_2)$
5. $ordered(x.y_2) \leftarrow ordered(y_2), lessall'(x, y_2)$
 $\overline{ordered(x.nil)}$

We will use $lessall'$ in the derivations for $lessall$. It is defined as

$$lessall'(x, y) \leftrightarrow \forall v (v \in y \rightarrow x \leq v)$$

The useful statements that we have derived about the $ordered$ relation are

- $$ordered(nil)$$
- $$ordered(x.nil)$$
- $$ordered(x) \leftarrow ordered(x_1), ordered(x_2),$$
- $$\forall u \forall v (u \in x_1, v \in x_2 \rightarrow u \leq v), append(x_1, x_2, x)$$
- $$ordered(x) \leftarrow ordered(x_1), ordered(x_2), lessall(x_1, x_2), append(x_1, x_2, x)$$
- $$ordered(x.y) \leftarrow ordered(y), lessall'(x, y)$$

We still have to derive a few procedures and lemmas for $lessall'$ and $lessall$. We begin with $lessall$.

1. $lessall(x, y) \leftarrow \forall u \forall v ((\underline{u \in x}_{(q+)}, v \in y) \rightarrow u \leq v)$
 $\overline{\neg u \in nil}$
2. $lessall(nil, y)$
3. $lessall(x, y) \leftarrow \forall u \forall v ((u \in x, \underline{v \in y}_{(q+)}) \rightarrow u \leq v)$
 $\overline{\neg u \in nil}$
4. $lessall(x, nil)$

These are the base case procedures for $lessall$.

1. $lessall(x, y) \leftarrow \forall u \forall v ((\underline{u \in x}_{(q+)}, v \in y) \rightarrow u \leq v)$

$$\overline{u \in v.y} \leftrightarrow u = v \vee u \in y$$

$$2. \text{ lessall}(z.x', y) \leftarrow \forall u \forall v ((u = z \vee u \in x'), v \in y) \rightarrow u \leq v$$

$$3. \text{ lessall}(z.x', y) \leftarrow$$

$$\quad \forall u \forall v (u = z, v \in y \rightarrow u \leq v),$$

$$\quad \forall u \forall v (u \in x', v \in y \rightarrow u \leq v)$$

$$4. \text{ lessall}(z.x', y) \leftarrow$$

$$\quad \frac{\forall v (v \in y \rightarrow z \leq v)}{(-)}$$

$$\quad \forall u \forall v (u \in x', v \in y \rightarrow u \leq v)$$

$$\text{lessall}'(u, x) \leftrightarrow \overline{\forall v (v \in x \rightarrow u \leq v)}$$

$$5. \text{ lessall}(z.x', y) \leftarrow \text{lessall}'(z, y),$$

$$\quad \frac{\forall u \forall v (u \in x', v \in y \rightarrow u \leq v)}{(-)}$$

$$\text{lessall}(x, y) \leftarrow \overline{\forall u \forall v (u \in x, v \in y \rightarrow u \leq v)}$$

$$6. \text{ lessall}(z.x', y) \leftarrow \text{lessall}'(z, y), \text{lessall}(x', y)$$

This statement can be used as a procedure for *lessall*. We now proceed with the derivation of lemmas.

$$1. \text{ lessall}(x, y) \leftarrow \forall u \forall v ((u \in x, v \in \underline{y}_{(q+)}) \rightarrow u \leq v)$$

$$\quad (\overline{u \in z} \leftrightarrow u \in z_1 \vee u \in z_2) \leftarrow \text{union}(z_1, z_2, z)$$

$$2. \text{ lessall}(x, y) \leftarrow \forall u \forall v (u \in x, (v \in y_1 \vee v \in y_2) \rightarrow u \leq v), \text{union}(y_1, y_2, y)$$

$$3. \text{ lessall}(x, y) \leftarrow$$

$$\quad \frac{\forall u \forall v (u \in x, v \in y_1 \rightarrow u \leq v)}{(-)}$$

$$\quad \frac{\forall u \forall v (u \in x, v \in y_2 \rightarrow u \leq v)}{(-)}, \text{union}(y_1, y_2, y)$$

$$\quad \text{lessall}(x, y) \leftrightarrow \overline{\forall u \forall v ((u \in x, v \in y) \rightarrow u \leq v)}$$

$$4. \text{ lessall}(x, y) \leftarrow \text{lessall}(x, y_1), \text{lessall}(x, y_2), \frac{\text{union}(y_1, y_2, y)}{(-)}$$

$$\quad \overline{\text{union}(v.nil, u, v.u)}$$

$$5. \text{ lessall}(x, u.y_2) \leftarrow \text{lessall}(x, u.nil), \text{lessall}(x, y_2)$$

Next, we derive a lemma that states that the *lessall* relation is not changed by permuting x or y .

$$1. \text{ lessall}(x, y) \leftarrow \forall u \forall v ((u \in x, v \in \underline{y}_{(q+)}) \rightarrow u \leq v)$$

$$\quad (\overline{u \in x} \leftrightarrow u \in y) \leftarrow \text{perm}(y, x)$$

$$2. \text{ lessall}(x, y) \leftarrow \forall u \forall v ((u \in \underline{x}_{(q+)}, v \in z) \rightarrow u \leq v), \text{perm}(z, y)$$

$$\quad (\overline{u \in x} \leftrightarrow u \in y) \leftarrow \text{perm}(y, x)$$

$$3. \text{ lessall}(x, y) \leftarrow \frac{\forall u \forall v ((u \in w, v \in z) \rightarrow u \leq v)}{(-)}, \text{perm}(z, y), \text{perm}(w, x)$$

$$\quad \text{lessall}(x, y) \leftrightarrow \overline{\forall u \forall v ((u \in x, v \in y) \rightarrow u \leq v)}$$

$$4. \text{ lessall}(x, y) \leftarrow \text{lessall}(w, z), \text{perm}(z, y), \text{perm}(w, x)$$

Another lemma that can be derived for *lessall* lets us exploit the transitivity property of *lessall*.

$$\text{lessall}(x, y) \leftarrow \text{lessall}(x, z), \text{lessall}(z, y)$$

We now begin on the derivations of programs and lemmas for *lessall'*.

1. $\text{lessall}'(x, y) \leftarrow \forall v(\underline{v \in y}_{(q+)}) \rightarrow x \leq v$ $\overline{\neg u \in \text{nil}}$
2. $\text{lessall}'(x, \text{nil})$

This forms the base case procedure for *lessall'*.

1. $\text{lessall}'(x, y) \leftarrow \forall v(\underline{v \in y}_{(q+)}) \rightarrow x \leq v$ $\overline{u \in v.y} \leftrightarrow u = v \vee u \in y$
2. $\text{lessall}'(x, u.y') \leftarrow \forall v(v = u \vee v \in y' \rightarrow x \leq v)$
3. $\text{lessall}'(x, u.y') \leftarrow \forall v(v = u \rightarrow x \leq v), \forall v(v \in y' \rightarrow x \leq v)$
4. $\text{lessall}'(x, u.y') \leftarrow x \leq u, \overline{\forall v(v \in y' \rightarrow x \leq v)}_{(-)}$ $\text{lessall}'(x, y) \leftarrow \overline{\forall v(v \in y \rightarrow x \leq v)}$
5. $\text{lessall}'(x, u.y') \leftarrow x \leq u, \text{lessall}'(x, y')$

This clause along with the base case derived earlier can be used as procedures for *lessall'*. We proceed with the derivation of lemmas for *lessall'*.

1. $\text{lessall}'(x, y) \leftarrow \forall v(v \in y \rightarrow \underline{x \leq v}_{(q-)})$ $\overline{u \leq v} \leftarrow u \leq w, w \leq v$
2. $\text{lessall}'(x, y) \leftarrow \overline{\forall v(v \in y \rightarrow w \leq v)}_{(-)}, x \leq w$ $\text{lessall}'(x, y) \leftrightarrow \overline{\forall v(v \in y \rightarrow x \leq v)}$
3. $\text{lessall}'(x, y) \leftarrow \text{lessall}'(w, y), x \leq w$

We now derive a lemma that links *lessall'* with *ordered*.

4. $\text{ordered}(y) \leftrightarrow \forall u \forall v(\underline{\text{precedes}(u, v, y)}_{(q+)}) \rightarrow u \leq v$
 $\overline{(\text{precedes}(u, v, x))} \leftrightarrow \text{precedes}(u, v, x_1) \vee \text{precedes}(u, v, x_2)$
 $\vee (u \in x_1, v \in x_2)$
 $\leftarrow \text{append}(x_1, x_2, x)$
5. $(\text{ordered}(y) \leftrightarrow \forall u \forall v((\text{precedes}(u, v, y_1) \vee \text{precedes}(u, v, y_2) \vee (u \in y_1, v \in y_2))$
 $\rightarrow u \leq v)) \leftarrow \text{append}(y_1, y_2, y)$
6. $(\text{ordered}(y) \rightarrow$
 $\overline{\forall u \forall v((\text{precedes}(u, v, y_1) \vee \text{precedes}(u, v, y_2) \vee (u \in y_1, v \in y_2))}_{(q-)} \rightarrow u \leq v)) \leftarrow \text{append}(y_1, y_2, y)$ $\overline{(P_1 \vee P_2 \vee P_3)} \leftarrow P_3$

7. $(ordered(y) \rightarrow \forall u \forall v ((u \in y_1, v \in y_2) \rightarrow u \leq v)) \leftarrow \overline{append(y_1, y_2, y)}_{(-)}$
 $\overline{append(u.nil, v, u.v)}$
8. $ordered(x.y') \rightarrow \forall u \forall v ((u \in x.nil, v \in y') \rightarrow u \leq v)$
9. $ordered(x.y') \rightarrow \overline{\forall v ((v \in y') \rightarrow x \leq v)}_{(+)}$
 $lessall'(x, y) \leftarrow \overline{\forall v (v \in y \rightarrow x \leq v)}$
10. $ordered(x.y') \rightarrow lessall'(x, y')$

which can be rewritten as

$$lessall'(x, y') \leftarrow ordered(x.y')$$

We use the previous two lemmas in the proof of the next lemma.

1. $lessall'(x, y) \leftarrow \overline{lessall'(w, y)}_{(-)}, x \leq w$
 $\overline{lessall'(x, y')} \leftarrow ordered(x.y')$
2. $lessall'(x, y) \leftarrow ordered(w.y), x \leq w$

We have only a couple more lemmas to go before starting with actual sorting programs.

1. $lessall'(x, y) \leftarrow \forall v (\underline{v \in y}_{(q+)}) \rightarrow x \leq v$
 $(\overline{u \in x} \leftrightarrow u \in y) \leftarrow perm(y, x)$
2. $lessall'(x, y) \leftarrow \overline{\forall v (v \in z \rightarrow x \leq v)}_{(-)}, perm(z, y)$
 $lessall'(x, y) \leftarrow \overline{\forall v (v \in y \rightarrow x \leq v)}$
3. $lessall'(x, y) \leftarrow lessall'(x, z), perm(z, y)$

And the last lemma is just as simple.

1. $lessall'(x, y) \leftarrow \forall v (\underline{v \in y}_{(q+)}) \rightarrow x \leq v$
 $(\overline{v \in y} \leftrightarrow (v \in y_1 \vee v \in y_2)) \leftarrow union(y_1, y_2, y)$
2. $lessall'(x, y) \leftarrow \overline{\forall v (v \in y_1 \rightarrow x \leq v)}_{(-)},$
 $\overline{\forall v (v \in y_2 \rightarrow x \leq v)}_{(-)}, union(y_1, y_2, y)$
 $lessall'(x, y) \leftarrow \overline{\forall v (v \in y \rightarrow x \leq v)}$
3. $lessall'(x, y) \leftarrow lessall'(x, y_1), lessall'(x, y_2), union(y_1, y_2, y)$

The complete set of programs and lemmas for *lessall* and *lessall'* are

$$lessall(nil, x)$$

$$lessall(x, nil)$$

$$lessall(u.x, y) \leftarrow lessall'(u, y), lessall(x, y)$$

$$lessall(x, u.y) \leftarrow lessall(x, u.nil), lessall(x, y)$$

$$\text{lessall}(x, y) \leftarrow \text{lessall}(w, z), \text{perm}(w, x), \text{perm}(z, y)$$
$$\text{lessall}(x, y) \leftarrow \text{lessall}(x, z), \text{lessall}(z, y)$$
$$\text{lessall}'(x, \text{nil})$$
$$\text{lessall}'(x, u.y) \leftarrow x \leq u, \text{lessall}'(x, y)$$
$$\text{lessall}'(x, y) \leftarrow \text{lessall}'(w, y), x \leq w$$
$$\text{lessall}'(x, y) \leftarrow \text{ordered}(x.y)$$
$$\text{lessall}'(x, y) \leftarrow \text{ordered}(w.y), x \leq w$$
$$\text{lessall}'(x, y) \leftarrow \text{lessall}'(x, z), \text{perm}(z, y)$$
$$\text{lessall}'(x, y) \leftarrow \text{lessall}'(x, y_1), \text{lessall}(x, y_2), \text{union}(y_1, y_2, y)$$

3.1 Naive Sort

We now have enough material to start the derivation of sorting programs. In fact, we have enough to actually build a naive sort program which is given by

$$\text{sort}(x, y) \leftarrow \text{perm}(x, y), \text{ordered}(y)$$
$$\text{perm}(\text{nil}, \text{nil})$$
$$\text{perm}(x, y) \leftarrow \text{union}(x_1, x_2, x), \text{perm}(x_1, y_1), \text{perm}(x_2, y_2), \text{union}(y_1, y_2, y)$$
$$\text{ordered}(\text{nil})$$
$$\text{ordered}(u.\text{nil})$$
$$\text{ordered}(x) \leftarrow \text{append}(x_1, x_2, x), \text{lessall}(x_1, x_2), \\ \text{ordered}(x_1), \text{ordered}(x_2)$$

together with the procedures already derived for *lessall* and procedures for *union* and *append*.

In the following section, we will derive a program for merge sort, which can be further transformed into a program for insertion sort. Following that we derive a program for quicksort which is further transformed into a program for selection sort. A lemma which can be quite easily proved from the specifications for the *sort* relation is

$$(\text{sort}(x, y) \leftrightarrow \text{perm}(x, y)) \leftarrow \text{ordered}(y)$$

We can also easily derive the following base cases for *sort* from its specifications and from the lemmas already derived.

$$\text{sort}(\text{nil}, \text{nil})$$
$$\text{sort}(u.\text{nil}, u.\text{nil})$$

3.2 Merge Sort

The merge sort derivation starts off with the usual definition of *sort*.

1. $sort(x, y) \leftarrow \underline{perm(x, y)}_{(-)}, ordered(y)$
 $\overline{perm(x, y)} \leftarrow perm(x, z), perm(z, y)$
2. $sort(x, y) \leftarrow \underline{perm(x, z)}_{(-)}, perm(z, y), ordered(y)$
 $\overline{perm(x, y)} \leftarrow union(x_1, x_2, x), perm(x_1, y_1),$
 $perm(x_2, y_2), union(y_1, y_2, y)$
3. $sort(x, y) \leftarrow union(x_1, x_2, x), \underline{perm(x_1, z_1)}_{(-)},$
 $\underline{perm(x_2, z_2)}_{(-)}, union(z_1, z_2, z), perm(z, y), ordered(y)$
 $\overline{perm(x, y)} \leftarrow sort(x, y)$
4. $sort(x, y) \leftarrow union(x_1, x_2, x), sort(x_1, z_1), sort(x_2, z_2),$
 $\underline{union(z_1, z_2, z), perm(z, y), ordered(y)}$
 formation of procedure *merge*
5. $sort(x, y) \leftarrow union(x_1, x_2, x), sort(x_1, z_1), sort(x_2, z_2), merge(z_1, z_2, y)$
6. $merge(z_1, z_2, y) \leftrightarrow \exists z(union(z_1, z_2, z), perm(z, y), ordered(y))$

The derivation of the merge procedure is a little harder since we have to consider more cases.

1. $merge(z_1, z_2, y) \leftarrow \exists z(union(z_1, z_2, z), perm(z, y), ordered(y))$
 deletion of existential quantifier
2. $merge(z_1, z_2, y) \leftarrow \underline{union(z_1, z_2, z)}_{(-)}, perm(z, y), ordered(y)$
 $\overline{union(nil, y, y)}$
3. $merge(nil, y, y) \leftarrow \underline{perm(y, y)}_{(-)}, ordered(y)$
 $\overline{perm(x, x)}$
4. $merge(nil, y, y) \leftarrow ordered(y)$

Similarly, we can derive the Horn clause

$$merge(x, nil, x) \leftarrow ordered(x)$$

We now proceed with the derivation of a procedure for *merge*, assuming that the first pair of terms of *merge* are already sorted, and taking into account the fact that the lists to be merged are not empty

1. $merge(u.z_1, v.z_2, y) \leftarrow \exists z(union(u.z_1, v.z_2, z), perm(z, y), ordered(y))$
 deletion of existential quantifier
2. $merge(u.z_1, v.z_2, y) \leftarrow \underline{union(u.z_1, v.z_2, z)}_{(-)}, perm(z, y), ordered(y)$
 $\overline{union(u.x, y, u.z)} \leftarrow union(x, y, z)$

3. $merge(u.z_1, v.z_2, y) \leftarrow union(z_1, v.z_2, z'), \overline{perm(u.z', y)}_{(-)}, ordered(y)$
 $\overline{perm(u.x, u.y)} \leftarrow perm(x, y)$
4. $merge(u.z_1, v.z_2, u.y') \leftarrow union(z_1, v.z_2, z'), perm(z', y'), \overline{ordered(u.y')}_{(-)}$
 $\overline{ordered(u.x)} \leftarrow ordered(x), lessall'(u, y)$
5. $merge(u.z_1, v.z_2, u.y') \leftarrow union(z_1, v.z_2, z'), perm(z', y'), ordered(y'), \overline{lessall'(u, y')}_{(-)}$
 $\overline{lessall'(u, x)} \leftarrow lessall'(u, z), perm(z, x)$
6. $merge(u.z_1, v.z_2, u.y') \leftarrow union(z_1, v.z_2, z'), \overline{perm(z', y')}, ordered(y')$,
 $\overline{lessall'(u, z), perm(z, y')}$
 $perm(x, y) \leftarrow \overline{perm(x, y), perm(x', y)}$
7. $merge(u.z_1, v.z_2, u.y') \leftarrow union(z_1, v.z_2, z'), perm(z', y'), ordered(y'), \overline{lessall'(u, z')}_{(-)}$
 $\overline{lessall'(x, y)} \leftarrow lessall'(x, y_1), lessall'(x, y_2), union(y_1, y_2, y)$
8. $merge(u.z_1, v.z_2, u.y') \leftarrow union(z_1, v.z_2, z'), perm(z', y'), ordered(y')$,
 $\overline{lessall'(u, y_1), lessall'(u, y_2), union(y_1, y_2, z')}$
 $union(x_1, x_2, y) \leftarrow \overline{union(x_1, x_2, y), union(y_1, y_2, y)}$
9. $merge(u.z_1, v.z_2, u.y') \leftarrow union(z_1, v.z_2, z'), perm(z', y'), ordered(y')$,
 $\overline{lessall'(u, z_1), lessall'(u, v.z_2)}_{(-)}$
 $\overline{lessall'(x, u.y)} \leftarrow x \leq u$
10. $merge(u.z_1, v.z_2, u.y') \leftarrow union(z_1, v.z_2, z')$,
 $perm(z', y'), ordered(y'), lessall'(u, z_1), u \leq v$

Introduction of existential quantifier
11. $merge(u.z_1, v.z_2, u.y') \leftarrow$
 $\overline{\exists z'(union(z_1, v.z_2, z'), perm(z', y'), ordered(y'))}_{(-)}, lessall'(u, z_1), u \leq v$
 $merge(z_1, z_2, y) \leftrightarrow \overline{\exists z(union(z_1, z_2, z))},$
 $\overline{perm(z, y), ordered(y)}$
12. $merge(u.z_1, v.z_2, u.y') \leftarrow merge(z_1, v.z_2, y'), \overline{lessall'(u, z_1)}, u \leq v$
 $\overline{lessall'(u, z_1)}$
13. $merge(u.z_1, v.z_2, u.y') \leftarrow merge(z_1, v.z_2, y'), u \leq v$

The other Horn clause for *merge* can be similarly derived and has the same form.

$$merge(u.z_1, v.z_2, v.y') \leftarrow merge(u.z_1, z_2, y'), lessall'(v, z_2), v \leq u$$

The complete sort program for merge sort is

$$\begin{aligned}
&sort(nil, nil) \\
&sort(u.nil, u.nil) \\
&sort(x, y) \leftarrow union(x_1, x_2, x), sort(x_1, z_1), sort(x_2, z_2), merge(z_1, z_2, y)
\end{aligned}$$

$merge(nil, y, y)$
 $merge(x, nil, x)$
 $merge(u.z_1, v.z_2, u.y) \leftarrow u \leq v, lessall'(u, z_1), merge(z_1, v.z_2, y)$
 $merge(u.z_1, v.z_2, v.y) \leftarrow v \leq u, lessall'(v, z_2), merge(u.z_1, z_2, y)$

3.3 Insertion Sort

The above program for merge sort can be modified a little in order to make it an insertion sort program. In an insertion sort, the first element of the list is removed, the rest of the list is sorted and then the first element is reinserted into the list in the right place without upsetting the ordering.

1. $sort(x, y) \leftarrow \underline{union(x_1, x_2, x)}_{(-)}, sort(x_1, z_1), sort(x_2, z_2), merge(z_1, z_2, y)$
 $\underline{union(v.nil, y, v.y)}$
2. $sort(v.x, y) \leftarrow \underline{sort(v.nil, z_1)}_{(-)}, sort(x, z_2), merge(z_1, z_2, y)$
 $\underline{sort(u.nil, u.nil)}$
3. $sort(v.x, y) \leftarrow sort(x, z_2), \underline{merge(v.nil, z_2, y)}_{(-)}$
 Formation of new procedure *insert*
4. $sort(v.x, y) \leftarrow sort(x, z_2), insert(v, z_2, y)$
5. $insert(v, z, y) \leftrightarrow merge(v.nil, z, y)$

We now derive procedures for *insert*.

1. $insert(v, z, y) \leftrightarrow \underline{merge(v.nil, z, y)}_{(\pm)}$
 $\underline{merge(x, nil, x)}$
2. $insert(v, nil, v.nil)$

This forms the base case procedure for *insert*.

1. $insert(v, z, y) \leftarrow \underline{merge(v.nil, z, y)}_{(-)}$
 $\underline{merge(u.z_1, v.z_2, u.y) \leftarrow u \leq v, merge(z_1, v.z_2, y)}$
2. $insert(v, x.z', v.y') \leftarrow v \leq x, \underline{merge(nil, x.z', y')}_{(-)}$
 $\underline{merge(nil, y, y)}$
3. $insert(v, x.z', v.x.z') \leftarrow v \leq x$

Similarly, we can derive the other Horn clause for *insert*.

$insert(v, x.z, x.z') \leftarrow x \leq v, insert(v, z, z')$

The complete insertion sort program is

$sort(nil, nil)$

$sort(u.nil, u.nil)$
 $sort(v.x, y) \leftarrow sort(x, z), insert(v, z, y)$

$insert(v, nil, v.nil)$
 $insert(v, x.z, v.x.z) \leftarrow v \leq x$
 $insert(v, x.z, x.z') \leftarrow x \leq v, insert(v, z, z')$

Other sorting programs such as, quicksort and selection sort had been synthesized by Varghese. Interested readers may refer to [10].

4 Summary and Discussion

Even after a decade of research on software engineering the productivity still remains a bottleneck. Formal method is one of many approaches proposed to solve the problem. When a program is synthesized with a formal method, the tedious tasks of verification and maintenance become some what trivial. This makes the formal synthesis very attractive.

We have provided a formal framework for deriving logic programs from its specification. The derivational method takes the best aspects of both the transformational and the deductive approaches. The derivation uses the nested resolution which is shown to be sound for the first order logic. Therefore, the derived program is sound, that is, the program is implied by the specifications. On the other hand, since the derived program not always imply the specification, the derivation system is some what weak. Our framework, however, has the advantage that a partial program can always be derived even from a partial specification. Note here that a complete specification is required to derive programs constructively using theorem proving approaches.

In this paper we have derived several sorting programs from the same specification set. Different sorting programs were derived by carefully selecting transformers and transformation rules. As it has been described, automating the derivation is not very practical because of possible combinatorial explosion. This is also true with many automating programming using theorem proving approaches. In our recent work, we have proposed a semi automated approach for deriving logic programs [11].

References

- [1] C. C. Green. A summary of the psi program synthesis system. In *Proceedings of IJCAI-5*, 1977.
- [2] C. C. Green and D.R. Barstow. On program synthesis knowledge. *Artificial Intelligence*, 10(3), 1978.
- [3] R. Loganantharaj and S. Keretho. Lopss: A logic program synthesis system. In *Technical Report, The center for Advanced Computer Studies, USL, Lafayette*, July 1988.

- [4] R. Loganantharaj and J. Varghese. Logic program synthesis. In *AAAI workshop on Automating Software Design: Current Directions*, August 1988.
- [5] Z. Manna and R. Waldinger. A deductive approach to program synthesis. *ACM TOPLAS*, 2(1), 1980.
- [6] Z. Manna and R. Waldinger. The origin of the binary search paradigm. *Science of Computer Programming*, 9, 1987.
- [7] N. Murray. Completely non-clausal theorem proving. *Artificial Intelligence*, (18), 1982.
- [8] J. Traugott. Deductive synthesis of sorting programs. In *Proceedings of the 8th Conference on Automated Deduction*. Springer-Verlag, 1986.
- [9] J. Traugott. Nested resolution. In *Proceedings of the 8th Conference on Automated Deduction*. Springer-Verlag, 1986.
- [10] J. Varghese. *A non clausal method for program derivation*. PhD thesis, Colorado State University, March 1986.
- [11] J. Varghese R. Loganantharaj. Logic program synthesis. In R. W. Wilkerson, editor, *Advances in Logic Programming and Automated Reasoning*. Ablex, 1990.

A STRUCTURE FOR MATURING INTELLIGENT TUTORING
SYSTEM STUDENT MODELS

Dr. Willard M. Holmes
Systems Simulation and Development Directorate
Research, Development, and Engineering Center
U.S. Army Missile Command
Redstone Arsenal, Alabama 35898-5252
Phone 876-1048

ABSTRACT

In this paper, a special structure is examined for evolving a "Detached" model of the user of an intelligent tutoring system. Tutoring is used here in the context of education and training devices. A "Detached" approach to populating the student model data structure is examined in the context of the need for time dependent reasoning about what the student knows about a particular concept in the domain of interest. This approach, to generating a data structure for the student model, allows an inference engine separate from the tutoring strategy determination to be used. This methodology has advantages in environments requiring real-time operation.

INTRODUCTION

During the past decade, a considerable increase in research on Intelligent Tutoring Systems (ITS) has resulted in an expanded body of knowledge about computer based tutoring systems. ITS are sharply contrasted with what is traditionally identified as Computer Aided Instructions (CAI). Early research on the distinguishing characteristics of ITS and differences compared with CAI are reported in a reference text "Intelligent Tutoring Systems" edited by Sleeman and Brown (1982)[7]. Wolf and McDonald (1984)[9] emphasizes the importance of student modeling in developing an effective tutoring system. A general state of technology development in the emerging technology of ITS is reported on by Clancy (1987)[1], Wenger (1987)[8], Kearsley (1987)[5], Polson and Richardson (1988)[6].

In addition to the man-machine interface, the classic model of an ITS includes a teaching module, an expert problem solving module, and a student model. A student model is an essential component of ITS. However, a student model in general cannot be developed entirely independent of the domain in which the model will be used. The vast majority of student model development efforts focus on the "Classic" tutoring problem, i.e., duplicating the tutoring function that takes place in a classroom. Another area of interest in the use of tutoring and student models is related to training and job aids. A mental model for identifying some differences between "Classic" instructional strategies, training, and job aids are reported on by Harman and King (1985)[2].

The use of a student model in context of a specific application domain is described by Holmes (1988)[3] and Holmes and Chamberlain (1988)[4]. For purposes here, the student model is defined as that component of an ITS that collects student model performance information to be used to make inferences

about what the student knows and does not know about a particular concept or required training task. Before the student model can be used to draw conclusions about the state of knowledge possessed by the student, the model must first be initialized or have results available from previous tutoring operations. More precisely, the student model must have a specified structure and a defined process for populating the structure. With the populated structure, inferences can be made about the knowledge state of the student.

By using an ITS in the context of a Training System (TS), it can be identified as an Intelligent Tutoring-Training System (ITTS). The task to be presented to the student in an ITTS application is similar in nature to the basic principles that would be used for an ITS for the "Classic" knowledge tutoring problem associated with classroom settings. In the ITTS operation, simulation systems are frequently included to support the exercise of both knowledge and skills in the tutoring operation.

HIERARCHICAL DECOMPOSITION OF TUTORING TASK

Given that a main level concept with specific performance objectives has been identified as an element to be used in a tutoring operation. A series of steps must be completed before the tutoring function can be implemented. An initial step is to perform hierarchical decomposition of the main concept into subconcepts. The decomposition continues until desired fidelity level is acquired. The fidelity of the decomposed task is related to the number of levels in the hierarchy. The lowest level subtask in the hierarchy is defined here as the component subtask. Skills and knowledge associated with the component subtask is identified as the primitives of the task. This decomposition process is typically an element that is part of a total task analysis effort. In conjunction with the task analysis is the skill analysis to identify requisite skill associated with the task elements. A conceptual model for identifying the task analysis components is shown in Figure 1.

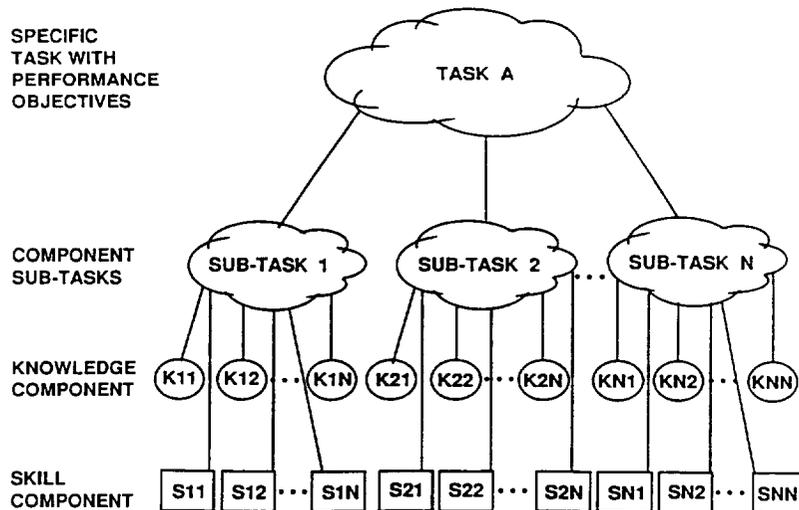


Figure 1. Conceptual model for identifying task analysis components.

The hierarchy of skill components indicated in Figure 1 is a simplification of the many subtask levels that can be associated with a task. A major point here is to focus on the desirability of having associated with each subtask a concept that is an element in the major task. The knowledge and skill components associated with the subtask is the knowledge necessary to master the concept and the skill to demonstrate the operation associated with the subtask. Knowledge is defined here to include what the student or operator needs to know (principles, concepts, facts, etc.) about the subtask to be accomplished. A skill is defined as having the requisite knowledge and the ability to apply that knowledge effectively.

SKILL BASED TASK ANALYSIS

An example emphasizing the use of information at the component subtask level with associated knowledge component and skill component is instructive. Consider this over simplified example of teaching a student how to fly a small airplane as indicated in the following table.

The observed performance with resulting conclusion and recommendation is typical of that made by a human tutor. The knowledge component is information the student can obtain in classroom sessions, books, and discussion with an experienced pilot. The measure of the student's knowledge can be a series of questions. The skill component is developed and tested either with a training device or the actual airplane. The performance measure can be an observation and measurement of action taken in response to a given stimuli. As can be observed, the same knowledge and skill component can appear in more than one subtask, i.e., adjust controls in ST.-1 and ST.-2. This is the same skill but the student must have the knowledge of the context of using that skill, i.e., in landing or stall situation. Hence, the ability to use the knowledge correctly.

The expert, in expressing his rule-of-thumb, may use terms and expressions not used at the knowledge component or the skill component level. However, the student's action can be compared with the expert's at the subtask level. Tutoring operation to improve deficiencies can use student performance at the subtask level and the related information at the knowledge component and skill component level.

STRUCTURE FOR STUDENT MODEL DATA GENERATION

For purposes of prototype model development, the structure shown in Figure 2 will be used to identify a procedure for developing student model data. As indicated, the basic structure includes a main concept, one or more sun-concepts, and primitive element. The primitive elements have context sensitive Primary (P) and Alternate (A) question associated with each primitive.

TABLE. Learn to fly a small airplane.

<u>COMPONENT SUBTASKS:</u>		
<u>SUBTASK.1</u>	<u>SUBTASK.2</u>	<u>SUBTASK.3</u>
ST.1 Learn how to land	ST.2 Learn how to take off	ST.3 Learn how to handle stalls
<u>KNOWLEDGE COMPONENTS</u>		
Head winds Cross winds Flight path angle Landing speed	Plane load Air temperature Take off speed Aerodynamic lift	Get nose down Increase speed Sufficient speed Control settings
<u>SKILL COMPONENTS</u>		
Adjust controls Observe airport wind indicator Compare approach Angle with horizon Adjust air speed	Set throttle Observe cross wind indicator Changes controls with correct air speed	Adjust controls Observe air speed Level off
<u>OBSERVED MEASURE OF PERFORMANCE FOR SUBTASK.1</u>		
Approach angle too high, hard landing Landing speed too high, ran off runway		
<u>CONCLUSION</u>		
Student and plane survived, but the student need more practice.		
<u>RECOMMENDATIONS</u>		
The student engage in extended practice session of touch-and-go landings with emphasis on control surface adjustment and speed adjustments.		

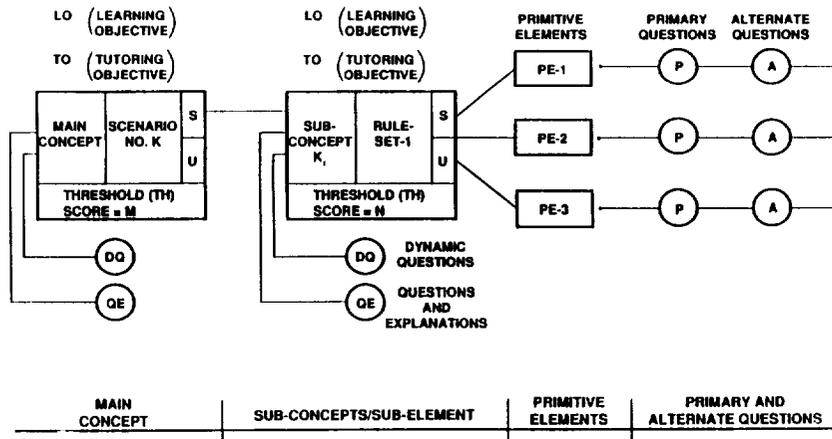


Figure 2. Basic structure for developing student model information.

The main concept is considered to be a particular scenario operation. A Learning Objective (LO) is identified for the main concept. The LO will be dependent on concepts included in the scenario. The Tutoring Objective (TO) will be dependent on the particular student being tutored, i.e., student model information and capabilities required to solve the scenario. As used here, a TO is related to the process of establishing the number of chunks of knowledge to be presented to the student. In turn, a LO is related to the process of testing to determine if sufficient knowledge has been mastered. Obtaining student model information at the main concept level may include presenting the student with a set of Dynamic Questions (DQ) involving both knowledge and skill or presenting static Question with Explanation (QE) as required.

The student may not be required to master 100 percent of the knowledge and concepts contained in this main event (scenario) Rule Set (RS) before advancing to the next main concept (scenario). A Threshold (TH) of performance is established for the main concept. An indication that the LO has been Satisfied (S) or Unsatisfied (U) is indicated by setting the appropriate performance indicator. The TO can be satisfied even if the LO was not satisfied. Results of the student's performance at the main concept level is added to the student model knowledge base.

Entering into a particular subconcept operation can be accomplished by one of three approaches. The learning objectives associated with the main concept was not satisfied and the tutoring strategy directed that a subconcept of the main concept be explored, (top down approach). Second, the tutoring strategy and student model contents indicate that the bottom line component subconcept be explored before advancing to a higher level concept (bottoms up approach). The bottom line subconcept level is defined as the level directly connected to the primitive elements.

The third approach to subconcept operation is associated with paths that includes several subconcept levels between the main concept and the primitive level. Under these conditions, the tutoring strategy can require that operations proceed to a particular subconcept to satisfy certain TO (arbitrary approach). This approach would be applicable to an expansion of the basic structure as shown in Figure 2. The subconcept levels directly associated with the primitive elements are identified as the subelement level.

The student model contains results of the student performance at all subtask levels, including the component subtask level. Any interaction with the primitive elements during a TO is not recorded in the student model. This requirement is tied to the fact that knowledge about the student's performance consist of two parts: the student's performance during Past Tutoring Efforts (PTE); and the students performance in the Present Tutoring Operation (PTO). The student's performance is not considered to be a past performance until the PTO for a subtask is satisfied.

Consider the option that while achieving a TO it is necessary to enter the subelement level of operation. Also, consider while satisfying the LO of a particular subelement it is necessary to interact with the associated primitive elements. The P question and A questions are used to inform the student about the characteristics of the primitive elements. Primitive element level interaction with the student continues until the threshold level of that particular subelement is achieved. When the performance threshold level of the subelement is achieved, the results at the subelement level is recorded in the student model. This gives information on the student's performance in relation to particular LO associated with the subelement and not in relation to the primitive elements. At this point, the results are based on past performance since the LO has been achieved. The interaction with the primitive elements occurred before the TO was achieved, i.e., in a present scenario operation mode.

This requirement, that the threshold level of the subelement be satisfied before continuing the TO with higher level concepts, can be used to establish lower bounds on the tutoring and training operation. An ideal structure would have a small number of primitive elements associated with each subelement, i.e., three. If the threshold performance level is set at 100 percent and the student cannot achieve the LO of the component subelement, then the student should receive training of a more fundamental nature than the particular ITTS can provide.

However, other options exist if the performance threshold level of the subelements are set at some value other than 100 percent. No record is kept of the order the PEs are presented to the student during the first tutoring session using the particular subelement. Since a flag was set indicating results of the last LO, at least one pass has been made using the subelement. In the event a second pass is required to meet a TO, the order of presenting the PEs on a second pass through can be reversed to make it interesting for the student. Shown in Figure 3 is a conceptual model of an expanded structure for student model data development.

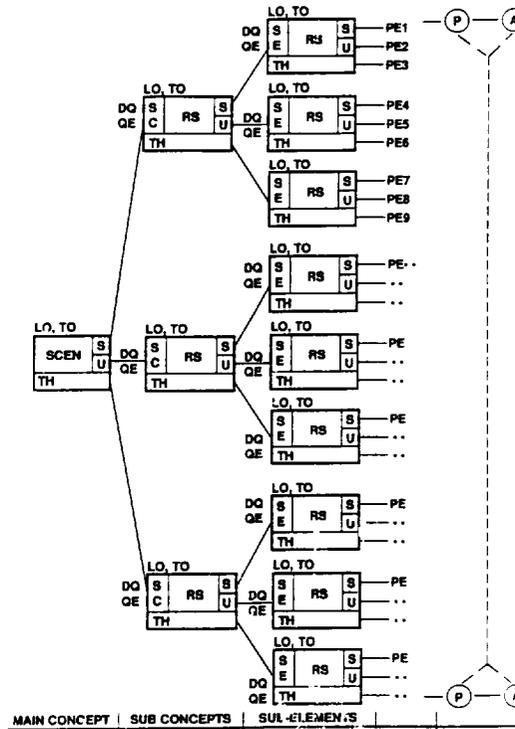


Figure 3. Expanded structure for student model development.

CONCLUSION

An initial implementation on the prototype structure, as described here, is effective in providing required information to a student model for a special application of tutoring systems. The implementation of the structure, described here, involved an initial prototype. The next step is to investigate the effectiveness of applying the process to larger tasks with an increased number of subtasks.

ORIGINAL PAGE IS
OF POOR QUALITY

REFERENCES

1. Clancy, William J., Knowledge-Based Tutoring: The GUIDON Program, The MIT Press, Cambridge, MA, (1987).
2. Harmon, Paul, and King, David, Artificial Intelligence in Business, John Wiley and Son, Inc., New York, NY, (1985).
3. Holmes, Willard M., and Chamberlain, David, "A Scenario Generator for an Intelligent Tutoring Device used in Maintaining Operator Skill Levels," Proceedings of the 1988 Summer Simulation Conference, Seattle, WA, pp. 613-619, July 1988.
4. Holmes, Willard M., "A Structure for Developing a Domain Specific Intelligent Tutoring System for Maintaining Proficient Skill Levels for Weapon System Operation," 2nd Annual Conference on Advancement in Training Technology, National Science Center for Communications and Electronics, Augusta, GA, 26-18 April 1988.
5. Kearsley, Greg, Artificial Intelligence and Instructions, Addison-Wesley Publishing Company, Menlo Park, CA (1987).
6. Polson, Martha C., and Richardson, Jeffrey J., (Eds.) Foundations of Intelligent Tutoring Systems, Lawrence Erlbaum Associates, Inc., (1988).
7. Sleeman, D., and Drown, J. S., (Eds.) Intelligent Tutoring Systems, Academic Press, Inc., New York, NY, (1982).
8. Wenger, Etienne, Artificial Intelligence and Tutoring Systems, Morgan Kaufman Publishers, Inc., 95 First Street, Los Alto, CA, (1987).
9. Wolf, Beverly, and McDonald, David D., "Building a Computer Tutor: Design Issues," Computer, IEEE Publication, pp. 67-73, September 1984.

A Tool for Modeling Concurrent Real-Time Computation

D.D. Sharma, Shie-rei Huang, Rahul Bhatt, N.S. Sridharan

*FMC Corp.
Knowledge Systems Research
Artificial Intelligence Center
Central Engineering Laboratory
1205 Coleman Ave., Box 580,
Santa Clara, CA 95052*

Review Area: Real-Time Performance

ABSTRACT

Real-time computation is a significant area of research in general, and in AI in particular. The complexity of practical real-time problems demands use of knowledge-based problem solving techniques while satisfying real-time performance constraints. Since the demands of a complex real-time problem cannot be predicted (owing to the dynamic nature of the environment) we need powerful dynamic resource control techniques to monitor and control the performance. In this paper we briefly describe a real-time computation model for a real-time tool, an implementation of the QP-Net simulator on a Symbolics machine, and an implementation on a Butterfly multiprocessor machine.

1. Introduction

Real-time computations is a significant area of research in general, and in AI in particular [1]. The tradition work on real-time has focused on the execution performance of programs and has lead to development of real-time operating systems where the emphasis is on optimizing execution speed of various parts of the operating system kernel. The complexity of practical real-time problems demand use of knowledge-based problem solving techniques while satisfying real-time performance constraints. The complexity of the problem requires that apart from the program execution speed we should also address other performance metrics such as responsiveness, timeliness, and graceful degradation [2]. Thus in addition to the current tools of real-time operating systems we need support for knowledge-based problem solving and user-controllable dynamic resource control. In this paper we describe the architecture and design of such a real-time tool.

Our real-time tool is based on a computation model (called QP-Net) to help the application developer in modeling the real-time problem solution [2]. The computational model supports both the parallel and sequential execution of tasks. The parallel computation model supports both the static and dynamic scheduling of processors to tasks. We have further refined the QP-Net model to provide features important to real-time problems such as asynchronous task execution, performance monitoring, and dynamic resource computation, and dynamic resource scheduling. In this paper we briefly describe QP-Net, the multilayered architecture for the real-time tool, a simple example, an implementation of the QP-Net simulator on a Symbolics machine, and an implementation on a Butterfly multiprocessor machine.

2. QP-Net

QP-Net is a computational model for exploiting concurrency inherent in an application. By exploiting concurrency we mean making concurrent aspects of application solution explicit and providing mechanisms to realize concurrent behavior. The objective of QP-Net model is provide a user with conceptual tools to express concurrent solutions in the ways that matches best with his/her understanding of the problem solution. The motivation being that such an approach will lead to "distributed AI solution architectures" and facilitate both systematic development of concurrent solutions and also help understand "distributed intelligent" architectures in an empirical manner.

Given an application problem the QP-Net based solution consists of three parts:

1. The Solution Model which defines asynchronous application tasks and their interrelationships in terms of a directed communication network.
2. The Execution Model which specifies the temporal order in which tasks are to be executed.
3. The Resource Model which specifies scheduling of finite resources statically or dynamically among the application tasks.

The *solution model* defines the logical execution ordering of the tasks. The interaction/communication between the tasks is explicitly defined by directed network links. The network allows the user in developing conceptual solution models by using network links to indicate "enables" and "restrictions" of interactions between the tasks.

The *execution model* consists of two parts. First, the execution of a task is triggered as a result of execution of a predecessor task defined by the network. A task upon execution passes an "execution token" to the tasks at the other end of the link. The execution of tasks is basically asynchronous which is achieved by associating an "input-buffer" with each task containing arriving execution tokens. The second part of the execution model consists of scheduling task instances. Various prioritization policies can be used to select a task token from the "input-buffer" for execution.

The *resource model* consists of two parts. First, resources can be assigned to tasks in some predefined manner. For example, a part of resources may be *dedicated* to a set of critical tasks and rest of the resources can be shared dynamically by the remaining tasks. The second part of resource allocation model consists of dynamically reallocating the resources.

The QP-Net based application solution model is "richer" than traditional concurrent models. Unlike CSP and Petri-Nets, QP-Net allows asynchronous execution of tasks while allowing synchronous execution as a special behavior. Unlike concurrent object-oriented models such as Actors [3], and Mace [4], QP-Net makes communication explicit and allows dynamic computation of priorities. Unlike any of the existing models, QP-Net allows dynamic reallocation of resources.

3. QP-Net and Real-Time Applications

The QP-Net approach specially addresses the needs of real-time problems by supporting the following:

- A task queue based solution model which supports asynchronous execution of logically parallel tasks.
- User controllable execution control defined based on application specific knowledge and heuristics in designing the network and task priority and scheduling policies.
- User tune-able performance driven resource re-allocation scheme.
- Development of knowledge-based procedures realized by building RT-Tool on an object-oriented platform.

3.1. Task-Queue Based Solution Model

First, a QP-Net solution model essentially consists of a network of logically parallel tasks. One can view the network connecting an arbitrary number of producer and

consumer tasks. Normally these tasks can be executed asynchronously. A producer or a consumer task can in principle be any arbitrary process as long as it can interact with other processes in two respects: It can receive a job to be done from the predecessor processes in the network and it can send a job to the successor processes in the network. The feature of asynchronous execution makes it possible to model real-time problems because it enables different processes to execute at different speeds which is necessary to satisfy certain real-time requirements.

The QP-Net approach also defines a specific type of model for each of the producer/consumer node in the network. Each node consists of task queue which contains "prioritized tasks" and a server to execute these tasks. The task queues receive tasks from predecessor nodes and enqueue them using certain enqueueing discipline. Thus task queues serve both as buffers (thus enabling asynchronous execution) and also as agendas for scheduled tasks. In QP-Net the task queue and server operations are also considered asynchronous. Thus more than processes of task-queue and server can be active simultaneously. The asynchronous operation enables performing task-queue operations to be responsive to environment demands and server operations to be responsive to task-queue demands.

3.2. Execution Control

In real-time applications a solution is required to be responsive to the data or jobs imposed by the environment. The demands of an environment cannot be directly controlled and in some situations it cannot be anticipated. Given a stream of data or jobs it is necessary to determine the priority in which these tasks should be done. QP-Net model explicitly supports used defined priority mechanisms and thus enables incorporating scheduling algorithms to select real-time tasks for execution.

3.3. Resource Allocation

Real systems have finite resources. Given the criticality of certain tasks it is necessary to allocate certain resources as dedicated and others as shared. As discussed in previous section QP-Net model supports both static and dynamic resource allocation and does specifically to control real-time performance.

3.4. Knowledge-based Procedures

Problem solving in real-time application is knowledge-based. Not only are the domain tasks knowledge-based but the control tasks required for execution control and resource allocation can also be knowledge-based. Nothing in QP-Net model precludes knowledge-based approach. The tool described in the next section facilitates building knowledge-based procedures by providing object-oriented support.

4. Prototyping QP-Net Model

4.1. QP-Net Model Primitives

The QP-Net model for a problem is defined as a network of the following primitive elements:

1. **Tasks:** A Task is a description of an user-defined activity to be performed and may involve local data or other behavior. It involves finite and bounded computation. Tasks are independent, can be executed asynchronously, and any dependence is modeled in terms of network. Execution of a task can have one of the following two effects.: changes to the local state of the task or changes posted to out-going network links.
2. **Task Queues:** It is a queue of tasks and serves two purposes: as a buffer to enable asynchronous execution of tasks, and to enable scheduling of tasks for execution. Four types of primitive task queues are recognized: fifo, priority, time-constrained, and synchronized.
3. **Q-Managers:** A Q-Manager manages task inflow and outflow for a set of priority task queues. Normally, task queues will be elements of a q-manager.

4. **Servers:** A Server attends to a specified q-manager. When free, it requests and receives a task from a q-manager, executes it, and goes back to the q-manager for another task. A server is not a physical processor but a process. Thus the behavior of a server can be defined by the user. Also the design of an application solution in terms of q-managers and servers can be independent of the number of processors.
5. **Task Flow Links:** Task flow links connect a q-manager to a server. The links are directed. The links can be best thought of as a pipe . Normally a server can have only one input link but several output links. In some designs (specially designs for synchronized behavior) more than one input links are possible.

4.2. High-Level Components in QP-Net Model

In this section we describe the operations permissible on a QP-Net element and operations to be performed by a QP-Net model. This will specify the semantics of the model elements.

4.2.1. Tasks

Tasks specify the activity to be performed. There are system tasks and application tasks. Here we will focus on application tasks.

Application tasks are typically generated by a server. Upon generation the execution of tasks is controlled by various factors such as its priority. The activity to be performed is expressed as the body of the task. A task has the following attributes:

1. **Trigger-Condition:** These are the conditions which should be true for the task to be generated.
2. **Precondition:** A task may not be ready for execution soon after its generation. Prior to its execution various resources should be available, conditions should exist that do not hinder the execution of task, and conditions for initiating the task execution should be true. The collection of all these conditions are defined as **Precondition**.
3. **Body:** The body of the task defines the activity to be performed.
4. **Priority:** The priority of the task is determined in relation to its need to have resources made available to it. It is a dynamically computable quantity.

Real-Time Task: Tasks to which the execution time is of critical importance are called **real-time tasks**. A real-time task has a timer associated with it. The timer is created at the time of task creation. At any time after that the timer can be queried to find the age of the task.

4.2.2. Task Queues

A task queue is a queue of application tasks. The following operations can be performed on a task queue:

1. **Create/Delete a task queue**
2. **Lock/Unlock a task queue**
3. **Enqueue/Dequeue a task on a task queue**

The following four types of primitive task queues are currently supported:

1. **FIFO Task Queue**
2. **Priority Task Queue:** A task is enqueued/inserted on the task queue at a location consistent with its priority. Dequeue of the task queue removes the highest priority task.
3. **Time-Constrained Task Queue:** A task is enqueued/inserted on the task queue based on its deadline. The task with the tightest deadline is at the head of the task queue.
4. **Synchronized task queue:** Synchronized task queue dequeues the task only when specified synchronization conditions are met.

4.2.3. Q-Managers

A Q-Manager controls the task inflow and outflow of its tasks queues. It may consist of a number of task queues. A Q-Manager has a unique name. Normally, a Q-Manager will be associated with specific types of tasks such as: update sensor data; execute plans for the operation of system, etc. A Q-Manager has procedures to perform the following:

1. Assign priority to a new task
2. Enqueue the task to appropriate task queue
3. Schedule a task for execution
4. Measure specific performance parameters such as the task queue length, time to deadline, the rate at which tasks have been arriving, and the rate at which tasks have been depleting.
5. Estimate the service needed.
6. Update the priorities of the tasks.
7. Change the queue size (i.e., in essence memory) dynamically to either free-up resources or produce more dynamically stable behavior.

An external object can interface with a Q-Manager in the following ways:

1. Send a new task
2. Request a new task

Priority: One of the features of Q-Manager is to estimate the priority of a task. The priority of a task is intended to be a dynamic quantity rather than being a static quantity assigned at the time of the creation of the task. Various factors influencing the priority are:

Time-Stress or the time between now and the deadline to begin or finish the task.

FIFO: A simple default prioritization

Relative Priority: Since priority drives resource reallocation it is estimated relative to the priorities of existing tasks.

4.2.4. Servers

Servers are the logical processors in QP-Net. Servers are logical processes to keep the program independent of assumptions about any particular processor allocation strategy. This is done to:

1. keep the logical structure of the users program simple;
2. provide maximum flexibility for choosing and experimenting with various processor allocation strategy.

A server requests Q-Manager for a task and then execute the task. A server has the following attributes:

1. A unique name
2. Name of the Q-Manager served.
3. A specification of what the server is supposed to do, i.e., a procedure.
4. Name of Q-Managers or output pipes and rules for determining where the results of executing of a task be enqueued.

4.3. Performance Metering

Performance metering can provide insights into the behavior of a program and help identify program features which can be further optimized. Performance measurement/analysis has been an important area in the design and analysis of operating systems. The analysis usually yields results such as the utilization of processors, the computer system throughput, task queue lengths, and response times. These results can then be used to evaluate or modify computer system designs. In QP-Net we are interested in measuring similar characteristics but our objective is to not simply

to collect benchmarking data but to control the performance characteristics of its application. This leads to both a problem and an opportunity. The problem is that much of the computer systems benchmarking techniques are not applicable. The opportunity exists because our measurements can be heuristic and less precise, the lack of precision is overcome by using a feedback control mechanism. We thus choose the heuristic approaches over formal performance modeling approaches.

In QP-Net the primary objective of performance measurement is to collect data about the programs execution which can be used for on-line control the performance of the program with respect to some predefined standards. Thus the performance metering is used both: (1) To provide comparative data to analyze the performance of various programs with respect to stated specifications, and more importantly (2) to provide data to be used in adjusting the resources to achieve the specifications. The following parameters are currently measured:

1. Latency: Average time lapsed between the the generation of a task and the beginning of its execution. If the current average latency is L , the latency of a new task is l , then new average latency, L_{new} can be given by:

$$L_{new} = f \times l + (1 - f) \times L,$$

where f is a weighting factor with value between 0 and 1. Similar formula is used in the following to compute the average.

2. Task Queue Size: The number of tasks waiting in the task queue for execution.
3. Granularity: The average size of the task, which is measured by elapsed execution time.
4. Inter-service Interval: Average time between two services at a queue.

The problem of how we take these parameter values and assess the performance status of the system is discussed in the section of Performance Control Strategy.

4.4. Resource Allocation

In QP-Net resource allocation essentially refers to assigning processors to the QP-Net based solution. Let us recall that a QP-Net model consists of network of task queues and servers. A pair of task queue (or q-manager) and server is called a QP-Site. Collection of QP-Sites are called Clusters. Processors can be allocated in the following manner:

1. Allocate processors to q-managers and servers.
2. Allocate processors to each site which in turn reallocates them to q-managers and servers.
3. Allocated processors to clusters which in turn can reallocate them to qp-sites and qp-sites can reallocate them to q-managers and servers.
4. Allocate processors to groups of logically parallel activities, called a module. A module can typically contain several q-managers and servers.

For generality, we will say that processors are allocated to modules where a module can be a cluster, a qp-site, q-manager, server, or an arbitrary composition of activities.

4.4.1. Specifying a Module

Given an application solution as a QP-Net work the user can define execution modules. Processors are allocated to modules. A module can be defined in the following manner:

(define-module M_i (Q1 S1 Q2))

where Q1, Q2 are q-managers and S1 is a server.

4.4.2. Schemes for Assigning Processor to Modules

The processor assignment can be either dedicated in which case a fixed number of processor is allocated to a module, or flexible where the processor can serve one of many groups. Given a set of modules, (M_1, M_2, M_3, M_4), how do we assign

processors to the modules? The processor allocation is specified by (Module-ID, Processor-ID). If all processors are identical and any module can be executed on any processor then the allocation can be specified as: (Module-ID, Processor Set) where any processor in the processor set can be used.

```
(define-assignment assign-1 (modules '(M1 M2 M3)
                                   number-of-processors 4
                                   allocation-strategy uniform-random))
```

```
(define-assignment assign-2 (modules '(M4)
                                   number-of-processors 2
                                   allocation-strategy dedicated))
```

4.4.3. Random Allocation Strategy

The dedicated allocation strategy is simple. Here we describe a random shared allocation strategy.

1. With each module M_i , associate a parameter d_i (called demand parameter). Parameters d_i can change values dynamically.
2. Let $D = d_1 + d_2 + \dots + d_n$. Out of a total of D executions allocate d_i to M_i . One scheme to allocate is to generate a random number, r , in range $(0, D - 1)$. If $\sum_{j=1}^{i-1} d_j \leq r < \sum_{j=1}^i d_j$, then assign the processor to M_i .

4.5. Performance Control Strategy

The objective of performance control strategy is to compute demand factors d_i for each module to which processors are to be allocated. The need to dynamically compute d_i arises because we want to maintain certain level of real-time performance. For example, suppose we want to keep the latency of tasks in a specific task queue to be less than some specified value l_{ref} . Any given allocation of processors to this task queue will be adequate to assure the desired latency as long as the task arrival rates do not increase or the service rates do not decrease. In real-time situations both of these things can happen. The task arrival rate depends upon the external environment and can change dynamically. The service rate in a finite resource system depends can change if some other module requires more than its allocated service. We need two types of control:

1. the capability to increase processor allocation to task queue if the demand exceed the current capabilities, and
2. the capability to decrease processor allocation if the demand decreases.

The current approach is to associate a demand factor d with each module. If more resources are needed then d is increased. If less resources are needed then d is decreased. *The specific problem thus is to determine when should we change d and how?*

Our approach to determining d is to formulate the problem as a feedback control problem.

Simple Heuristic Control Schemes

First we discuss simple control schemes based on intuitions of how feedback control systems behave and their adaptation to the problem of performance control. In defining these schemes we are assuming a central scheduler, as discussed in the previous section.

1. Latency Control

Let l_{ref} be the desired value of latency, l be the predicted value of latency, d be the current demand factor, int be the current inter-service time, q be the number of task in the queue waiting for execution, and Δd be the desired change in d . l

can be estimated as:

$$l = q \times int.$$

Assuming a linear relationship between d and l , we have the following rule:

$$\Delta d = \frac{(l - l_{ref}) \times d}{l_{ref}}.$$

It can be shown that the above heuristic corresponds to controlling a constant task-queue size and the service rate is proportional to demand factor.

The above scheme gives a continuous control of latency, i.e., every time a task is added or removed the demand factor d is computed. In order to limit the overhead of computing d and rescheduling resources we can use a "threshold monitoring scheme" described below.

Let l_1 be the upper limit on latency and l_2 be the lower limit on latency. If l is in range (l_1, l_2) , then do nothing; else recompute d . This is a simple form of non-linear control.

2. Controlling Queue Size

Let q_{ref} be the desired queue size, d be the current demand factor, and q the current queue size; then,

if $0.8q_{ref} < q < 1.2q_{ref}$
do nothing;
else

$$\Delta d = \frac{q - q_{ref}}{q_{ref}}.$$

3. Deadline Control

Let us consider a time-constrained task queue. The head of the queue is a real-time task which has the tightest deadline. Let t_d be the deadline for the task to be completed, t be the current time, g be the average granularity of the tasks in this queue, and t_a is a pre-determined constant denoting the allowance time for scheduling task execution.

If $t_d > t + int + g + t_a$, do not schedule the task; else remove the task and execute it. After the execution, if there is a new task in the head of the queue, new demand factor, d_{new} is computed below. Let $k = \frac{t_d - t - g - t_a}{int}$. If $k \leq 0$, the new task should be executed immediately by the same processor; otherwise set $d_{new} = \frac{d}{k}$

5. Implementations

A simulator has been implemented on Symbolics under Genera 7.2 and a prototype is running on a BBN Butterfly Multiprocessor.

5.1. QP-Net Simulator

The advantage of simulation is to be able to:

1. Simulate different architectures.
2. Provide debugging environment.
3. Observe the execution cycle of different processors.
4. Independent Metering of parameters.
5. Observe operation at different rate of execution.

The simulator was designed with the idea that this would enable the user to quickly evaluate different models. This evaluation requires that the system be easily modifiable

and observable. Together with this, the system should have a reasonable speed of operation. Accuracy of the results could be traded for speed. Hence the simulator does not simulate all the detailed communication and resource parameters. Instead, what was provided is a means to give the average values for each of these parameters. Hence the net effect of these parameters is taken into account. Also the simulation provides an environment to easily debug the application under consideration. Most of the multiprocessor environments lack this capability. Using a simulator makes the task of metering independent of the real time clock. Also monitoring of the different metering parameters is made easy because of having the control of the simulation dedicated to a single processor.

5.2. Butterfly Based Implementation

A prototype of QP-Net has been designed and implemented on the BBN Butterfly Multiprocessor using Butterfly Scheme. The prototype contains four main tool components: queue constructs, performance monitor, resource controller, and dynamic resource allocator. A simple object-oriented programming shell is first implemented on the top of the Butterfly Scheme to facilitate the object-oriented design of the QP-Net.

Basic queue constructs are implemented as object classes so that programmers can inherit them to develop application programs. Four kinds of basic queue constructs have been identified and implemented; they are: the FIFO queue, the priority queue, the time-constrained-queue, and the synchronized queue. Queues are supposed to be filled with "real-time tasks". By real-time task we mean a task that has a private timer associated with it. The timer is an object instance which can be read, reset, stopped and set to run. When a real-time task is created, the timer associated with it is reset and run. A task can be very light-weighted. It may contain only minimal information that the server needs to carry out certain actions.

Real-time tasks cannot be all independent. Especially when tasks are generated by breaking up big tasks, certain order of execution must be followed by the small tasks for correct operations. The synchronized queue construct can be used to synchronize tasks in QP-Net. Tasks are synchronized by producing a tagged data each to a synchronized queue. A synchronized queue accepts two or more tagged data each from a different server. The synchronized queue then only send a complete group of data with the same tag to the server. The synchronization mechanism in a synchronized queue is similar to but more powerful than the data-driven synchronization in Petri Net because data available must have the same tag in order to fire a task in a synchronized queue.

The algorithms used for the implementation of resource controller and the dynamic resource allocator were discussed in Section 7.3 and Section 7.4. The prototype has demonstrated some desired features of a real-time system. More experimental results will be documented in the near future.

6. Conclusion

In this paper we have described the design of a real-time tool based on a new computation model (QP-Net). The computation model's features have been specifically customized to address issues relevant to real-time problems. The real-time tool is currently implemented on a Symbolics Lisp machine and a 16-node Butterfly Multiprocessor.

Based on our work so far we have identified several interesting research issues. We find that real-time applications are highly knowledge rich and complex issues of deadline scheduling and resource allocation can be simplified by appropriate task modeling. For example, we have defined a real-time task whose priority is dynamically computed. Our approach also enables treating time both as a resource and a constraint. We find that modeling a problem solution in terms of logically parallel tasks enables maximum advantage of application inherent concurrency. However, tasks need to be coalesced to control the overhead of resource reallocation. In real-time problems it is also required that task merging should be done based on cost effectiveness of

reduced overhead of resource reallocation (by increasing task size) and the loss of responsiveness (because processors will be busy for longer per task).

We have identified a few issues at the level of operating systems where further work can be beneficial. If the system architecture is distributed then a major problem is to improve processor utilization. This calls for monitoring processor state and then migrating objects accordingly. Another possible approach is to distribute an object over the processors and provide schemes for redirecting messages to the appropriate processor. Apart from the issue of processor utilization we also need to worry about the communication overhead. Again the ongoing research in this area needs to take into account the constraint of responsiveness.

If the system architecture is based on distributed memory then processor utilization issue is solved by using dynamic scheduling of processors. An interesting problem arises because of the overhead of task allocation from a single task queue to multiple processors. The speed-up of task dequeuing is limited by the saturation behavior. Thus if we are operating near the saturation point then allocation of additional processors will not lead to desired performance characteristics. The allocation scheme will have to be sensitive to this phenomenon. Or even, better the task queue data structure may be split into two and at the OS level the processor scheduling issue be handled.

References

- [1] Stankovic, J. A., *Misconceptions about real-time computing: a serious problem for next-generation systems*, Computer, Vol 21, No. 10, Oct. 1988, pp. 10-19.
- [2] Sharma D. D. and Sridharan, N. S., *Knowledge-based real-time control: a parallel processing perspective*, in AAAI-88, Saint Paul, Minnesota, Aug. 1988, pp. 665-670.
- [3] Hewitt, C., *Viewing control structures as patterns of passing messages*, Artificial Intelligence, 1977, pp. 323-364.
- [4] Gasser, L., Braganza, C. and Herman, N., *Mace: a flexible testbed for distributed AI research*, in M. Huhns ed., *Distributed Artificial Intelligence*, Morgan Kaufmann Publishers, Los Altos, CA, pp. 119-152.

**Construction of Dynamic Stochastic Simulation Models
Using Knowledge-Based Techniques**

**M. Douglas Williams
Advanced Technology, Inc.
555 Sparkman Dr., Suite 454
Huntsville, AL 35816**

**Sajjan G. Shiva
Computer Science Department
University of Alabama in Huntsville
Huntsville, AL 35899**

ABSTRACT

Over the past three decades, computer-based simulation models have proven themselves to be cost-effective alternatives to the more structured deterministic methods of systems analysis. During this time, many techniques, tools and languages for constructing computer-based simulation models have been developed. More recently, advanced in knowledge-based system technology have led many researchers to note the similarities between knowledge-based programming and simulation technologies and to investigate the potential application of knowledge-based programming techniques to simulation modeling.

This paper discusses the integration of conventional simulation techniques with knowledge-based programming techniques to provide a development environment for constructing knowledge-based simulation models. A comparison of the techniques used in the construction of dynamic stochastic simulation models and those used in the construction of knowledge-based systems provides the requirements for the environment. This leads to the design and implementation of a knowledge-based simulation development environment.

These techniques have been used in the construction of several knowledge-based simulation models including the Advanced Launch System Model (ALSYM).

1.0 INTRODUCTION

Knowledge-based simulation extends the set of tools available to the simulation modeler by incorporating techniques from the field of artificial intelligence. [Smith et al 1988] In the context of this research, simulation techniques are limited to discrete-event simulation methods for constructing dynamic stochastic simulation models. The techniques from

artificial intelligence that have proved most useful are broadly classified as knowledge-based programming techniques, hence the name knowledge-based simulation. There are many similarities between knowledge-based programming techniques and conventional discrete-event simulation techniques, as well as some important differences.

The most important similarity is the separation of the domain problem-solving knowledge from the control strategy applying this knowledge to solve some problem instance in the domain. In knowledge-based programming, this knowledge is represented in the form of rules or logical statements. The control strategy is implemented by an inference engine which repeatedly chooses the most appropriate of these rules or logical statements based on the current state of the system. The control strategy then performs the actions specified by that rule or logical statement to effect a change in the state of the system. In discrete-event simulation models, the knowledge is represented in the form of events and the control strategy repeatedly chooses the next imminent event, i.e. the one which is to occur next in simulated time, advances the simulation clock to the time the imminent event is scheduled to occur, and executes the code associated with the event to effect a change in the state of the system. In both cases, this allows the programmer (i.e. knowledge engineer or simulation modeler) to concentrate on developing the domain knowledge without having to worry about the exact order in which the individual chunks of knowledge are applied in solving a particular problem instance.

Another important similarity is that both fields use similar methodologies for representing the current state of the system. This is most often some variation of the entity-attribute approach. The entity-attribute approach was developed as a modeling technique in which the system is decomposed into its constituent components which are called entities. Each entity is then modeled in terms of its attributes where each attribute represents some aspect of the entity. In knowledge-based systems, the entity-attribute approach serves as the basis for the frames knowledge representation technique which extends the entity-attribute approach by modeling attributes in terms of their facets. Each facet of an attribute represents a different aspect of the attribute, and may include procedural attachments called daemons which are activated in response to references to the attribute. In addition, operations on an entity are specified as procedural attachments on the entity itself and are activated in response to messages directed at the entity. A programming system based on frames is called an object-oriented programming system.

Finally, an iterative development methodology is regarded by both fields as the appropriate approach to system development. In discussing his methodology of simulation model development, Shannon stated that "the evolutionary nature of model building is inevitable and desirable" [Shannon 1975]. This is comparable to the accepted knowledge engineering methodology where the "system evolves by proceeding from simple to increasingly hard tasks, improving incrementally the organization and representation of knowledge" [Hayes-Roth 1983]. The iterative development

methodology has been extended by knowledge-based development environments to provide rapid prototyping of knowledge-based systems.

Figure 1 summarizes the similarities between conventional discrete-event simulation techniques and knowledge-based programming techniques.

One important difference between knowledge-based programming techniques and conventional discrete-event simulation techniques is the development environments within which each is typically implemented. Knowledge-based programming tools and languages are typically implemented using, or as extensions to, symbolic programming languages with interactive development environments on personal workstations, whereas conventional discrete-event simulation tools and languages are typically implemented using, or as extensions to, conventional programming languages using modest development environments provided by traditional mini-computer/main-frame operating systems. The fact that these conventional programming languages can be efficiently translated into executable machine code while the symbolic programming languages rely on either an interpretive execution environment or a non-conventional hardware architecture has hindered the integration of knowledge-based techniques into conventional programming environments, and vice versa.

The interactive development environments within which knowledge-based languages and tools exist rely on the interpretive nature of symbolic programming languages to allow changes in the program to be immediately reflected in the system being developed. This allows knowledge-based systems to support the iterative development method, not only at a macro level, but at all levels of development. This is the basis of the popularity of the use of rapid prototyping techniques in the development of knowledge-based systems.

Conventional Discrete-Event Simulation Techniques	Knowledge-Based Programming Techniques
<ul style="list-style-type: none"> • Domain knowledge separate from control strategy 	<ul style="list-style-type: none"> • Domain knowledge separate from control strategy
<ul style="list-style-type: none"> • Entity-attribute data organization 	<ul style="list-style-type: none"> • Object-oriented data organization with active objects
<ul style="list-style-type: none"> • Iterative development 	<ul style="list-style-type: none"> • Iterative development with rapid prototyping

Figure 1. Similarities Between Conventional Discrete-Event Simulation and Knowledge-Based Programming Techniques

Another important difference between knowledge-based programming techniques and conventional discrete-event simulation techniques is their ability to deal with the concept of time within their respective domains. The concept of time is an inherent part of dynamic simulation models. The discrete-event simulation techniques that have been developed to construct such models are centered around providing an appropriate abstraction of time. On the other hand, the inferencing mechanisms used within knowledge-based systems, e.g. rule-based or logic-based approaches, are typically confounded by time-varying data. Dealing with these problems requires the specification of additional meta-knowledge which defines the dynamic effects of changes to time-varying data used in deducing other data.

Figure 2 summarizes the differences between conventional discrete-event simulation techniques and knowledge-based programming techniques.

Conventional discrete-event simulation techniques and knowledge-based programming techniques complement each other when integrated into a unified knowledge-based simulation development environment. The knowledge-based programming techniques provide the enhanced data abstraction capabilities associated with object-oriented programming techniques, rule-based or logic-based inferencing capabilities for dealing with complex decision processes, and a superior development environment for supporting incremental development. The conventional discrete-event simulation techniques provide the basic entity-attribute approach for data abstraction and the capabilities to deal with time-varying data within the domain.

2.0 Design of a Knowledge-Based Simulation Environment

There are many techniques from conventional simulation modeling and knowledge-based programming which are applicable to knowledge-based simulation modeling. Applicable conventional simulation modeling techniques include:

- the entity-attribute approach,
- the event-scheduling control strategy, and
- the process-interaction control strategy.

Conventional Discrete-Event Simulation Techniques	Knowledge-Based Simulation Techniques
<ul style="list-style-type: none"> • Event scheduling approach to control • Time central concept in control strategy • Quantitative models (i.e., primarily numeric data) • Model data probabilistic • Model results probabilistic, model is run many time and results distributions computed 	<ul style="list-style-type: none"> • Rule-based or logic-based inferencing • Time normally not used in inferencing • Qualitative models (i.e., primarily symbolic data) • Model data have associated certainty factors • Model results deterministic, may have associated certainty factors

Figure 2. Differences Between Conventional Discrete-Event Simulation and Knowledge-Based Programming Techniques

Applicable knowledge-based programming techniques include:

- object-oriented programming,
- rule-based programming,
- forward-chaining inferencing, and
- backward-chaining inferencing.

These techniques must be analyzed to eliminate redundancy and to insure compatibility when integrated into a knowledge-based simulation development environment.

The entity-attribute approach and object-oriented programming both provide for the representation of data in their respective domains. Object-oriented programming provides all of the techniques needed to represent model entities, their attributes, and operations on them in a unified manner. Object-oriented programming therefore provides all of the capabilities of the entity-attribute approach and extends it to include procedural operations on entities.

Knowledge-based simulation models must be able to deal adequately with the concept of time. This requires the inclusion of a simulation clock and a mechanism for maintaining its value. The event-scheduling control

strategy is the obvious choice for providing these timing functions. An alternative approach would be to put the simulation clock on the blackboard and allow rules to access and/or update its value. This would allow rule-based programming to be used for maintaining the value of the simulation clock. This approach, however, lacks the clarity and efficiency of the event-scheduling approach.

The process-interaction approach unifies the event-scheduling control strategy and object-oriented programming. A process represents an active entity in the model and is represented in the same manner as any other entity using object-oriented programming. In addition to having attributes and operations specified, a process has associated events which define its dynamic behavior. Individual events within a process may communicate via the attributes of its associated process instance.

Rule-based programming is used to model complex decision-making processes within a knowledge-based simulation model. All decisions are assumed to be instantaneous in simulated time, therefore rules must not use the simulation clock in any of their preconditions and must not alter the value of the simulation clock in any of their actions. This precludes the possibility of any conflicts between the event-scheduling control strategy and the inferencing control strategy. Any event may initiate an inferencing procedure to model a decision-making process, and any rule action may schedule events to be executed or modify the future event list to effect changes in future event execution.

Either forward-chaining or backward-chaining, or both, may be used in implementing an inferencing control strategy. It is possible to structure rules such that the same rule may be used in both forward-chaining and backward-chaining inferencing. However, because backward-chaining requires the ability to identify the facts that a rule's actions may place on the blackboard, this can only be done by restricting the actions that a rule is allowed to perform. The prototype development environment is therefore restricted to using a forward-chaining inferencing mechanism in its inference engine.

The language for the knowledge-based simulation model development environment is based on the Common Lisp programming language. There are several reasons for this choice:

1. Common Lisp provides features to support the embedding of new language features within the language, most notably, the macro facility which allows new special forms to be added to the language.
2. There are object-oriented programming systems available within Common Lisp implementations. One of these can serve as the basis for the object-oriented programming system.
3. There are sophisticated development environments available which support incremental development and rapid prototyping.

The Common Lisp implementation used is Symbolics Common Lisp which runs on the Symbolics Lisp Machine.

The language elements that comprise the knowledge-based simulation development environment are divided into five protocols:

1. The modeling protocol provides the language elements for the object-oriented programming system for the representation of model elements.
2. The simulation protocol provides the language elements to extend the modeling protocol for the definition of the dynamic simulation elements.
3. The simulation control protocol provides the language elements for the control of the dynamic simulation elements.
4. The inferencing protocol provides the language elements for the rule-based programming techniques for the modeling of complex decision processes.
5. The inferencing control protocol provides the language elements for the control of the forward-chaining inferencing mechanism.

Each of these protocols will be discussed in the following sections.

2.1 Modeling Protocol

The modeling protocol provides the static modeling language elements. The modeling protocol is provided by the Flavors package which is an object-oriented programming package implemented on top of Common Lisp. A complete description can be found in the Symbolics Common Lisp Language Concepts Vol. 2A. [Bromley et al 1987; Symbolics 1988a; Symbolics 1988b]

The object-oriented programming package provided by the Flavors package uses terminology that differs from similar packages. Entities classes are represented by flavors. A flavor serves as a template for all objects in the corresponding entity class. Each object is an instance of a flavor. The entity attributes for a flavor are represented by instance variables. Each instance of a flavor has its own values for its instance variables, and the values of these instance variables define the state of the instance. Each instance variable may have a default initial value and may be defined to be inittable, i.e., its value may be specified when an instance is created; readable, i.e., its value may be read via a function call; and/or writable, i.e., its value may be updated via a call to the `setf` macro. There are no class variables for a flavor, though the same concept may be easily

emulated using the property list of the symbol naming the flavor. [Bromley et al 1987; Symbolics 1988a; Symbolics 1988b]

The operations for the entity class represented by a flavor are defined as methods on the flavor. A method is a function associated with a flavor. When the method is activated for a specific flavor instance, the variable `self` is bound to the instance and all of the instance variables for the instance are available to the method as local variables. [Bromley et al 1987; Symbolics 1988a; Symbolics 1988b]

One of the main strengths of the Flavors package is the ability to combine existing flavors with newly defined flavors. When defining a flavor, a list of component flavors whose characteristics are to be included in the new flavor is specified. The instance variables and methods of the component flavors are inherited by the new flavor. [Bromley et al 1987; Symbolics 1988a; Symbolics 1988b]

2.2 Simulation Protocol

The simulation protocol provides the dynamic modeling language elements. These language elements provide for the definition of processes and the events which implement their actions. The protocol also provides language elements for process creation, interprocess communication, and other operations on processes and events. The simulation protocol implements a process interaction approach to discrete event simulation.

A process represents an active entity in the reference system. Processes may have instance variables to represent attributes unique to each process instance like other entities. A process is defined using the `defprocess` macro. A call to this macro has the following form:

```
defprocess name
  &rest instance-variables
```

where *name* is a symbol that is the name of the process being defined, and each *instance-variable* is the name of an attribute of the process and has the same format as an instance variable specification in a flavor definition. The primary action of the macro is to define a flavor whose name is *name* based on the abstract flavor `process`. The macro also defines a predicate to recognize instances of the process. The name of this predicate function is the print name of *name* with "-P" appended.

The simulation protocol has two mechanisms for creating process instances. The more primitive mechanism for creating a process instance is the `create-process` function. This function creates a new process instance and optionally initializes some instance variables in the newly created process instance. A call to this function has the following form:

```
create-process name
  &rest initializations
```

where *name* is the name of the process of which to create an instance, and the *initializations* are optional keyword/value pairs specifying initial values for process instance variables. The newly created process instance is returned as the value of the function call.

Most processes have an initial event which is scheduled to execute as the first event after the process is created. The simulation protocol provides an `initiate` macro which simplifies the normal process of process creation and initial event scheduling. The `initiate` macro creates a new process instance, optionally initializes some instance variables in the newly created process instance, and schedules its initial event for execution at some specified simulated time. A call to this macro has the following form:

```
initiate time name
        &rest initializations
```

where *time* is the simulated time at which the initial event of the process is to be executed or `:now` if the initial event is to be scheduled to execute immediately, *name* is the name of the process to be initiated, and the *initializations* are optional keyword/value pairs specifying initial values for process instance variables. Note that the *name* argument is not evaluated. The newly created process instance is returned as the value of the macro call.

The actions of an active entity are specified as events within the process representing the active entity. Each event has a parameter list and accepts argument like a normal Common Lisp function, but rather than being executed immediately in response to a function call their executions are scheduled and performed in their proper sequence by the event-scheduling control strategy. Note that a process may also have normal methods defined on them using the `defmethod` macro. An event is defined using the `defevent` macro. There are two types of events: process events and non-process events. As the names suggest, a process event is an event which is associated with some process, whereas a non-process event is not. A call to this macro has the following form:

```
defevent
  (name
   &optional process
   &key :initial-event)
  lambda-list
  &body body
```

where *name* is a symbol that is the name of the event being defined, *process* is the name of the process with which this event is associated or `nil` (the default) for a non-process event, the key `:initial-event` is true if this event is the event to be scheduled when the associated *process* is initiated and `nil` (the default) otherwise. The *lambda-list* specifies the arguments to this event and may contain any structure valid in a function

lambda list. The *body* specifies the forms that are to be executed as the body of this event.

There are often cases where an event executing within a process instance is required to communicate with another process instance. This may be to read or update an instance variable within the referenced process instance or to schedule an event within the referenced process instance. A common example of this occurs when there is some initial handshaking operations that must be performed between a process instance and its initiator. The `with-process` macro provides a convenient mechanism for performing these actions. A call to this macro has the following form:

```
with-process
  (process
   &optional instance-form)
  &body body
```

where *process* is the symbol naming the process whose instance is required, and *instance-form* is an optional form which, when evaluated, returns the desired process instance. The *body* specifies the forms that are to be evaluated with *process* bound to the referenced process instance. If an *instance-form* is not given, or is `nil`, then *process* is bound to the ancestor process of the indicated type.

The final language element provided by the simulation protocol is a macro to iterate over the events defined within a process. The `do-process-events` macro provides this iteration capability. A call to this macro has the following form:

```
do-process-events
  (var process
   &optional result-form)
  &body body
```

The effect of a call to the `do-process-events` macro is to evaluate the forms in *body* with the symbol *var* bound to successive events in *process*. After all of the events have been exhausted, *result-form*, which defaults to `nil`, is evaluated and the result returned as the value of the call.

2.3 Simulation Control Protocol

The simulation control protocol provides the language elements for the creation and manipulation of simulation environments. The most important elements of the simulation environment are the future event list and the simulation clock. A simulation environment provides the data structures for the process interaction control strategy of a simulation model. Multiple simulation environments may exist simultaneously either in a hierarchical (i.e., nested) manner or as independent environments within separate Common Lisp dynamic referencing environments.

A simulation environment automatically exists within each independent Common Lisp dynamic referencing environment. To create a new simulation environment within an existing simulation environment, the `with-new-simulation-environment` macro is used. A call to this macro has the following form:

```
with-new-simulation-environment
  &body body
```

where *body* contains the forms needed to implement the encapsulated simulation model. These forms would normally include one or more process initiations followed by a `start-simulation` function call (see below).

A simulation environment can be reset to its initial state using the `reset-simulation-environment` function. A call to this function has the following form:

```
reset-simulation-environment
```

The effect of a `reset-simulation-environment` function call is to clear the future event list by releasing all of the previously scheduled events and to reset the simulation clock to zero. The effect of the call is limited to the innermost simulation environment in the current Common Lisp dynamic referencing environment.

The process interaction control strategy is initiated within a simulation environment using the `start-simulation` function. A call to this function has the following form:

```
start-simulation
```

The effect of a `start-simulation` function call is to begin the event scheduling control strategy of removing event notices from the future event list, advancing the simulation clock, and executing the corresponding event code. This process continues until either the future event list is empty or a dynamic throw is executed whose catch is not within the dynamic environment of the control strategy. The effect of the call is limited to the innermost simulation environment in the current Common Lisp dynamic referencing environment.

The `schedule` function calls the scheduler to add a new event notice to the future event list representing the future execution of an event. A call to this function has the following form:

```
schedule
  time event process-instance
  &rest arguments
```

where *time* is a number representing the time the event is to be scheduled or `:now` if the event is to be placed at the head of the future event list to be executed at the current simulation time as the next event, *event* is the name of the event to be scheduled, *process-instance* is the process instance which the scheduled event is to be associated, and *arguments* are the actual arguments to the scheduled event. The effect of the call is that an event notice is created representing the future event, and this event notice is placed at the correct position on the future event list.

It is sometimes necessary within a simulation model to make decisions based on already scheduled events, or to remove or reschedule such events. The `do-scheduled-events` macro provides a mechanism for doing this. A call to this macro has the following form:

```
do-scheduled-events
  (var &optional
    event-list result-form)
  &body body
```

The effect of a call to the `do-scheduled-events` macro is to evaluate the forms in *body* with the symbol *var* bound to successive event notices on *event-list*, which defaults to the future event list in the current simulation environment. After all of the event notices have been exhausted, *result-form*, which defaults to `nil`, is evaluated and the result returned as the value of the call.

2.4 Inferencing Protocol

The inferencing protocol provides the language elements for the rule-based modeling of complex decision-making processes. These language elements provide for the definition of rulesets and the rules specifying their problem-solving knowledge. The protocol also provides language elements for ruleset creation and other operations on rulesets and rules. The inferencing protocol implements a rule-based programming system.

The basic elements of the inferencing protocol are rules and rulesets. A rule is a single piece of problem-solving knowledge expressed as a set of preconditions for the application of the rule and a set of actions that are to be performed when the rule is executed. Related rules are grouped together to form rulesets which may be activated as needed in response to problem-solving demands.

A ruleset represents a body of problem-solving knowledge. A ruleset may also have instance variables to represent attributes unique to each ruleset instance. A ruleset is defined using the `defruleset` macro. A call to this macro has the following form:

```
defruleset name
  &rest instance-variables
```

where *name* is a symbol that is the name of the ruleset being defined, and each *instance-variable* is the name of an attribute of the ruleset and has the same format as an instance variable specification in a flavor definition. The primary action of a call to the macro is to define a flavor whose name is *name*, based on the abstract flavor *ruleset*. The macro also defines a predicate to recognize instances of the ruleset. The name of this predicate function is the print name of the *name* with "-P" appended.

A ruleset instance must be created to perform inferencing using the rules in the ruleset. The `activate` macro creates a new ruleset instance optionally initializes some instance variables in the newly created ruleset instance, and makes the rules in the ruleset eligible for firing by adding the newly created ruleset instance to the list of active rulesets. A call to this macro has the following form:

```
activate name
  &rest initializations
```

where *name* is the name of the ruleset to be activated, and the *initializations* are optional keyword/value pairs specifying initial values for ruleset instance variables. Note that the *name* argument in an `activate` macro call is not evaluated. The newly created ruleset instance is returned as the value of the macro call.

A rule represents a single piece of problem-solving knowledge and is associated with a particular ruleset. The specification of a rule includes a set of precondition patterns that are to be matched against facts on the blackboard and a set of actions which are forms to be evaluated in an environment binding the variables in the precondition patterns to their matching elements in the matched facts. A rule is defined using the `defrule` macro. A call to this macro has the following form:

```
defrule (name ruleset)
  &rest preconditions-and-actions
```

where *name* is the name of the rule being defined, *ruleset* is the name of the ruleset with which this rule is associated, and the *preconditions-and-actions* have the following form:

```
{preconditions}*
==>
{actions}*
```

where each *precondition* is a pattern to be matched against facts on the blackboard, and the *actions* are the forms to be evaluated in an environment binding the variables in the precondition patterns to their matching elements in the match facts when the rule is executed.

The final language element provided by the inferencing protocol is a macro to iterate over the rules defined within a ruleset. The `do-ruleset-rules` macro provides this iteration capability. A call to this macro has the following form:

```
do-ruleset-rules
  (var ruleset
   &optional result-form)
  &body body
```

The effect of a call to the `do-ruleset-rules` macro is to evaluate the forms in *body* with the symbol *var* bound to successive rules in *ruleset*. Note that the *ruleset* argument is evaluated. This is to allow dynamic specification of the ruleset name. After all of the rules have been exhausted, *result-form* which defaults to `nil` is evaluated and the result returned as the value of the call.

2.5 Inferencing Control Protocol

The inferencing control protocol provides the language elements for the creation and manipulation of inferencing environments. The most important elements of the inferencing environment are the blackboard and the agenda. An inferencing environment provides the data structures for the forward-chaining inferencing control strategy for the rule-based components of a knowledge-based simulation model. Multiple inferencing environments may exist simultaneously either in a hierarchical (i.e., nested) manner or as independent environments within separate Common Lisp dynamic referencing environments.

An inferencing environment automatically exists within each independent Common Lisp dynamic referencing environment. To create a new inferencing environment within an existing inferencing environment, the `with-new-inferencing-environment` macro is used. A call to this macro has the following form:

```
with-new-inferencing-environment
  &body body
```

where *body* contains the forms needed to implement the encapsulated rule-based inference system. These forms would normally include one or more ruleset activations and one or more assertions followed by a `start-inferencing` function call (see below).

An inferencing environment can be reset to its initial state using the `reset-inferencing-environment` function. A call to this function has the following form:

```
reset-inferencing-environment
```

The effect of a `reset-inferencing-environment` function call is to clear the blackboard by removing all of the previously asserted facts and to clear the agenda by removing all of the previously triggered rule instances. The effect of the call is limited to the innermost inferencing environment in the current Common Lisp dynamic referencing environment.

The forward-chaining inferencing control strategy is initiated within an inferencing environment using the `start-inferencing` function. A call to this function has the following form:

```
start-inferencing
```

The effect of a `start-inferencing` function call is to begin the forward-chaining control strategy of matching asserted facts against preconditions of the rules in the active rulesets to form the conflict set, choosing the most appropriate of these rule instances for application, and applying the rule in the appropriate ruleset instance. This process continues until either there are no rule instances in the conflict set after the matching phase of the forward-chaining inferencing algorithm has been executed or a dynamic throw is executed whose catch is not within the dynamic environment of the control strategy. The effect of the call is limited to the innermost inferencing environment in the current Common Lisp dynamic referencing environment.

A fact is a statement about the domain that has either been explicitly given by the knowledge-engineer in the code of the program or by the user in response to some action by the program or has been deduced from these basic facts using the rules in the knowledge base. A fact is represented by a list structure whose structure is determined by the knowledge engineer. The blackboard is a data structure which contains the facts that have been asserted in the current inferencing environment. The `add-fact` function adds a new fact to the blackboard. A call to this function has the following form:

```
add-fact fact
```

where *fact* is the list structure representation of the fact to be added to the blackboard.

Within the actions of a rule, it is often necessary to remove the fact which matched a particular precondition pattern. The `remove-matching-fact` function removes from the blackboard the assertion representing the fact that matched a given precondition pattern. A call to this function has the following form:

```
remove-matching-fact n
```

where *n* is the number of the precondition pattern whose matching fact is to be removed from the blackboard.

It is sometimes necessary to remove a number of related facts from the blackboard. The `remove-assertions` function removes existing facts from the blackboard that match a given pattern. A call to the `remove-assertions` function removes some facts from the blackboard. A call to this function has the following form:

```
remove-assertions pattern
```

where *pattern* is a list structure representation of a pattern to match against facts on the blackboard. All facts on the blackboard matching *pattern* are removed.

3.0 CONCLUSION

The knowledge-based simulation development language presented has been used in the construction of several knowledge-based simulation models. These include:

- The Advanced Launch System Model (ALSYM) which includes an end-to-end model of the entire Advanced Launch System (ALS) industrial infrastructure.
- The Space Station Freedom Model which provides system available measures for the operational phase of the space station and its support infrastructure.
- The Software Development Process Model which models the performance of a software development organization.

These techniques have provided a capability to develop early prototype models in support of trade studies during the concept development phase of projects as well as detailed models to provide decision support for managers of operational systems.

REFERENCES

- Banks, J. and Carson, J. S., II, **Discrete-Event System Simulation**, Prentice-Hall, 1984
- Bromley, H. and Lamson, R., **LISP Lore: A Guide to Programming the LISP Machine**, Second Edition, Kluwer Academic Publishers, 1987
- Brownston, L., Farrell, R., Kant, E., and Martin, N., **Programming Expert Systems in OPS5**, Addison-Wesley, 1985

Ferguson, E., "*Using Scheme for Discrete Simulation*", **Texas Instruments Engineering Journal**, Vol. 3, No. 1, Jan.-Feb. 1986

Hayes-Roth, F., Waterman, D. A., and Lenat, D. B., **Building Expert Systems**, Addison-Wesley, 1983.

Kerckhoffs, E. J. H. and Vansteenkiste, G. C., "*The Impact of Advanced Information Processing on Simulation - An Illustrative Review*", **Simulation**, Jan. 1986, Vol 46, No. 1, pp. 17-26

Kulikowski, C. A., "*Artificial Intelligence, Modeling, and Simulation*", **Artificial Intelligence, Expert Systems and Languages in Modeling and Simulation**, North-Holland, Jun. 1987, pp. 5-13

Murray, K. J. and Sheppard, S. V., "*Knowledge-Based Simulation Model Specification*", **Simulation**, Mar. 1988, Vol. 50, No. 3, pp. 112-119

O'Keefe, R. M., "*The Role of Artificial Intelligence in Discrete-Event Simulation*", **Artificial Intelligence, Simulation, and Modeling**, John Wiley and Sons, 1989, pp. 359-380

Round, A., "*Knowledge-Based Simulation*", **The Handbook of Artificial Intelligence**, Vol. IV, Addison-Wesley, 1989, pp. 417 - 518

Shannon, R. E., **System Simulation: The Art and Science**, Prentice Hall, New Jersey, 1975

Shannon, R. E., Mayer, R., and Adelsberger, H. H., "*Expert Systems and Simulation*", **Simulation**, Jun. 1985, Vol. 44, No. 6

Smith, H.R. and McVicar, K., "*Knowledge-Based Simulation with Frameworks*", **Proc. of Multiconference 1988**, Feb. 1988

Steele, G. L., Jr., **Common LISP: The Language**, Digital Press, 1984

Symbolics Common LISP - Language Concepts, Symbolics, Inc., 1988

Symbolics Common LISP - Language Dictionary, Symbolics, Inc., 1988

Waterman, D. A., **A Guide to Expert Systems**, Addison-Wesley, 1985

Widman, L. E. and Loparo, K. A., "*Artificial Intelligence, Simulation, and Modeling: A Critical Survey*", **Artificial Intelligence, Simulation, and Modeling**, John Wiley and Sons, 1989, pp. 1-44

Williams, M. D. and Smith, M. E., "*Knowledge-Based Simulation*", **Proc. of the Second International Software for Strategic Systems Conference**, Oct. 1988

The Composite Load Spectra Project

J. F. Newell and H. Ho
Rockwell International, Rocketdyne Division
6633 Canoga Avenue
Canoga Park, CA 91303

R. E. Kurth
Battelle Columbus Laboratory
505 King Avenue
Columbus, OH 43201

Abstract

The objective of the composite load spectra project is to develop probabilistic methods and generic load models capable of simulating the load spectra that are induced in space propulsion system components. Four engine component types (the transfer ducts, the turbine blades, the liquid oxygen posts and the turbopump oxidizer discharge duct) were selected as representative hardware examples. The composite load spectra that simulate the probabilistic loads for these components are typically used as the input loads for a probabilistic structural analysis.

The knowledge-based system approach used for the composite load spectra project provides an ideal environment for incremental development. The intelligent database paradigm employed in developing the expert system provides a smooth coupling between the numerical processing and the symbolic (information) processing. Large volumes of engine load information and engineering data are stored in database format and managed by a database management system. Numerical procedures for probabilistic load simulation and database management functions are controlled by rule modules. Rules were hard-wired as decision trees into rule modules to perform process control tasks. There are modules to retrieve load information and models. There are modules to select loads and models to carry out quick load calculations or make an input file for full duty-cycle time dependent load simulation. The composite load spectra load expert system implemented today is capable of performing intelligent rocket engine load spectra simulation. Further development of the expert system will provide tutorial capability for users to learn from it.

I. Objective and Approach

The objective of the composite load spectra (CLS) program under the sponsorship of NASA Lewis Research Center (LeRC) (Ref. 4) is to develop generic load models to simulate loads that are induced in space propulsion system components. Representative engine components that are considered in the study are transfer ducts, turbine blades, liquid oxygen (LOX) posts and engine system ducts. The simulated loads from the CLS load models are being compared to available test results and other analysis results. These probabilistic loads are needed in the probabilistic structural analyses for load effect and perturbation study. They are used primarily as input loading for the probabilistic structural analysis methodology (PSAM).

Current load analysis methodologies use deterministic models. Conservative bounding techniques are applied in load analyses and design. The design can be very conservative as a result of superposition of several individual bounding loads. In other cases there is insufficient information available to define realistic bounding loads and their variations. Measurement of loads and responses internal to hardware are very difficult due to a combination of high pressures, temperatures, flowrates and pump speeds. The probabilistic method allows a rational approach for quantifying these uncertain loads. They can then be utilized for structural response and reliability evaluations during the design phase. Probabilistic design analysis can help to locate the problem areas and allow cost-effective trade-offs to reach design goals.

Probabilistic load synthesis demands sophisticated methodology and modeling. It requires knowledge of state-of-the-art space propulsion system load analysis and calculation tools. These analyses and calculations involve intensive numerical processing. The load synthesis also demands knowledge on the engine model that determines pressures, temperatures, and flows and the load information. Management of the load information becomes an important issue.

The end product of the CLS program is a system that can help engineers generate loads on components from a set of primitive variables and their uncertainties based on previous historical information or expert opinion on expected variations. The system provides load information and data conveniently to the users and provides expertise in load evaluation.

These objectives point to a knowledge-based system (Ref. 3 and 9) approach that can provide knowledge of the space propulsion system and component loads and provide expertise in probabilistic methodology and load simulation. The knowledge-based system should be a coupled system for symbolic and numeric processing, should be able to manage a large volume of load information and data, should provide easily accessible information to users, and needs to be user friendly so that nonexpert users can use and learn from it.

The knowledge-based system approach has the facilities to encompass the knowledge of the space propulsion system and its loads, the numerical databases for load parameters, and load evaluation procedures. In addition, the knowledge-based system environment allows incremental development and modularization of the knowledge. Modules of knowledge (e.g., the load model, load data, or load calculation procedures) can be implemented and readily available to other modules.

The knowledge-based system built for the CLS program is an intelligent database system. The engine load models and load information such as the load distribution parameters (mean and coefficient of variation, and distribution type) are stored in database format. Rule modules in decision tree format are implemented to provide intelligence to the system to perform consultation, data retrieval and preparation of data for load simulation evaluation. Successful implementation of the basic system and experimentation with it has been accomplished in the last few years. It has demonstrated that an intelligent database system is one of the most appropriate approaches for an engineering knowledge-based application such as the CLS program.

Application of the CLS technology to synthesize component loads for the four sample components has been completed. The component loads were generated in the form of correlation fields accounting for the component load variations caused by the uncertainties of various engine parameters and engine inlet operation conditions. The component loads thus generated were utilized in the probabilistic structural analyses of the turbine blade (Ref. 6), the LOX post (Ref. 7), and high-pressure ducts (Ref. 1). Figures 1 and 2 depict the processes and analyses that are involved in the probabilistic structural analysis of the turbine blade. Figure 1 shows the process of applying the CLS technology to synthesize the turbine blade loads. Figure 2 shows how these loads are used in the probabilistic structural analysis process using the NESSUS (Nonlinear Evaluation of Stochastic Structures Under Stress) code (Ref. 8).

II. Engine Load Model and Load Databases

A space propulsion system is sophisticated and complex. Major subsystems typically include the main injector and combustion chamber, the nozzle, the high-pressure turbopumps, the low-pressure turbopumps, the ducts and pipings, and the control systems and valves. Figure 3 is a typical schematic showing these components. In the center are the main injectors, the combustion chamber and the nozzle. On the left are the low-pressure fuel turbopump (LPFTP) and the high-pressure fuel turbopump (HPFTP) that are interconnected to the rest

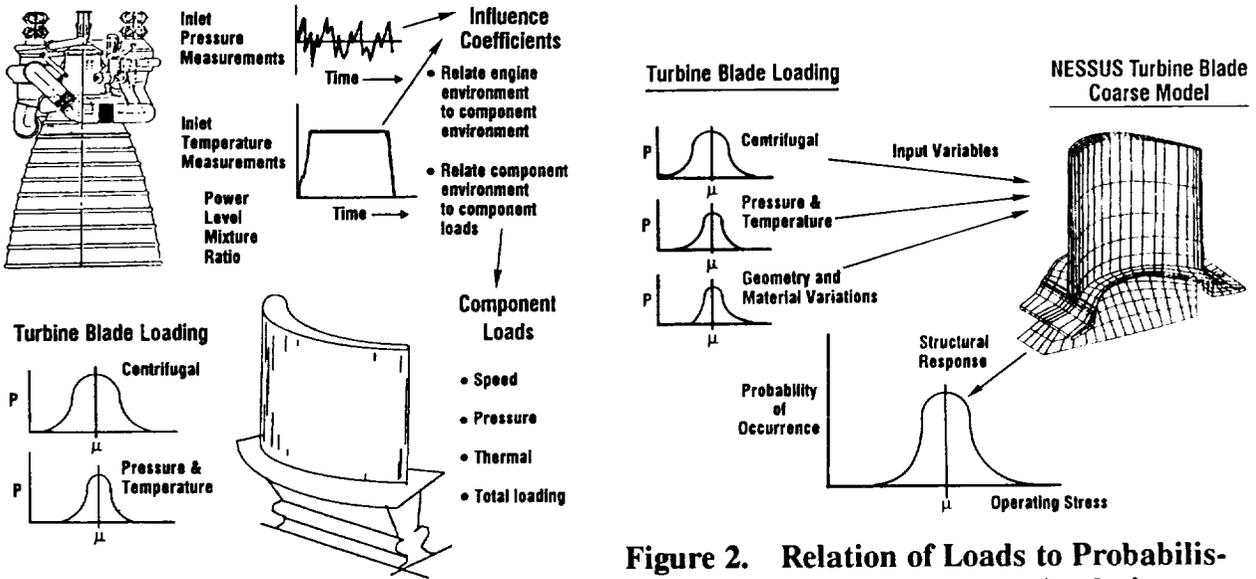


Figure 1. Typical Load Process

Figure 2. Relation of Loads to Probabilistic Structural Response Analysis

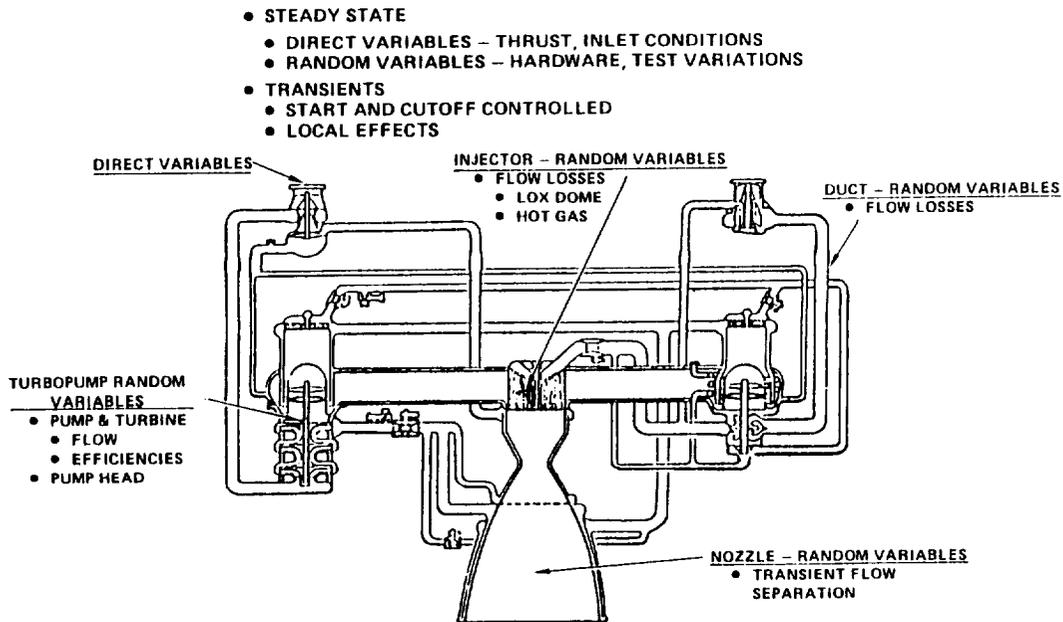


Figure 3. Typical Engine Schematic

of the engine by a series of ducting. On the right are the low-pressure oxidizer turbopump (LPOTP), the high-pressure oxidizer turbopump (HPOTP) and similar ducting.

The engine model implemented for the CLS is a numerical abstraction of the engine accounting for load variations as caused by various engine parameters and inlet conditions. It is a multilevel engine model as shown in Figure 4, which has been developed using a baseline engine model. The multilevel engine model is composed of the engine system influence model and various component load models. The engine system influence model is the foundation of the multilevel engine model, which allows various component load models to be built on it. The engine system model evaluates system performance variables and engine subsystem

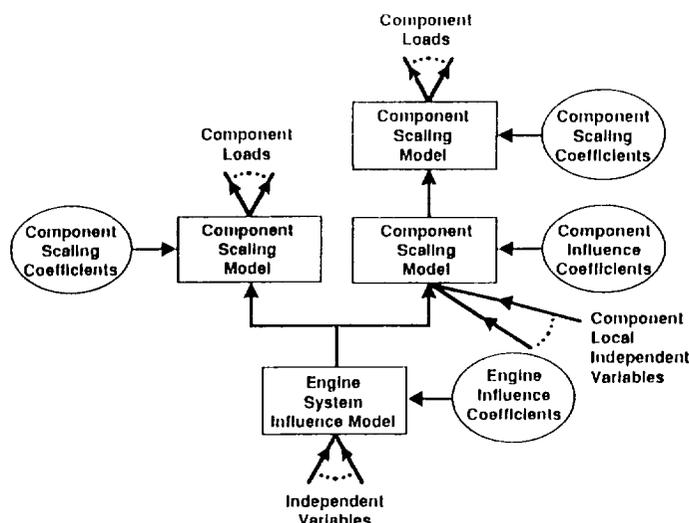


Figure 4. CLS Engine Model

operation loads (both types classified as system dependent loads) based on the engine operating power level, engine hardware and operating parameters (the engine hardware and operating parameters are classified as system independent loads). The component load models evaluate loads local to a component (classified as component loads) using the system loads as the component boundary loads (subsystem interface operating loads). Details of this model are reported in Reference 4.

The component load models are at the higher level of the multilevel engine model. The engine influence model calculates system interface variables based on engine performance. These system interface variables (e.g., turbine inlet pressure and temperature, and the turbine speed) provide the operating condition loads for the component load models to evaluate the component loads (e.g., the turbine blade nodal pressures and temperature, and the turbine blade centrifugal force).

The component load models are developed using several techniques and algorithms depending on the load type and the component. The general techniques include scaling and the probabilistic influence method. Scaling models include direct scaling with the system variable (e.g., the simple case of the turbine centrifugal force which is directly proportional to the square of the turbine speed) and the indirect scaling with a reference nodal load profile (e.g., those utilized in the turbine blade component pressure and temperature load models). The scaling technique is also used in normalized power density spectra for the fluctuation pressure loads and vibration loads. The probabilistic influence method is utilized in the component thermal load models. The probabilistic component load models retain the detailed deterministic analytical information inherent in the reference case analyses and yet provide a powerful algorithm to analyze the variations on different engine performance and operating conditions.

III. Probabilistic Methodology and Probabilistic Models

Probabilistic tools are required to generate the probabilistic engine loads (Ref. 5). The probabilistic methods available in the CLS expert system are (1) the Gaussian moment method, (2) RASCAL (Ref. 3), and (3) Monte Carlo. The Gaussian moment method is a moment propagation method which assumes that all of the load variables and engine parameters are normally distributed. The method referred to as the Quick Look Model (QLM) provides a fast, efficient method for determining the composite load distribution, if the basic variables' distributions are not severely skewed. The RASCAL method is a variance of the Discrete

Probability Distribution (DPD) method. Instead of combining all possible values, a Random Sampling Condensation Algorithm (RASCAL) was developed to handle the combination of random variables. The advantage of this method is that it is capable of handling standard distributional forms (e.g., normal, lognormal, Weibull, etc.) and nonstandard forms such as bimodal, and provides a range of levels for accuracy. This method can also be used to perform importance sampling, which can be used to examine regions of concern for the composite loads. Finally, the Monte Carlo method is also available. The Monte Carlo method can generate distributions with high accuracy and can calculate confident limits to access the accuracy of the predicted loads.

The engine loads experience three phases of operation during a mission: the engine start transient, the steady-state operation and the cutoff transient. The probabilistic tools required for load simulation must include probabilistic models that can handle transient states. For slowly varying loads, a quasi-steady-state approximation is provided. For transient loads with large fluctuation, the transient spike model and spike arrival model are available. A rare event model is also available to simulate a low probability event such as “pop,” an internal detonation caused by uneven burning. These probabilistic tools are available to be called upon by the load calculation module or other rule modules for transient load synthesis.

IV. The Load Expert System: LDEXPT Version 3.0

The load expert system, LDEXPT version 3.0, was implemented on NASA LeRC’s main-frame computer. The structure of the system is shown in Figure 5. The load expert system has a rule-based module (RBMS) and a knowledge-based module (KBMS). The rule-based module has the user interface system SESUIM, which takes care of the user query and answer functions. The rule-based driver controls the overall processing of running a user selected rule module, performing a load calculation with the ANLOAD module, etc. The knowledge-based module has a database system, a duty cycle data processing module and a file I/O module. The database system manages the knowledge base and takes care of database functions such as database retrieval and update. The file I/O module performs file input and output to the operating system.

The Knowledge Base

The domain knowledge for the probabilistic engine load synthesis of a space propulsion system consists of two main areas: the probabilistic methodology and modeling, and the rocket engine structural load information and evaluation. The probabilistic methods and calculation are implemented on the load expert system with the traditional algorithmic and procedural codes. These coding routines are included in the load calculation module ANLOAD. The load information and the load model information are implemented in the knowledge base. The information of the knowledge base is utilized and processed by the rule modules.

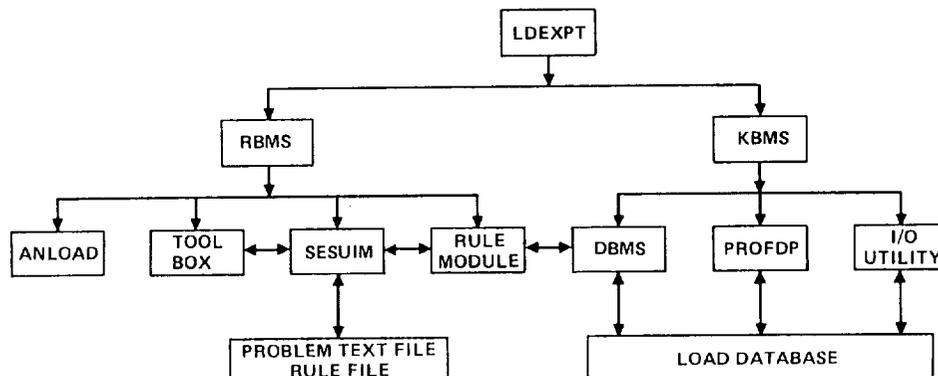


Figure 5. CLS Load Expert System

The synergism of the two knowledge domains and the coupling of the symbolic and numeric processing are brought about to shape a successful knowledge-based system for the CLS project.

The knowledge base of the load expert system is managed by the database system. The knowledge base has engine parameter information represented by their distribution parameters and distribution types (Figure 6). Data on the 64 system independent parameters, 99 system dependent parameters and numerous component loads are included. Their mean values, coefficients of variation and distribution type are stored in the knowledge base in database format. The influence coefficient set for the engine influence model also resides on the knowledge base.

The knowledge base also includes the information on the component load models available for each space propulsion component implemented on the system. Figure 7 lists some of the component load models implemented into the CLS knowledge base. The load dependency and scaling information of the component load models, the duty cycle data information, and component geometry information, etc., are required and included in the knowledge base. Much of the information is numerical. A rule-based system without a link to a database would have a difficult time managing the knowledge base. A dump database system would also have difficulty in handling the knowledge.

In addition to this, some of the knowledge includes the variations of the engine loads categorized as engine-to-engine variation, test-to-test variation and time slice-to-time slice (within a test) variation. The start transient event time line for a typical engine is delineated in Figure 8 for the first 4 seconds of a mission. The timing and operation of these events are critical in order to meet the stringent engine start transient thrust requirements. The best representation of the knowledge and how it would fit into the existing knowledge base is a task we are addressing at this time.

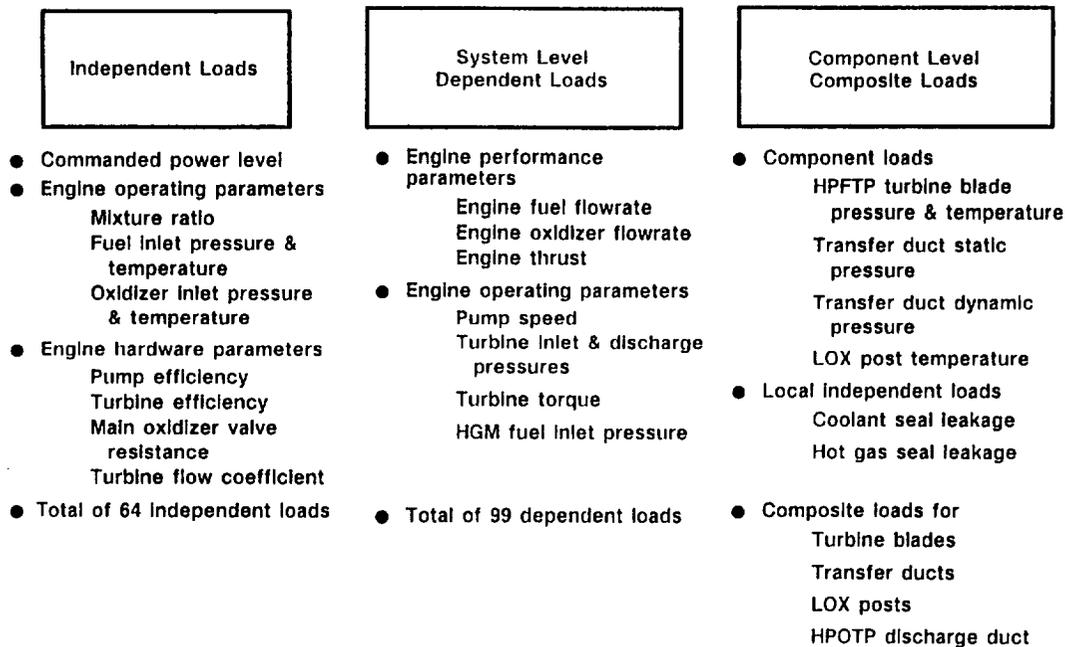


Figure 6. CLS Loads Knowledge Base

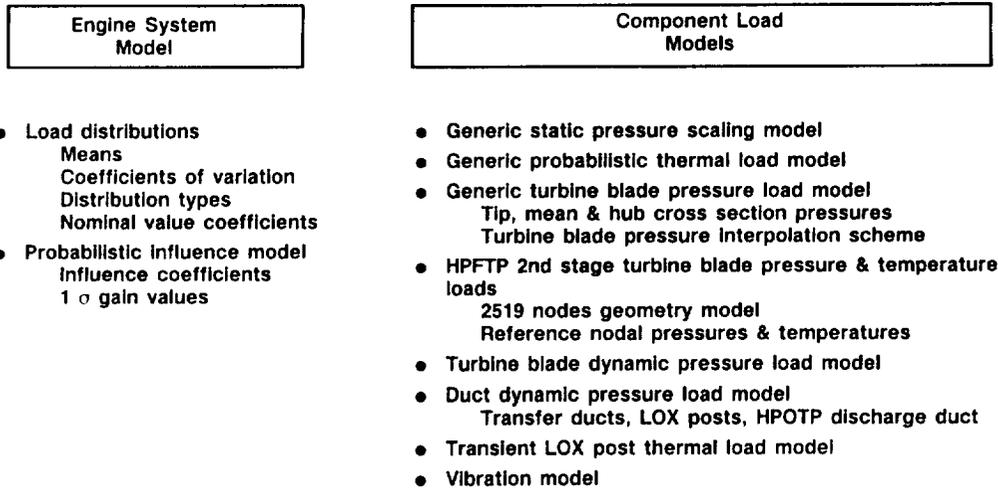


Figure 7. CLS Load Models Knowledge Base

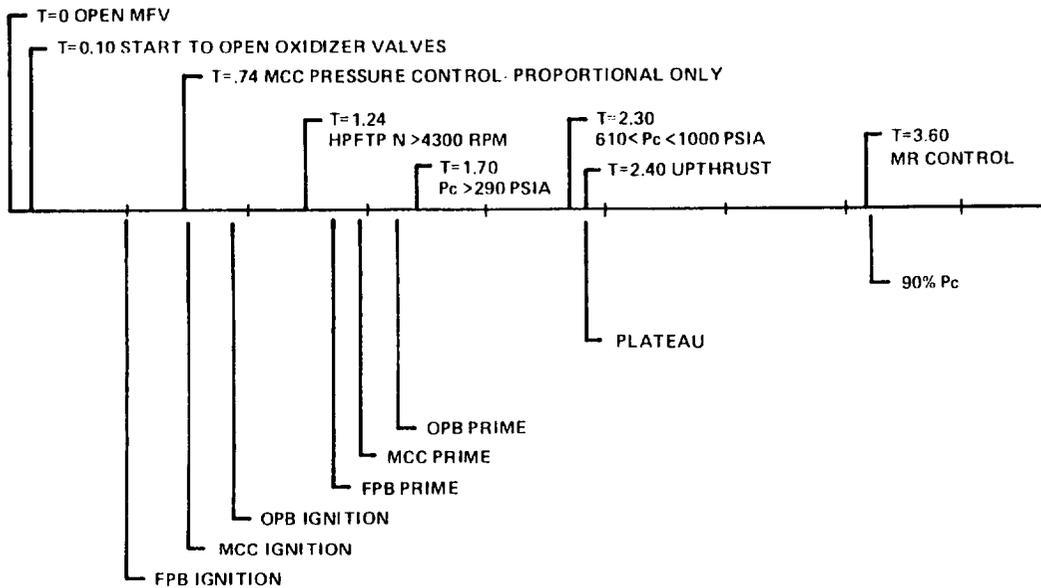


Figure 8. Engine Start Event Time Line

Validation and Verification

Validation and verification of the CLS expert system are essential to the success of the project. Validation of the correctness of the three implemented probabilistic methods and various models was carried out with makeup sample cases. The results of the case studies show that the three probabilistic methods (the Gaussian moment method, the RASCAL and the Monte Carlo method) and the various models perform as expected.

Verification of the CLS engine influence model and uncertain independent variables was performed by comparing the CLS results with those of the deterministic limit study of independent variables using measured data. Table 1 lists the results of the limit study. The results calculated by CLS agree with these results except those for the pump cavitation. Further investigation is in progress to improve our model. Verification of the load calculation was also performed by comparing the CLS calculations with the 10 second averaged database from the flight data and test data. In this study, only the effects of the following five independent vari-

Table 1. The Deterministic Limit Study Results

Independent Variable	Variation Ground Test	Difference	HPOTP Speed		HPFTP Speed		HPFTP Turbine Temperature		HPFTP Turbine Torque	
			Gain Effect	COV	Gain Effect	COV	Gain Effect	COV	Gain Effect	COV
Mixture Ratio	5.94 6 6.06	-0.06 0.06	-79.11 79.11	0.000938	138.402 -138.402	-0.00131	9.936 -9.936	-0.00180	105.81 -105.81	-0.00360
Fuel Inlet Pressure	20 30 45	-10 15	-0.59 0.885	0.000008	50.58 -75.87	-0.00074	4.25 -6.375	-0.00096	-13.34 20.01	0.000568
Oxidizer Inlet Pressure	23 100 160	-77 60	158.851 -123.78	-0.00167	16.016 -12.48	-0.00016	-9.471 7.38	0.001533	7.469 -5.82	-0.00022
Fuel Inlet Temperature	37 37 40	0 3	0 -5.55	-0.00003	0 679.335	0.004028	0 46.575	0.004238	0 113.49	0.001934
Oxidizer Inlet Temperature	163 164 171	-1 7	-58.07 406.49	0.002754	-2.77 19.39	0.000131	1.58 -11.06	-0.00115	-1.29 9.03	0.000175
Cavitation HPOTP	0 0.02 0.08	-0.02 0.06	-163.576 490.728	0.003879	-7.784 23.352	0.000184	4.608 -13.824	-0.00167	-3.632 10.896	0.000247
LPFTP Nozzle Area	0 5 20	-5 15	0.785 -2.355		-5.365 16.095		-6.8 20.4		-1.135 3.405	
Min Increment			-433.031		-242.671		-57.466		-131.027	
Max Increment			1130.414		804.768		81.446		143.491	
Individual Variation 3 Sigma			438.4848		579.6615		105.7691		415.1440	
Nom			28108		35131		1831.5		9779.6	
Max			29676.89		36515.42		2018.715		10338.23	
Min			27236.48		34308.66		1668.264		9233.428	
Gain %			8.682277		6.281523		19.13460		11.29704	
COV			0.014470		0.010469		0.031891		0.018828	

COV – coefficient of variation

D545-0016

ables were considered: the commanded mixture ratio, the fuel inlet pressure, the oxidizer inlet pressure, the fuel inlet temperature and the oxidizer inlet temperature. A few of the results are presented here. Figure 9 shows the calculated cumulative distribution functions (CDFs) for the LOX mass flowrate by the RASCAL method and the Monte Carlo method together with the engine flight and test data. Figure 10 presents the result for the high-pressure oxidizer turbine (HPOT) discharge temperature. In all cases, the calculated CDFs fit well or close to the engine flight data and not very well with the test data. This is to be expected because the five independent variables have significant effects on engine performance, whereas the engine hardware parameters are dominant during tests whose effects were not included in the calculation. These engine-to-engine variation effects are apparent from the step shift in the CDFs of the test data.

V. Summary

The development of the CLS technology is evolving. The composite loads synthesized by CLS have been successfully applied to the probabilistic structural analyses of the SSME turbine blade, the HPOTP discharge duct and the LOX post. The knowledge-based system

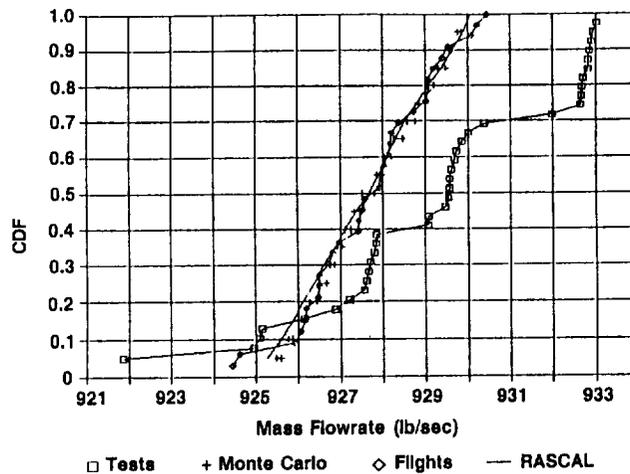


Figure 9. LOX Mass Flowrate Comparison (104% Power 10 Second Database)

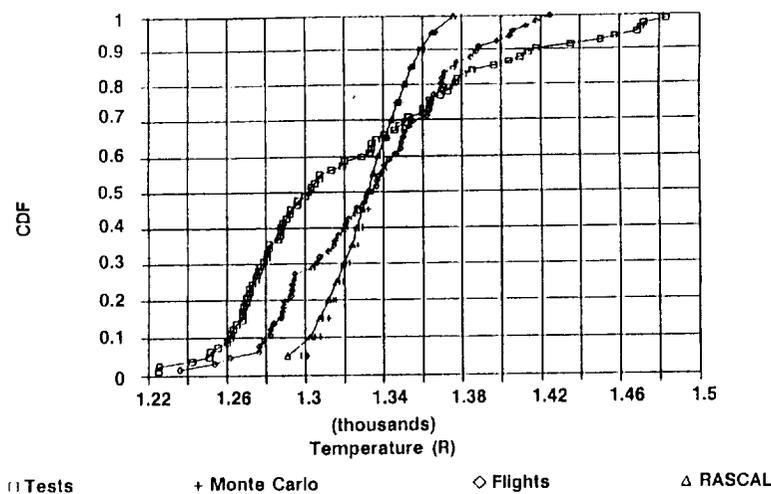


Figure 10. HPOT Discharge Temperature Comparison (104% Power 10 Second Database)

approach has provided an ideal environment for incremental development and modularization of the CLS system.

The intelligent database format has provided the crucial link between the engineering data and information and the decision tree inferencing. Most importantly, the knowledge and expertise of the Rocketdyne engineers on space propulsion system design, operation and analysis have made this a feasible research project.

The CLS expert system requires further development in areas such as making the CLS expert system more user friendly and making it a tutorial system that provides guidance to engineers in load synthesis.

The CLS has advanced the technology of the space propulsion system probabilistic load simulation. Correlation field approach is developed and being tested. Deeper understanding of engine flight and test data is evolving from the project. We are looking forward to more applications of the CLS technology to the probabilistic structural analysis, the structural reliability evaluation and life prediction of the space propulsion system components.

References

1. Debchaudhury, A., Rajagopal, K. R., Ho, H., and Newell, J. F., "A Probabilistic Approach to the Dynamic Analysis of Ducts Subjected to Multibase Harmonic and Random Excitation," to be presented at the 31st Structures, Structural Dynamics, and Materials Conference, Long Beach, California, April 1990
2. Feigenbaum, E. A. et al. (eds.), "The Handbook of Artificial Intelligence," Vol. I, II, & III, William Kaufmann, 1981-82
3. Kurth, R. E., "The Development and Application of A New Probabilistic Analysis Technique for Nuclear Risk Calculations," Ph.D. Dissertation, Ohio State University, 1985
4. Newell, J. F., Kurth, R. E., and Ho, H., "Composite Load Spectra for Select Space Propulsion Structural Components, First Annual Report," NASA Contractors Report NASA-CR-179496, Rocketdyne Division, Rockwell International, 1986
5. Newell, J. F., Kurth, R. E., and Ho, H., "Composite Load Spectra for Select Space Propulsion Structural Components, Final Report," in preparation for NASA Lewis Research Center, Rocketdyne Division, Rockwell International, 1989
6. Newell, J. F., Rajagopal, K. R., and Ho, H., "Probabilistic Structural Analysis of Space Propulsion System Turbine Blade," AIAA 30th Structures, Structural Dynamics and Materials Conference, Mobile, Alabama, April 1989
7. Newell, J. F., Rajagopal, K. R., Ho, H., and Cunniff, J. M., "Probabilistic Structural Analysis of Space Propulsion System LOX Post," submitted to the 31st Structures, Structural Dynamics and Materials Conference, Long Beach, California, April 1990
8. Southwest Research Institute, University of Arizona, Rocketdyne, and J. B. Dias, "Probabilistic Structural Analysis Methods for Select Space Propulsion System Components (PSAM)," NASA Contract NAS3-24389, Annual Reports, 1985, 1986, 1987, 1988
9. Waterman, D. A., "A Guide to Expert Systems," Addison-Wesley, 1985

INTELLIGENT TUTORING SYSTEMS FOR SPACE APPLICATIONS

Carol A. Luckhardt-Redfield, Ph.D.
Southwest Research Institute
6220 Culebra Rd.
San Antonio, Texas 78228-0510

May 1990

ABSTRACT

Artificial Intelligence has been used in many space applications. Intelligent tutoring systems (ITSs) have only recently been developed for assisting training of space operations and skills. In this paper, an ITS at Southwest Research Institute is described as an example of an ITS application for space operations, specifically, training console operations at mission control. A distinction is made between critical skills and knowledge versus routine skills. Other ITSs for space are also discussed and future training requirements and potential ITS solutions are described.

INTRODUCTION

There are many applications of expert systems technology to the space shuttle (27, 32) and Space Station Freedom (7, 8) and of artificial intelligence in general to space systems (10, 11, 12, 16, 21, 25, 26). The following is a representative list of potential and existing space application systems from various publications. Notice that there are three intelligent trainers or tutoring systems in this list.

- Attitude control for spacecraft
- Autonomous flight control for a Mars Balloon
- Autonomous maintenance of platforms
- Autonomous navigation and guidance for rovers
- Autonomous rendezvous and docking for a space vehicle
- Computer vision for automated rovers
- Diagnosing spacecraft problems
- EVA Retriever
- Fault diagnosis of electrical power systems for Space Station
- Flight Telerobotic Servicer
- Hubble Space Telescope planning and scheduling
- Intelligent interface for satellite operations
- Intelligent trainer for shuttle flight controllers

- Intelligent tutoring system for shuttle diagnosis
- Liquid Oxygen Expert System
- Mars Rover Sample Return
- Mission planning
- Mission Telemetry System Monitor
- Radar tracker scheduling for shuttle and satellites
- Satellite scheduling and control
- Scheduling and planning for ground control systems
- Shuttle Main Engine Test Evaluator
- Spacecraft communications configuration optimization
- Spacecraft operations scheduling
- Space Shuttle Payload Integration
- Special Purpose Dexterous Manipulator (Canada — for Station)
- Venus Orbit Planner

An intelligent tutoring system (ITS) is a training system implemented on a computer, as in computer-based training. However, such systems also include some AI-based techniques that allow them to be more adaptive and responsive to a student's needs, making them appear smart. ITSs can be divided into four main modules: an intelligent interface, an instructional expert, a domain expert, and a student model (2, 19). The instruction module can be thought of as the center point which receives information from the other modules as in Figure 1. The domain expert module contains a representation of the knowledge or skill to be trained. It may also be associated with a simulator that allows a student to explore the domain. The student module is similar to the expert module except it usually is an empty shell or structure of the knowledge or skill to be trained. It is then filled with what the student learns and knows with respect to the domain based on the interactions that student has with the ITS. The instructional module is a representation of the knowledge and skill of teaching the subject matter of the expert module. The intelligent interface is the method of communication between the student and the ITS. An interface is intelligent in what is communicated to the student based on the student's response and history with the system. Knowledge-based system techniques can be used in any of these four modules in order to implement the "intelligence."

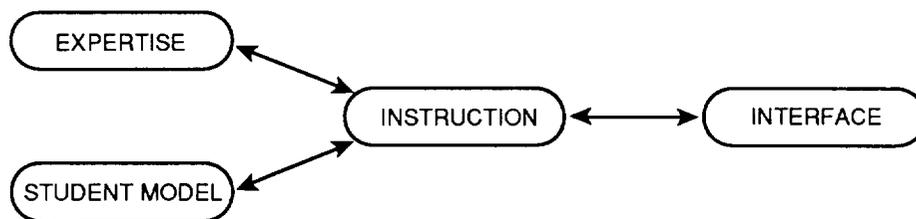


Figure 1. ITS Structure

As life becomes more complex and faster paced, there will be an increased demand for readily available, individualized, effective teaching and training sources. For example, on the average, the next generation of workers will change jobs more than five times and 95% of all jobs will involve information handling that will be highly computerized (3). At Johnson Space Center, the training

for Space Shuttle missions was beginning to fall behind schedule before the Challenger accident in 1986. As NASA gears up to full capability of Shuttle flights, the demand on training will be resumed. Much of the training at NASA is one-on-one, human-to-human instruction with full-scale simulations. The training time that requires human instructors needs to be reduced. ITSs can fulfill some of the new training requirements, inside and outside of NASA, so that human teachers can spend more time on personalizing instruction for the more exceptional cases.

THE PRESENT

There are ITSs for a multitude of applications (13, 20, 31) including aircraft (1, 18, 23, 24) and space systems (4, 5, 14, 22, 30). However, ITSs are still very much in the research stage and used on a small scale. They are only beginning to be available to large groups of students (29).

Most of the areas of application for ITSs that already exist or that are being researched for space systems are in the environment of spacecraft command and control. For example, at Southwest Research Institute an ITS for training how to operate a console in the Mission Control Center at NASA's Johnson Space Center is under development (6). Currently, the ITS trains the operations for the Manual Select Keyboard (MSK). The MSK is used for initializing the console for the different phases of any mission such as ascent, orbit, and descent of the space shuttle. The initialization includes skills of formatting light panels and selecting video displays. Although the tutor would be useful in training new flight controllers in the use of the console, its main use is for research into the effectiveness of ITSs in the training of high performance skills. The training of high performance skills is a major issue with which NASA must constantly contend.

The console tutor runs on an Apollo Domain 4000 with a color monitor and is written in C, CLIPS, and GPR. The tutor provides a low-physical-fidelity, high-cognitive-fidelity training environment. Figure 2 shows what the student sees when interfacing with the tutor. The top half of the screen is a depiction of a control center console. Each of the sections or components on the console can be selected in the figure by clicking the mouse when the pointer is on the desired component. The component then appears on a larger scale in the bottom left-hand portion of the screen. In the figure, the MSK has been selected. The bottom right-hand side of the screen contains the text for information, instructions and feedback. The text that appears in the figure is information for training an overview of the system. The console tutor trains five levels of familiarity with the MSK: an overview of the MSK components, an overview of the procedure, example exercises for demonstration and accuracy training, example exercises for speed training, and exercises for automatizing the process. The tutor has been built with the purpose of training the MSK operation to the point where the operator can do the procedure automatically. This means that the console operator will be able to perform MSK manipulation while processing something else such as holding a conversation or responding to other auditory inputs. For training such automaticity in a skill, the tutor must provide a secondary task for the student to perform while he or she is performing the primary task of MSK manipulation. In this case, the secondary task is the recognition of a pattern of beeps and the response of hitting the corresponding function key. Advancement through the tutoring system and "graduation" depend upon performance accuracy and speed in some cases. The tutor will remediate and even return to the beginning of the training if required, based on the student's performance.

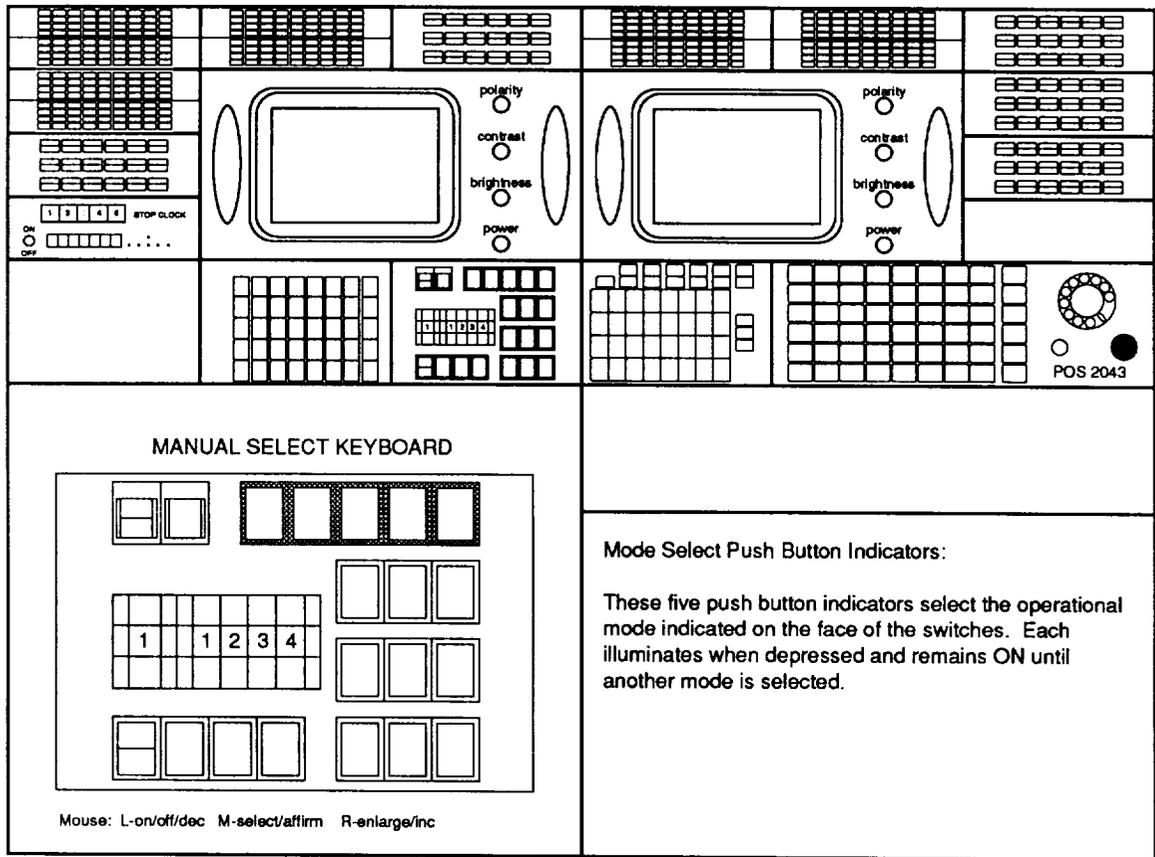


Figure 2. Graphical Interface of the console Tutor

The console tutor can be expanded to include all the operations of a console in mission control. Figure 3 shows a structure that the tutor can use in order to train all of the components and procedures required for console operations (5). This tutor would be a useful tool for the current console operators and training to become an operator. Those who have already become proficient on the console could use the system to refresh their skills or improve their speed. Currently, training new operators consists of many hours of one-on-one, on-the-job type instruction. Trainers of the console operators are in high demand. Some of the expensive one-on-one and full simulation time could be saved with a tutoring system such as the console operations tutor.

There are other ITSs for space application domains, such as the OM (orbital mechanics) tutor, and MITT for maintaining the space shuttle fuel cells (22). ITSSO (intelligent tutoring system for satellite operations) is an embedded training system for operators of ground control systems for near-earth unmanned scientific satellites (30). A prototype ITS is under development to assist in teaching the command and control language STOL (systems test and operations language) at NASA (4). The Payload-Assist Module Deploys/Intelligent Computer-Aiding Training system trains mission control center flight dynamics officers to deploy a satellite from the shuttle (14).

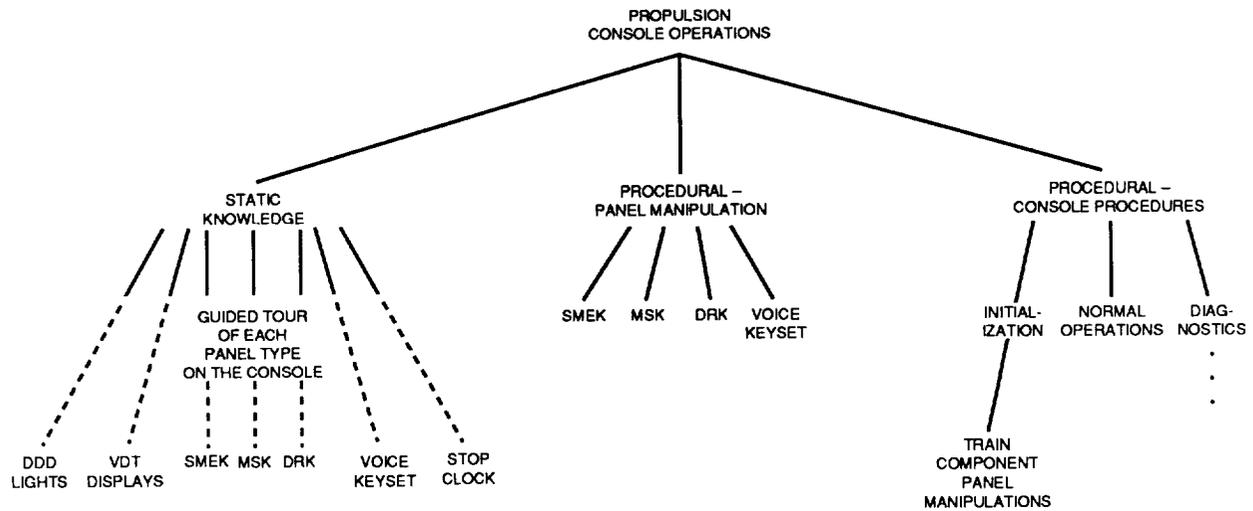


Figure 3. Top Level Overview of the Knowledge Needed to Train Console Operations

THE FUTURE

We expect to have a space station before the turn of the century, a lunar base shortly thereafter, and a manned mission to Mars after that. The President of the United States confirmed this in his speech of July 20, 1989, the 20th Anniversary of humanity first stepping onto the moon. The shuttle is the key to the station and the rest of our accomplishments in space right now. With the aggressive shuttle flight schedule, training for missions will become overloaded using the current training methodologies that require extensive time, one-on-one training, and large, three-dimensional, full-fidelity simulators. Future training will need to be consolidated and made more effective while using less human and full-fidelity simulation intensive techniques. ITSs could be used to assist in training for many aspects of space related endeavors by lowering requirements for so much human intervention.

For example, the instructors to the astronauts work very closely with the astronauts many months before a mission. The astronauts train in the shuttle mission simulator in order to have experience with many different shuttle flight scenarios. The instructors using this system are inundated with data and information during a simulation. An ITS could be utilized to process the data and decide what is important and pertinent for the instructors' use in their decisions of what malfunctions to introduce into the simulation for the most effective training of the astronauts undergoing the training. The ITS could serve to train the instructors as well. The ITS could be written to be an intelligent interface for the instructors in viewing the current status of the simulation and to input malfunctions. It could be an advisor of malfunctions to help achieve a session's objectives, and it could be a record keeper for reporting the results of a session and tracking training objectives.

A payload specialist aboard the shuttle will use equipment and run experiments, all in an unfamiliar, weightless environment. ITSs can be coupled with simulations to assist in determining curricula, tracking and analyzing student performance, running individualized instruction and

scenarios, and making recommendations to students. Finally, ITSs would be useful for embedded training in addition to training and retraining specialized equipment usage and repair.

One of the recommended applications of knowledge-based systems for the space station is onboard personnel training (9). In the Lunar Evolution Case Study of the Office of Exploration's Study Requirements Document, there is a requirement that autonomous, on-site crew training be available for all crew operated, safety critical systems (17). ITSs can fulfill these needs. The people who live and work on the station will have many skills and much knowledge. When the skills and knowledge are not utilized, they degrade over time. ITSs can be used for keeping skills and knowledge intact and fully accessible to the people staying on the station. A lunar city, long term voyage in space, and a Mars base have similar problems with respect to knowledge degradation from lack of use and would also benefit from ITSs in this way.

Skills and knowledge areas for living in and running space systems could be split into different categories of training such as those that are routine and those that are used in emergency situations. Routine tasks such as equipment maintenance and repair, facility cleaning, and food management may need to be handled in a different manner than tasks that are needed for life threatening circumstances such as structural or propulsion breakdowns. Paper, on-line manuals, or conventional computer-based training may be best for the routine tasks where time can be taken by the learner. On the other hand, an ITS with an interactive videodisc interface could be best suited for emergency-type tasks that need to be learned and used immediately. The ITS could be geared to the particular learner and the specific, required task and repair, so that the training can occur quickly. Having ITSs accessible could make an important difference in critical situations.

One place to look for ITS applications is where expert systems have been built, such as the system for failure diagnosis on the space station (15). It is possible to build an ITS around an expert system (28). Smith, Fink, and Lusth present an approach to taking a specific expert system design and developing a tutoring system around it. The expert system design is called the Integrated Diagnostic Model where information about the domain is represented in two levels, the experiential and the functional. The experiential level contains information gained from the experience of working in the domain. The functional level contains information about the operation of the domain and how the physical devices work. The experiential part can be used as a tutor that can impart the overall behavior of a system to the student. The functional model can be used to teach a deeper, more detailed knowledge of a system. An instructional designer can choose between the two representations as is appropriate and most effective for each skill to be taught.

CONCLUSION

Intelligent tutoring systems need to continue to be developed and improved in order to come up to speed for future requirements, especially for space systems. NASA and other future-looking organizations should be very interested in ITS development for space applications and other training needs from manipulating and diagnosing complex systems to teaching language and computer literacy.

REFERENCES

1. Biegel, Dr. J.E., "An Intelligent Simulation Training System", Third Annual Workshop on Space Operations Automation and Robotics, Abstracts for Technical Sessions, Houston, TX, July 1989
2. Burns, H. and Parlett, J., "The Dimensions of Tomorrow's Training Vision: On Knowledge Architectures for Intelligent Tutoring Systems", Proceedings of the 2nd Intelligent Tutoring Systems Research Forum, San Antonio, TX, April 1989
3. Cetron, M.J., "Class of 2000: The Good News and The Bad News", The Futurist, Vol. 21, No. 6, pp. 9-15, 1988
4. Elke, D., Seamster, T., and Truszkowski, W., "Functional Description of a Command and Control Language Tutor", Third Annual Workshop on Space Operations Automation and Robotics, Abstracts for Technical Sessions, Houston, TX, July 1989
5. Fink, P.K., "Issues in Representing Knowledge for Training High Performance Skills," Proceedings of the 2nd Intelligent Tutoring Systems Research Forum, San Antonio, TX, April 1989
6. Fink, P.K., "The Role of Domain Knowledge in the Design of An Intelligent Tutoring System", Intelligent Tutoring Systems: Evolutions of Design, Lawrence Erlbaum, NJ, 1990
7. Firschein, O. et al, Artificial Intelligence for Space Station Automation, Noyes Publications, NJ, 1986
8. Friedland, P., Swietek, G., and Bullock, B., "A Study of Knowledge-Based Systems for the Space Station", AIAA 27th Aerospace Sciences Meetings, Reno, NV, January 1989
9. Greeley, R. and Williams, R. (Editors), Experiments in Planetary and Related Sciences and the Space Station, NASA Conference Publications 2494, Tempe, AZ, September, 1986
10. Griffin, S. (Editor), Second Annual Workshop on Space Operations Automation and Robotics, SOAR 1988, Proceedings, NASA Conference Publication 3019, Dayton, OH, July 1988
11. Heer, E. and Lum H., "Progress in Astronautics and Aeronautics", Machine Intelligence and Autonomy for Aerospace Systems, Vol. 115, American Institute of Aeronautics, Inc., NJ, 1988
12. Johnson, J.R., in Proceedings of the Fourth Annual Aerospace Applications of Artificial Intelligence Conference, Dayton, OH, 1988
13. Kearsley, G., Artificial Intelligence and Instruction, Addison-Wesley, MA, 1987
14. Loftin, R.B., Wang, L., Baffes, P., Hua, G., "An Intelligent Training System for Space Shuttle Flight Controllers", Workshop on Space Operations Automation and Robotics, Dayton, OH, July 1988 Page 7

15. Marsh, C., "The ISA Expert System - A Prototype System for Failure Diagnosis on the Space Station," International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Tullahoma, TN, June 1988
16. NASA, JSC, and USAF, Third Annual Workshop on Space Operations Automation and Robotics, Abstracts for Technical Sessions, Houston, TX, July 1989
17. Office of Exploration, Study Requirements Document, NASA Johnson Space Center, Houston, TX, 1989
18. Parker, E., "Success in Tutoring Electronic Troubleshooting", Third Annual Workshop on Space Operations Automation and Robotics, Abstracts for Technical Sessions, Houston, TX, July 1989
19. Polson, M.C. and Richardson, J.J. (Editors), Foundations of Intelligent Tutoring Systems, NJ, Lawrence Erlbaum Associates, 1988
20. Psozka, J., Massey, L.D., and Mutter, S.A., Intelligent Tutoring Systems: Lessons Learned, Lawrence Erlbaum Associates, NJ, 1988
21. Rash, J. and Hughes, P. (Editors), Proceedings of 1988 Goddard Conference on Space Applications of Artificial Intelligence, NASA Conference Publication 3009, Greenbelt, MD, May 1988
22. Regian, J.W., "Intelligent Tutoring Systems Research in the Training Systems Division: Space Applications", Workshop on Space Operations Automation and Robotics, Dayton, OH, July 1988
23. Regian, J.W., "Representing and Teaching High-Performance Tasks within Intelligent Tutoring Systems," Proceedings of the 2nd Intelligent Tutoring Systems Research Forum, San Antonio, TX, April 1989
24. Regian, J.W., "Training High-Performance Tasks with Intelligent Tutoring Systems", Third Annual Workshop on Space Operations Automation and Robotics, Abstracts for Technical Sessions, Houston, TX, July 1989
25. Rouse, D.M. and Hummel, T.C., "The Pilot's Associate: Enhancing Situational Awareness Through Cooperating Expert Systems", Aerospace Technology Conference and Exposition, Long Beach, CA, October 1987
26. Scientific and Technical Information Branch of NASA and University of Alabama, Proceedings of the Fourth Conference on Artificial Intelligence for Space Applications, NASA Conference Publication 3013, Huntsville, AL, November 1988
27. Shankar, K., "Shuttle Artificial Intelligence Applications", AIAA 27th Aerospace Sciences Meeting, Reno, NV, January 1989 Page 8

28. Smith, H.R., Fink, P.K., Luth, J.C., "Intelligent Tutoring Using the Integrated Diagnostic Model: An Expert System for Diagnosis and Repair", IEEE Conference on Expert Systems in Government Symposium, Washington, DC, October 1985
29. Swigger, K., "Managing Communication Knowledge", Proceedings of the 2nd Intelligent Tutoring Systems Research Forum, San Antonio, TX, April 1989
30. Truskowski, W.F., "Intelligent Tutoring in the Spacecraft Command/Control Environment", Workshop on Space Operations Automation and Robotics, Dayton, OH, July 1988
31. Wenger, E., Artificial Intelligence and Tutoring Systems, Morgan Kaufmann, CA, 1987
32. Wood, K., Hanly, L.G., Lawson, B., Randall, C., Turner, R., and Wang, C., "Shuttle Failure Detection", Aerospace America, July 1989



DOCUMENTATION AND KNOWLEDGE ACQUISITION

Daniel Rochowiak — Johnson Research Center
Warren Moseley — Computer Science
University of Alabama in Huntsville

ABSTRACT

Traditional approaches to knowledge acquisition have focused on interviews. An alternative focuses on the documentation associated with a domain. Adopting a documentation approach provides some advantages during familiarization. A Knowledge Management Tool has been constructed to gain these advantages.

INTRODUCTION

The familiarization aspect of knowledge acquisition (KA) continues from the beginning of a knowledge based programming project until the final content for the project is determined. Familiarization, in this sense, is not a distinct episode of knowledge engineering, but consists in all those activities which the knowledge engineer (KE) engages to prepare for the project and to prepare for particular knowledge acquisition sessions. It is important to note that these preparatory activities are both important and time consuming. They are important since they lay the ground for shared common content about the domain, and are time consuming since the knowledge engineer is required to become acquainted with terms, concepts, methods, and theories that may be far different from those with which he or she has already become familiar. In the familiarization process at the beginning of the project the knowledge engineer attempts to find the sources of important information, organize that information, read the documents, charts and other materials that have been assembled, and gain an elementary mastery of the vocabulary of the domain. As the project progresses the KE will continually need to become familiar with new material. However, the familiarization process for knowledge acquisition sessions based on this new material ideally should be less time consuming since a base has been established by previous efforts.

Familiarization is document-driven. Documents play a primary role even when there is a mentor to guide the KE through the material. The documents become a base on which KA can proceed. There are two reasons for this. (9) The first is practical. In order to interact effectively with an expert, the KE and the expert must have some shared conception of the domain. The shared conception is not to be understood as a detailed, precise, accurate or comprehensive account of the domain. Rather the shared account is the base that will continue to develop in the KA process. If the interaction with the expert requires that there be some common understanding, then it should be clear that in the beginning this must be provided in a way other than the interview process. In general, the information needed to establish this shared level of understanding is contained in documents associated with the expert's domain. The second reason is structural. Organizations collect knowledge in documents. These documents represent the stored knowledge of the organization. As such, the knowledge in these documents is social and intersubjective, and constitutes the background against which both individual knowledge and expertise are defined. Thus, for practical and organizational reasons the familiarization aspect of KA is document-driven.

The focus on documentation generates advantages.

- Documents are often "approved" knowledge sources.
- The writers of the documents have "decompiled" to some degree the domain knowledge.
- Documents tie down references in the knowledge dictionary.

- Documents make multiple lines of reasoning available.
- The documents provide a context needed to gain access to specific expertise.
- The documents provide a source of material for both explanation and help facilities.
- Attention to the documentation leads to the identification of weaknesses in the documents.
- Attention to the documents provides a point of reference for the expert and the user.
- Attention to documents leads to tighter coupling and resource-sharing between KE and technical writer.

Methods and aids must be devised to gain these advantages, however.(4) There are at least two ways in which such methods and aids might be built. The first focuses on the direct analysis of existing documents. The objective of this way is to create tools that would directly analyze documents and abstract knowledge. The second way focuses on the management of the familiarization process associated with the documents. We have adopted the latter way. The methods and aids that we are developing focus on the idea of a knowledge dictionary that is similar to the idea of a data dictionary in traditional database operations, and the expanded model of reasoning articulated by Toulmin, Rieke, and Janik (10).

A DOCUMENTATION APPROACH TO KNOWLEDGE ACQUISITION

Traditional approaches to knowledge engineering emphasize the interview process. Interview driven methods assume that interviewing an expert is the best way to acquire knowledge that is “chunked” and “compiled”. Knowledge is “chunked” when items of knowledge are organized into meaningful units. Such chunking is believed to increase the performance of human experts. Unfortunately, such chunking makes knowledge acquisition more difficult especially when such chunks are “compiled”. “Compiled” knowledge is knowledge that has been distilled and abstracted of all unnecessary elements; elements which may have originally been needed to gain the knowledge are removed. Further, the organizational schemas may be altered to increase the efficiency of recalling the items in the chunks. Compiled chunks may account for the fact that experts recall all of the content of one chunk before processing a subsequent chunk.

Knowledge engineers are familiar with the problems of chunked and compiled knowledge, and have developed various techniques for acquiring various kinds of knowledge. The documentation approach is consistent with the assumption that knowledge is chunked and compiled, and adds to the available techniques, especially those available during the familiarization activities of knowledge acquisition. Such familiarization activities are part of the episodic units in knowledge acquisition. The episodic units of knowledge acquisition include preparing, conducting, and reviewing interviews. The preparation activities which include familiarization are most intensive during the initial phase of a project. Although it is difficult to obtain data on this topic, informed, but informal, estimates suggest that during the initial phase of a project the ratio of preparation time to session is as high as 8 to 1, while over the life of the project the ratio might be closer to 3.5 to 1. (3) In either case it is clear that a significant portion of a knowledge engineer’s time is spent in preparation activities and that such activities are more time consuming during the beginning of the project.

Preparation during the initial phase of a project is a complex undertaking. The knowledge engineer’s activities are geared to becoming familiar with the domain. But how does one become familiar with a domain and in what does that familiarity consist? Our suggestion is that the KE becomes familiar with the domain through documents and that this familiarity leads to the production of a knowledge dictionary.

During preparation, the KE attempts to amass documents about the domain. Such documents include text books, reports, instructional materials, design plans, and, in general, any written (hard copy or electronic) materials about the domain. In using the documentation approach, it is assumed that documents have a degree of authority for the experts in the domain, that the experts in the

domain would recognize the authority of the documents, and that documents are written and revised in order to establish common understandings and frameworks. We do not assume that there is any direct correspondence between the chunks and terms identified in the documents and those used by domain experts. We do assume, however, that the documents act as constraints on the domain expert. In this sense, the documents constitute an official and authoritative framework within which the expert brings his or her skill to bear. These documents act as a backdrop for two important KEing tasks: building a knowledge dictionary and analyzing it.

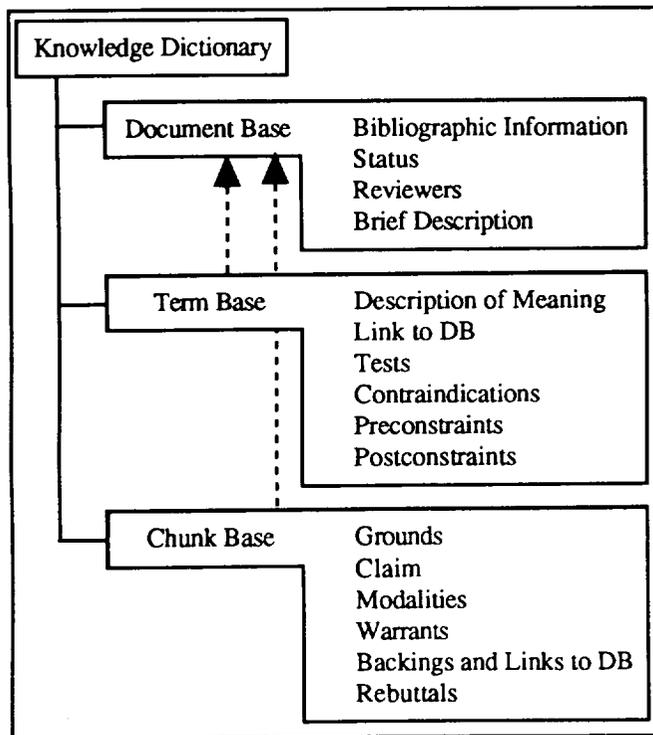


Figure 1

A knowledge dictionary, in its initial formulation, consists of a document base, a term base, and a chunk base. (See Figure 1) These three bases provide a map of the domain and a “first pass” collection of materials for automation.

The document base consists of bibliographic material, the status of the document, indications of whether and by whom a document has been reviewed, and a brief general description of the content and utility of the document. The materials in the term and chunk bases are keyed to these document base. Since in many cases the documents undergo revision as the project evolves, keying the terms and chunks to the document base provides a way of systematically reviewing the materials in the knowledge dictionary in light of revised documentation.

The term base provides information about the meaning and application of the term. An entry for a term provides a brief

ordinary language description of the term and any appropriate abbreviation or symbol for it. Additionally, an entry contains typical information about the values the term may take, tests associate with the term, contraindications for the application of the term, and pre and post constraints on the application of the term. The specification of the source for the information provides a link to the documents base.

Knowledge in the chunk base is represented using the Toulmin, Rieke, and Janik (TRJ) model of reasoning. (See Figure 2). Using this model a knowledge chunk is treated as an argumentative or inferential structure. However the model allows for greater depth and flexibility than more strictly logical models. When working with documents, one notices the flexibility of arguments. Even in highly technical areas, assumptions and premises are often not made explicit. (6) Further, the use of

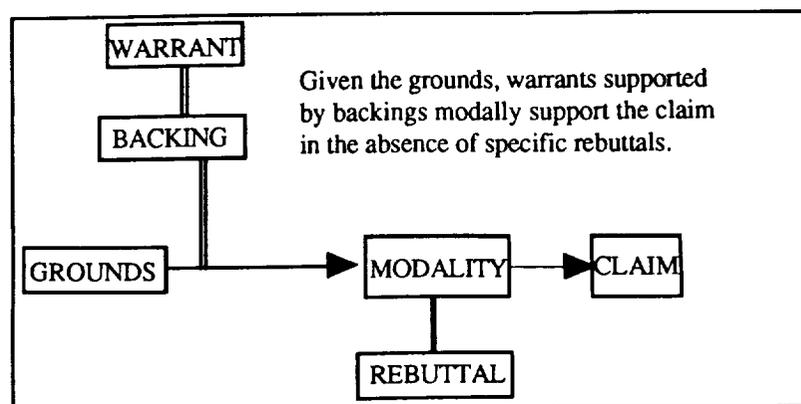


Figure 2

language makes it possible to link various knowledge elements in subtle, but important ways. The TRJ model more closely represents this sort reasoning.

In the TRJ model a claim is analogous to a conclusion in a deductive model or the the facts that are added to the knowledge base after a rule is fired in a rule based system. The grounds are analogous to premises or the facts in the knowledge base before a rule is fired. The modality can be thought of as a confidence value or some other measure of the strength of the claim based on the grounds. The warrant is most often the conditional statement that allows the grounds to lead to the claim. The warrants may be expressed as a rule, but other representations are possible. The backing indicates the support or basis for the warrant. The rebuttal indicates the considerations that would inhibit or prevent the assertion of the claim.

The TRJ model of reasoning is much more flexible than traditional models that emphasize the logical (propositional and predicate) style of representation. First, it should be noted that a warrant might have multiple backings. If this is so, then the removal of any one of the backings is not sufficient for retracting a warrant. This suggests that a modal logic might be applied to reasonings about warrants. This has been explored in Rochowiak (5). There a very minimal modal system, T, showed promise. Second, the use of the rebuttal notion may prove valuable in nonmonotonic reasonings. For example, if the reasoning unit were implemented in a frame like structure, then the rebuttal slot could be used as a trigger for retracting the rule's application and the retraction of the facts asserted in the claim. Or, it could be used to prohibit the application of a rule that would otherwise match a pattern in the facts. Third, it should be noticed that the notion of a backing provides a very natural way to include references to documents and can be easily extended to include the statements of experts in interview situations. Finally, the availability of backings for warrants (rules) allows for a clearer separation of the system and domain oriented notions of explanation . (7,8).

Given a knowledge dictionary composed of the bases specified above how is it to be analyzed? This question can begin to be given an answer by specifying the sorts of operations that a KE would want to have performed on the bases.

Beginning with the simplest case the KE should be advised of possible alteration sites when a document is updated, revised, deleted, or in some way altered. In an effort to build an essentially bureaucratic system this would be of great importance. A bureaucratic system is one that attempts to automate some process in a bureaucracy. The administration of loans and the purchasing of parts are typical examples. In these cases new rules or forms may require an alteration in either the term or the chunk bases. Another important arena is that in which the KE activity is occurring while the domain is being constructed. In this arena changes to the domain in terms of designs or specifications may force changes in the dictionary. An operation for alerting the KE to potential changes is needed for the analysis of the dictionary. A more complex case involves the grouping and reporting of the materials in the dictionary. Operations that would provide reports of how terms, chunks, and documents are linked, as well as the frequency of such linking, would help the KE to better understand how the knowledge is clustered. At another level operations that would identify some gaps or sites for decomposition are desirable. Such operations might begin with the identification of terms used in the chunks that are not defined or the identification of missing elements in the definition of the term. The identification of empty elements in the chunks would be equally important. Additional operations would be desirable for allowing multiple views of the knowledge dictionary, tracing of particular elements (say particular backings or typical tests) through the knowledge dictionary, and identifying links between chunks (grounds to claims, claims to grounds). Finally in keeping with the spirit of the documentation approach, there should be operations that would generate relevant sections of the knowledge dictionary as a document. Such documents would be useful in creating reports, setting agendas for interviews, and constructing materials for interviews. This is not, of course, an exhaustive account of the sorts of operations that a KE might need in analyzing the knowledge dictionary, but it is indicative of the

kinds of concerns that are relevant in the construction of a tool for generating and analyzing a knowledge dictionary.

The process of analyzing and updating the knowledge dictionary is one that will continue over the life of the project. Ideally, the final dictionary would contain all of the information required to document the knowledge aspect of the finished system. For example, the coding that implements a chunk should be tied to the dictionary. This sort of tie would facilitate the identification of the locations in the code that need to be update as a result of some change in the domain knowledge.

A TOOL FOR THE DOCUMENTATION APPROACH

A tool that implements the documentation approach to knowledge acquisition is being developed. "Knowledge Management Tools" (KMT) is constructed primarily in HyperTalk™ and CLIPS for the Macintosh™ family of computers. The use of a hypertext system is desirable since the hypertext facilities are of themselves useful in KA activities. (1,11). CLIPS is being used since it provides a readily available inference system.

The associational character of the construction and analysis of the knowledge dictionary strongly suggests that a hypertext system is appropriate. During familiarization the entry of data is not strongly structured. The KE may obtain information on one topic and then another without there being a clear connection between the units of information. However, as more information is entered into the dictionary it is reasonable to think that patterns will emerge. These patterns can be quickly and easily captured in associational links. Such links can provide a map of the material in the dictionary and represents the KE's view of the structure of the domain. Further, in familiarization the KE may become aware of new elements that should be added to the dictionary only in the process of reviewing information already entered. This again suggests that an associational link should be created that will allow the KE to easily add the needed information. Finally when there is more than one KE it will often be necessary to review what another KE has done. This review is again an associational link. Each of these reasons suggests the desirability of using a hypertext approach to the management of the familiarization process.

From a management point of view the knowledge dictionary can be treated as a (nonlinear) text. The production of the text should be such that a KE or a member of a KE team can add additional text to an entry during review. This factor means that the text in the knowledge dictionary not only can serve as the background against which a KE formulates interview sessions, but also is a means of communication for members of a KA team. The knowledge dictionary, in this sense, serves as a shared, common background for further knowledge acquisition. These management features again suggest that a hypertext approach is appropriate.

The inclusion of CLIPS in KMT is both an illustrative and cautionary tale. The inclusion of CLIPS was motivated by practical considerations. CLIPS is readily available, and some projects needed to use CLIPS. Further, since reading CLIPS code can be difficult, the direct association of the CLIPS code and the text in the knowledge dictionary would seem desirable. That is, the material in the knowledge dictionary would indicate what a segment of CLIPS code was intended to represent. On the other hand, this approach leads to an effort to coerce the information and knowledge into CLIPS form. This coercion while having some practical advantages leads to difficulties. Most importantly, rather than trying to capture knowledge and information as would seem to be natural, an effort is made to capture knowledge and information in a way that is amenable to CLIPS. It is almost as if an assumption is made that CLIPS is the appropriate tool for the domain. This difficulty is a general one. The problem can be put clearly in the following way: Should the selection of the tool be a determinate of the KA process, or should the KA process be a determinate of the tool? This essay will not attempt to resolve this difficulty, but it should be noted as a serious one.

KMT-CLIPS includes a K-edit stack, a K-dictionary stack, and a K-document stack. Separation of these three stacks adds to the efficiency of the system. The links between the stacks are in the beginning directed toward the K-document. The system as a whole attempts to keep a list of the associations. As the dictionary develops other links are established. Lists of these associations are also kept by the system. Internally, KMT is a collection of associated lists of association links. Access to the text information stored in the system is provided in various ways be accessing these lists.

The key stack is the K-edit stack. Currently this stack contains four screens. Future screens for analysis are planned. The idea of the K-edit stack is to allow the user to enter knowledge based elements and later provide the CLIPS code. However, this is not required and all materials can be entered at one time. Additionally, CLIPS is interactively available. The K-edit stack implements the idea of a chunk base only in a partial way in its rule card. Backings for the warrants are limited to References. Further since CLIPS is a rule based inference system the grounds and claims components of the TRJ model are identified as Conditions and Actions. The idea of a term base is also only partially implemented in the parameter card. The parameter card does not contain fields for all of the features identified above.

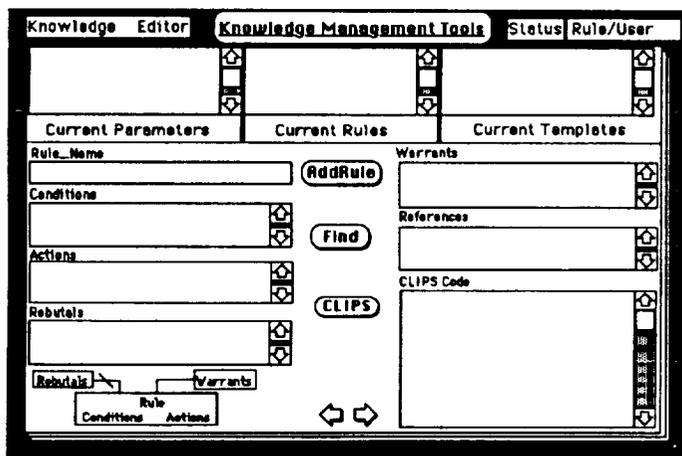


Figure 3

The first card of K-edit is a rule card that contains a chunk template. (See Figure 3) The user provides the name of the rule, its conditions, and actions as ordinary text. Rebuttals are specific circumstances that would prevent the application of the rule. Warrants are the reasons why the rule is being asserted. Both of these are ordinary text, as is the field for references. The diagram in the lower left illustrates the structure. At the top of this card and every card is a list of the currently known parameters, rules, and templates. This provides an interactive access to other parts of the knowledge dictionary. The entries for the

term base are found in the Current Parameter and Current Template list. These identifications were selected to provide a more CLIPS-like interface.

The field at the lower right is used for CLIPS code. This may be added or not at the time the chunk is entered. Additionally, it can be tested by clicking the CLIPS button. This button will add the CLIPS code to a user specified text file, and additionally, if desired, load CLIPS and place that file in a buffer. The buffer can then be compiled and run. On exiting CLIPS, the user will be returned to the stack with the clipboard in tact. Thus, if modifications are made to the CLIPS code in the CLIPS environment, those changes can be copied and pasted into the card. It might be handy to have the new rule load into a buffer and then load the previous rules or templates into another buffer. The KE can then paste the new rule into the old CLIPS code and test it. When it is the way the KE wants it, it can be saved as a CLIPS code file. If this approach is taken the KE will need to clean up the dictionary at a latter date. By treating the CLIPS code as a document and building tools that understand CLIPS code, cleaning up the dictionary will be much easier. We are currently developing such tools.

The AddRule button adds the rule elements to a database indexed by the name. The Find button allows the user to find previous elements in the different bases. By selecting an element from those currently known and clicking Find, the user is taken to the database element in K-dictionary. Clicking the Return button in K-dictionary will return to the current card.

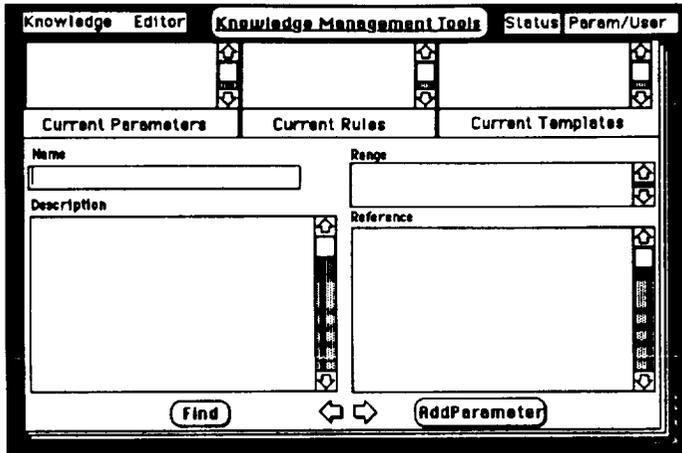


Figure 4

The parameter card partially implements the idea of a term base and functions in a way similar to the rule card. (See Figure 4) The parameters are used to identify the terms in the dictionary. Currently, only the description of the meaning of the term, the range of values, and the reference for the term are included. Fields for additional elements can be added. The AddParameter button adds the data in the fields to the database indexed by name in K-dictionary. Find will find the selected item as in the case of the chunk cards.

The template card is again similar to the rule card. (See Figure 5) The idea of a template is needed in order to make the general ideas of the documentation approach amenable to the latest version of CLIPS (4.3). A template is a structure that is somewhat similar to a frame and allows for more flexible access to and modification of facts. The template card also provides access to CLIPS since CLIPS code is used to define templates. It should be noted that the file to which the template is saved will load into the CLIPS buffer. If there is any additional editing to be done, it can be done there. The AddTemplate button adds the field to the database in K-dictionary indexed by name. Find will find the selected dictionary item

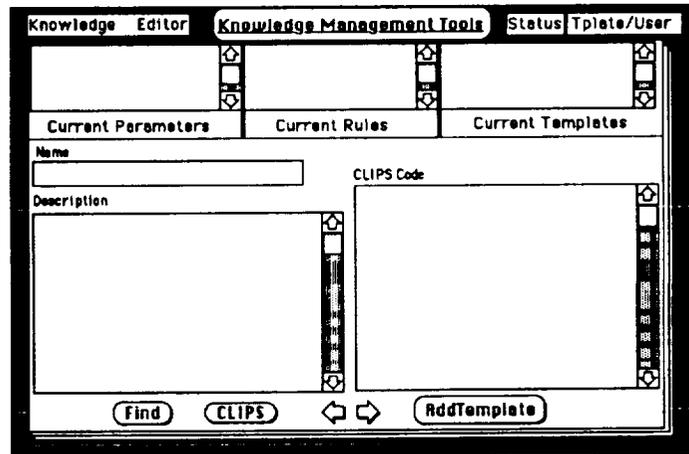


Figure 5

Currently work is under way to provide more of the features of the documentation approach and to generalize KMT. While there are practical reasons for orienting the system toward a specific inferencing mechanism, that selection also brings problems. The focus on rules and the need for a specific template card are examples. From the beginning, the KE is thinking in terms of the concepts and structures that will ultimately be used in the encoding of the knowledge rather than the knowledge itself. In an ideal case, the software should conform to the knowledge, rather than the knowledge conforming to the software. In improving KMT a more general approach will be taken.

The generalization of KMT will allow for alternative ways of entering information and provide a greater integration with tools that can be used in the interview process. Treating KMT as a "poor man's" knowledge acquisition tool, provides a way of adding different strategies. (4) Of particular interest are the additional representational strategies found in BDM-Kat and MacKat. (2)

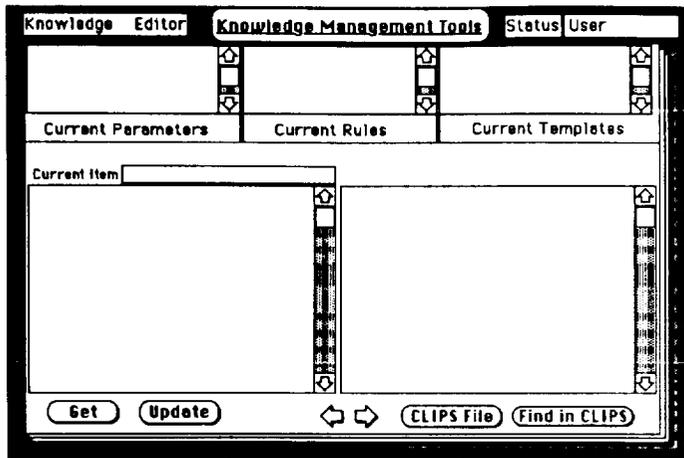


Figure 6

procedures allow the KE to update the CLIPS code.

While the compare card performs several useful functions, there is much more that it should be able to do. Currently several features are being added that improve on the analysis capabilities of KMT and make greater use of inferencing about the material in the cards. For example, scanning the materials in either of the two fields should be able to produce a list of common items, and a list of items contained in the CLIPS code but not in the database. This would alert the user to check the common elements and to determine if new entries are needed for the elements in the CLIPS code that do not match terms in the dictionary.

K-dictionary and K-document currently share the same structure. The fields on the cards and the operations available are, however, easily tailored to specific needs.

The main card for the two stacks controls the operations of the stack. (See Figure 7) The buttons along the bottom of the card allow the KE to enter or alter the material in the stack rather freely. The New Term Button allows the user to enter a new term or document into the appropriate stack and indexes the entry. The Remove Term button removes the term and updates the index. In each case the scrolling list in the Dictionary Entries field is updated and alphabetized. The Write Dict. button allows the KE to build a text file of the materials in the stack. This file can be imported into a word processor, or saved as a separate file that can be loaded by Build Dict. This allows the user to operate with multiple files. The Clear Dict.

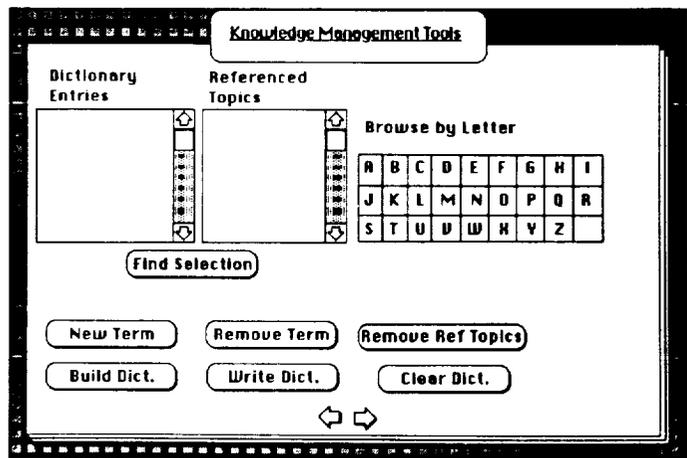


Figure 7

button is used in conjunction with the two previous buttons to initialize the stack and prepare for a new file. The Browse by Letter area allows the user to select a letter and browse the entries for that letter. The scrolling list in the Referenced Topics field operates in two ways. In the first way the KE simply create a list of terms that he or she needs to add to the stack. Selecting one of these terms and clicking the Find Selection button will notify the user if the term already exists. If it does not the KE can then enter the information. The Find Selection button also works in connection with the items in the Dictionary Entries field. The Remove Ref Topics button clears the Referenced Topics field. Items in this field can also be cleared manually.

The compare card is used to generate comparisons between the database and the current CLIPS file. The CLIPS File button allows the KE to select a CLIPS knowledge base. The knowledge base is loaded into the field on the right of the card. Selecting a current parameter, rule, or template and clicking the Get button will load the database item from K-dictionary into the field at the left, place its name into the current item field, and move the CLIPS code to the first occurrence of the item. The items text can then be edited. Copy and paste can be used on the two fields. The Update button updates the database in K-dictionary. Similar

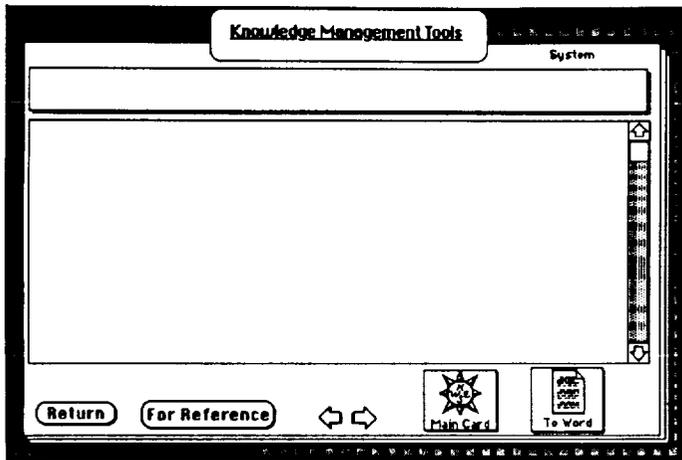


Figure 8

The cards that store the information currently have minimal structure. (See Figure 8) The information can, however, be structured in multiple fields, and resources in the stack allow the information to be gathered in multiple ways. The buttons on the bottom of the card provide a number of functions. The To Word button links to the KE's word processor. The word processor is loaded with the clipboard in tact so that the KE can paste the information into the document. The Main Card button takes the user to the main card of the stack. By highlighting material in the cards entry field and clicking the For Reference

button, the Referenced Topic field of the main card is updated. This allows the KE to scan through a stack and quickly note terms that need definitions. The Return button takes the KE back to the K-edit stack, if this stack was entered through it.

The K-dictionary and K-document stacks currently share the same structure and are only distinguished by their content. Links can be established between the two stacks in several ways. We are currently working on ways to make the linking of the information in the three stacks easier. It should also be noted that the K-dictionary and K-document stack contain resources for formulating a frequency-recency model of the user interaction. This may prove to be helpful when an expert is allowed to view the stack or when tracing the flow of knowledge through a stack.

CONCLUSION

The documentation approach to the management of knowledge acquisition provides a way in which the familiarization aspect of knowledge acquisition can be made more productive. The emphasis on existing documentation, especially in bureaucratic systems or systems in the design phase, can be significant. The KMT tools partially implement the documentation approach. The KMT tools have been used on several projects and have been very useful. It should be remembered that the documents in a bureaucracy have been the traditional repository for its knowledge. The documentation approach is directed toward making use of this repository and augments the classical interview approach to knowledge acquisition.

ACKNOWLEDGEMENTS

This research has been supported by Marshall Space Flight Center, NAS8-36955 / D.O. 50, "Space Station Software Automation Research."

REFERENCES

- (1) Barrett , Edward(ed). *Text, ConText, and HyperText* (Cambridge, Mass.: MIT Press, 1988).
- (2) McGraw, Karen L. "Developing a cognitively-based toolkit for knowledge acquisition," *Workshop on Knowledge Acquisition* (1989) 7-9.
- (3) McGraw, Karen L. and Karan Harbison-Briggs. *Knowledge Acquisition* (Englewood Cliffs: Prentice Hall, 1989).
- (4) Moseley, Warren. Final Report for FAST-1, "Automated Software Tools," 1989.
- (5) Rochowiak, Daniel. "Expertise and reasoning with possibility," *Proceedings of the Second Conference on Artificial Intelligence for Space Applications*, 1987.
- (6) _____ "Extensibility and completeness: an essay on scientific reasoning," *The Journal of Speculative Philosophy* 2 (1988): 241-266.
- (7) _____ "Simple explanation and reasoning," *Proceedings of the AAAI'88 Workshop on Explanation*, 1988, 95-98.
- (8) Rochowiak, Daniel, B. Ragsdale, and L. Wurzelbacher. "An integrated hypertext and rule based system for explanation," *Proceedings of Expert Systems '89*, 1989, 345-352.
- (9) Rochowiak, Daniel, D. Hays, and D. Ford. "Document driven knowledge acquisition in the construction of expert systems for aquaculture," *Proceedings of Expert Systems '89*, 1989, 109-120.
- (10) Toulmin, S., R. Rieke, and A. Janik, *An Introduction to Reasoning* (New York: Macmillan, 1984).
- (11) Wells, Tracy. "Hypertext as a means for knowledge acquisition," *SIGART Newsletter* 108 (April 1989): 136-138.

INDUCTIVE KNOWLEDGE ACQUISITION EXPERIENCE WITH COMMERCIAL
TOOLS FOR SPACE SHUTTLE MAIN ENGINE TESTING

Dr. Kenneth L. Modesitt
Head, Department of Computer Science
Western Kentucky University
Bowling Green, KY 42101

Abstract: Since 1984, an effort has been underway at Rocketdyne, manufacturer of the Space Shuttle Main Engine (SSME), to automate much of the analysis procedure conducted after engine test firings. Previously published articles at national and international conferences have contained the context of and justification for this effort (Refs. 3, 7, 10, 11, 15, 16). Here, progress is reported in building the full system, including the extensions of integrating large databases with the system, known as "Scotty." Inductive knowledge acquisition has proven itself to be a key factor in the success of Scotty. The combination of a powerful inductive expert system building tool (ExTran), a relational data base management system (Reliance), and software engineering principles and Computer-Assisted software Engineering (CASE) tools makes for a practical, useful and state-of-the-art application of an expert system.

INTRODUCTION

Every time a Space Shuttle Main Engine (SSME) is test fired, hundreds of measurements are taken directly from a wide variety of sensors. Many more values are also calculated from these. All of these data values, when combined with previous engine and component performance, are used by the engineering staff at Rocketdyne, the propulsion division of Rockwell International, to determine the future tests. These outcomes can vary from all requirements being met, to a few minor events, to a rare significant event. As the SSME is the world's most complex reusable liquid-fuel (oxygen and hydrogen) rocket engine, Rocketdyne and NASA, the customer, conduct thorough investigations of each test firing by their most highly-trained engineering staff. The author is a former employee of the Rocketdyne division.

To continue its virtually perfect record of supporting shuttle flights, Rocketdyne is always looking for ways, both technical and organizational, to improve the quality of the product while working within customer guidelines. One of the major methods involves making the most accurate diagnosis, analysis, and recommendation possible for the the next engine test or shuttle flight. To perform this task, reliance has been on maximal use of sophisticated tools and the expertise of an engineering staff. This staff has accumulated experience dating back to 1975 and covering 1400+ SSME firings, plus numerous other ones: Apollo F-1, J-2, and Atlas engines.

Rocketdyne was confronted with a significant dilemma: how to improve the quality of the SSME test analysis in the face of

diminishing senior staff. Several options to solve this dilemma were discussed in Ref. 7. It was decided to use a combination of staff, results from previous SSME tests, and automated software tools to build a prototype for automated corporate expertise related to reusable propulsion components.

Rocketdyne was far from alone in being confronted with the above problems. Indeed, the corporation had ample "company" in deciding to use a type of automated tool known as expert systems, part of the artificial intelligence technology. The company is certainly not the first to decide to concentrate initially on a diagnosis type of application, a type currently of considerable importance to industry despite being "old-hat" to the AI research community. So what is unique about Scotty, the name given to the automated system? There are two unusual aspects.

One such aspect is the incorporation of Scotty as "another", albeit advanced, software tool which must:

1. Meet corporate-wide software engineering development and quality guidelines.
2. Live in a distributed corporate environment,
3. Talk to large data bases,
4. Be maintained by existing engineering staff,
5. Execute on standard computers,
6. Be amenable to parallel processing hardware, and
7. Run with color graphics terminals,

The other unusual aspect is a technical one which increases the ease with which Scotty can be constructed. By use of a type of Expert System Building Tool (ESBT) known as inductive or example-based, the historical expertise now reposing in data bases, both in human and machine form, from the hundreds of SSME tests can be transformed into examples, and thence automatically into rules. These rules will, in turn, drive Scotty during normal day-to-day operation in future years.

Scotty: HISTORY

In 1984, the author was hired by Rocketdyne to assist in the construction of an automated tool for SSME test analysis. The employment was on a half-time basis, and was in addition to his position as Professor of Computer Science at California State University, Northridge. Within two months, a proof-of-concept model for a High Pressure Oxidizer Turbo Pump (HPOTP) had been built. This involved recommendation of an inductive ESBT, Expert Ease by Intelligent Terminals, Ltd (ITL), now known as KnowledgeLink, in Glasgow, Scotland, and the first such PC-based ESBT commercially available. The tool was purchased and used, after minimal training time, by a mechanical engineer, to diagnose HPOTP anomalies, by specifying 42 examples and nine attributes. A 48 rule subsystem was automatically generated by Expert Ease. No rules were required of the engineer. This prototype and the problem context, rationale, and solution were described in an early paper (Ref. 7). A desirable tentative system configuration is shown in Figure 1.

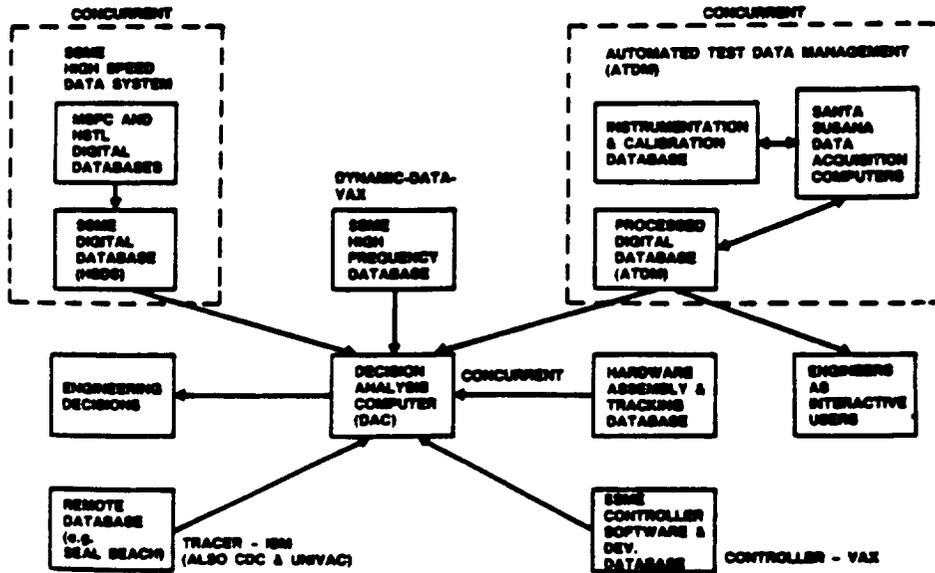


Figure 1. Scotty - Final System Configuration

During 1985 and 1986, the system (now named Scotty) underwent several extensions. From a tool viewpoint, a more powerful ESBT became available. ExTran 7, an industrial strength Fortran-based inductive ESBT from ITL which runs on a wide variety of machines from PCs to workstations to super-minis to mainframes, was recommended (Ref. 1). A process for using ExTran is given in Figure 2. ITL ported the product to the available Concurrent Computer Corporation 3260 super-mini at minimal cost. The HPOTP examples were immediately transported to ExTran and the resulting module was now a true, albeit simple, knowledge base system (KBS) utilizing "Why", "How", and "What if" type questions, history files, external interfaces, and all the other features usually associated with a KBS.

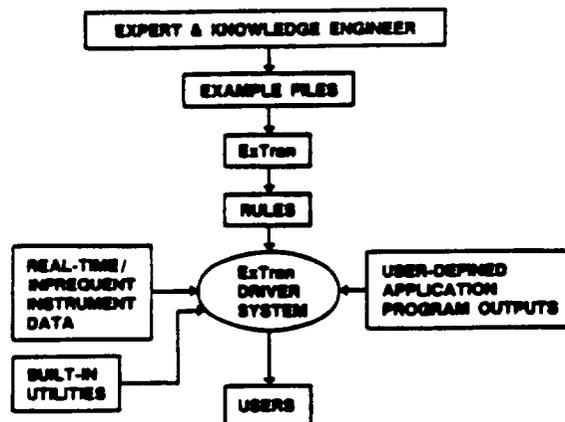


Figure 2. Inductive Expert System

Conceptually, Scotty was extended in several directions during this same time period. It was demonstrated that multiple problems could be run concurrently on the multiple processor Concurrent 3260. Graphics routines (PLOT-10 and GKS libraries) were tied to ExTran with a minimum interface. In-house statistical routines were easily linked to Scotty. Small Fortran routines were written to access SSME test files and output attribute values for input to Scotty sub-problems. Additional SSME component modules were specified. A major extension was the run-time interface between ExTran and the large data base management system DMS/32 supplied by Concurrent, then known as Perkin-Elmer (Ref. 6). These are all described extensively in a paper presented in 1986 (Ref. 3).

Scotty: CURRENT STATUS

As of mid-1988, Scotty underwent field-testing on a sub-system basis, using the taxonomy of Waterman (Ref. 19). Parts of Scotty were run in parallel with previous modes of operation to help determine the validity of the system, and to update its knowledge base. Scotty consists of far more than "just" an expert system, as is clearly shown in figure 3, but rather is one component in a fairly extensive software system. This reflects the strong belief that viable expert systems are most likely to succeed in a hybrid and integrated environment, where they must communicate easily with other standard existing and future sub-systems. This had been stressed by the author since the initial conception, contrary to the host of stand-alone KBSs being proposed in the early mid-80's, thanks to his 25 years of software engineering experience.

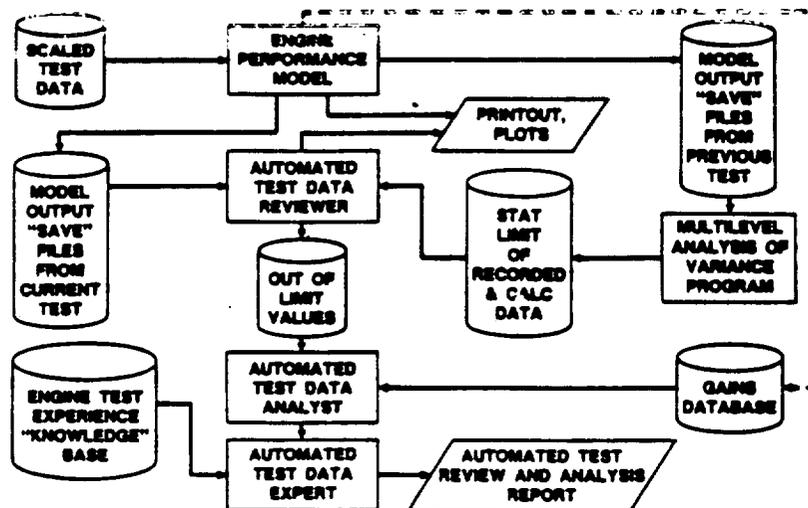


Figure 3. Context of Scotty - Automated Test Data Expert

Scotty, as of early 1988, consisted of 48 ExTran modules comprising 5400 lines of code (LOC) in Fortran. Supporting code required 7100 LOC. The ExTran generated code was derived automatically from approximately 1100 examples. Only 125 rules have involved any manual intervention to date. The other 1400 rules have been induced automatically.

KNOWLEDGE ACQUISITION ISSUES

The above numbers should be considered extremely carefully. Note that the knowledge acquisition task involves far more than simply eliciting examples from an expert or a data base. In fact, this component is relatively easy. The much more critical and difficult task revolves around the structuring of Scotty! Many in the AI field have become so enamored with the power of induction that they have forgotten some very basic software engineering principles. The top-down (divide-and-conquer) strategy has shown itself to be an extremely powerful one for thousands of years in the engineering field. Do not give it up just because a new powerful bottom-up technique is now possible!

The process of induction which turns an unordered set of examples (an operational specification of a task) into an ordered set of rules or code is a very powerful tool. This addition to existing computer-aided system engineering (CASE) tools would be welcome, and is probably on the horizon, based on recent press releases. However, the process is really only concerned with the generation of a software module. Most current research (Ref. 14) and the Scotty experience indicates that the majority of the expertise of an expert lies in her/his ability to structure the overall complex solution. Considerable work in the area of civil engineering at Wayne State University (Ref. 2) also substantiates this belief.

What good does it do (and what havoc can be wrought) to have one enormous module, derived from hundreds of examples with dozens of attributes? To be sure, the resulting rules probably execute with blazing speed and derive the "correct" answer. However, and this is a big caveat, who will be able to understand the resulting rule? Who would be willing to verify that the resulting rule set is accurate? When such a huge module is generated, experience to date shows that the expert finds the rules to be simply incomprehensible. What must the poor end user think? What has happened to the "transparency" of the underlying system, one of the most valuable additions of expert systems to the software field? Of what use is the much-touted explanation capability now? Why do some vendors promote that their tools can operate with thousands of examples and hundreds of attributes? ExTran, on the contrary, encourages the expert to break down her problem into sub-problems by issuing a warning whenever the length of a rule exceeds certain bounds. There are also various versions which differ in the maximum number of attributes per problem.

Is it too much to ask that practicing software engineers and expert system developers actually work together? It just "may" be that each has something to offer the other. It is so frustrating to this author, after being in both fields, and in both industry and academia since 1961, to see such miniscule amounts of two-way communication between these two groups of professionals. Only recently have there been any hopeful signs, in terms of joint conferences.

Scotty: EXTENSIONS IN PROGRESS

Development is continuing on a number of fronts for Scotty. Included are: beta-testing of a new product jointly developed by Knowledglink and Concurrent, augmenting the potential sources of existing data which can provide hidden or latent knowledge, and effectively utilizing graphics.

The major extension underway is the intention to use Reliance Expert (Ref. 5), which is the result of a joint project between Knowledglink and Concurrent with roots in the earlier work at Rocketdyne (Ref. 3). This product extends the interface between ExTran and a powerful data base system to include the knowledge acquisition component of the former, as well as the run-time interface discussed in Ref. 3 (Figures 4, 5, and 6). This product is currently undergoing beta testing at Rocketdyne.

Basically, Reliance Expert permits any data, when represented as records in a relational DBMS to serve as a source of knowledge (usually hidden or latent) for the knowledge acquisition phase (induction) of ExTran. One of the uses for this portion of Reliance Expert would be to serve as an "expert" for historical knowledge of Scotty, as it can now be transformed automatically into examples and then to rules. So, once again, the knowledge acquisition bottleneck becomes less and less of an issue, as it will be possible to go directly from records in a DBMS to production rules in an expert system. Moreover, it is even possible for the expert system component to modify the DBMS, should that be desirable.

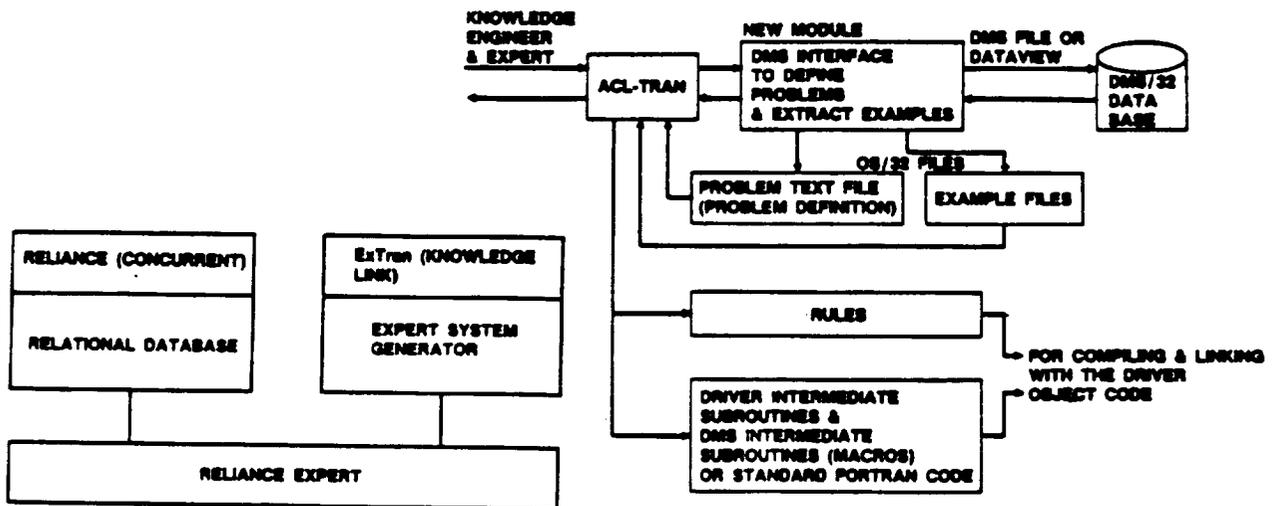


Figure 4. Reliance Expert Structure

Figure 5. Reliance Expert Development Phase

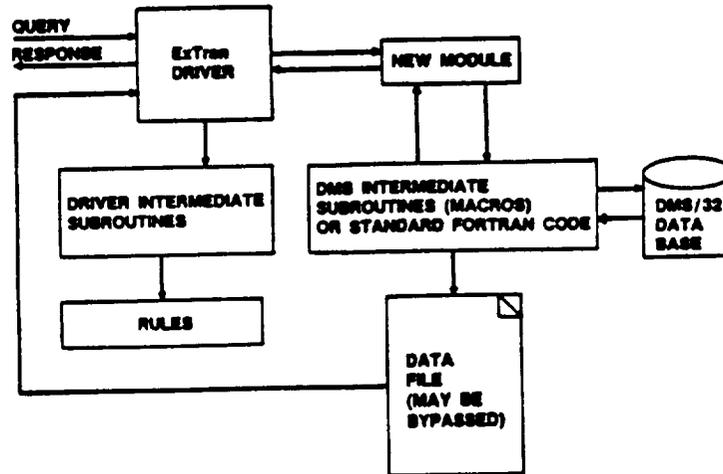


Figure 6. Reliance Expert Run-time Phase

There is a wide selection of existing data bases which could lend themselves to exercising the Reliance Expert product. Anomaly data from SSME testing is one source among several that also include Failure Modes and Effects Analysis (FMEA), turbopump build and history, and hazard tree data. Anomaly data, although primarily hardware-oriented, is a useful source of information. It provides a starting point for converting much of the SSME testing expertise repository into machine readable form. Some efforts are underway to use this source to augment the experience now encapsulated in the heads of senior engineering staff. Each anomaly data sheet consists of three major fields: problem (symptoms), analysis (causes), action for next test and other recommendations. Zero or more anomalies are recorded for each test, usually very minor ones. By carefully reviewing each anomaly and any back-up plots/tables, it is possible to convert each one into an example format consisting of a set of attribute-values and decisions.

Graphics is also being included in future versions of Scotty. A SSME instrumentation chart, now taped to the walls of hundreds of Rocketdyne engineering offices, has been converted to a dynamic color computer graphics form. The graphics subsystem has capabilities to zoom, highlight problem areas (according to actual test data measurements), and depict flow. This is not CAD/CAM, although there are a few common themes, nor is it extensive CFD modeling of the National Aerospace Plane (NASP) using multi-million dollar CRAY 2s. It is a practical and feasible use of moderate color resolution on the readily available super-mini and terminals. Engineers on the floor, as would be expected, are very pleased to see in graphical form what they have hitherto had to dig out of static tables and plots.

FUTURE DEVELOPMENT ISSUES

Further in the future are several concerns. There is an interest in each as a potential contributor to improving the quality of SSME test analysis. Obviously, Rocketdyne is keenly concerned also about technology transfer to other types of engines, in addition to the SSME. The company is deeply committed to supply the power system for Space Station Freedom, as a result of being named the prime contractor. The National Aerospace Plane (NASP) engines are also likely candidates for Rocketdyne. Expendable Launch Vehicles (ELV), the Advanced Launch System (ALS), Orbital Transfer Vehicles (OTV), and other propulsion and energy systems are also promising areas.

These further-reaching concerns are concentrated both in application and technical areas. On the application side, Rocketdyne would like to investigate the potential of extending Scotty to handle a limited subset of the measurement data for flight engines. The incorporation of health and test monitoring is also of high interest. Design of modified and new engines is a challenging option. This could perhaps involve using the current computer model for SSME test analysis to help generate examples for potential design consideration. A recent paper gives some insights on such proposals (Ref. 4). An obvious application is to enlarge the context of Scotty to include new hire training on SSME test analysis.

On the tool side, the issue of dealing with uncertain and/or noisy example data is significant. Real engineering problems involve uncertain and incomplete information. A noted nuclear engineer, Dr. Billy Koen at the University of Texas in Austin, has gone so far as to define the engineering method as "the use of heuristics to cause the best change in a poorly understood or uncertain situation within the available resources" (Ref. 9).

It is apparent, based on recent IJCAI, AAAI and IEEE conferences that induction is receiving considerable attention, so fuzzy induction is probably just around the corner. A recent U.S. based inductive workshop (Ref. 2), just on the heels of an international conference on induction and the founding of an International Special Interest Group on Inductive Programming in 1987, all bodes well for this extremely active area of research. We will see additional and powerful tools on the market which offer such practical features. Recent work at the University of Tennessee Space Institute holds considerable promise for dealing with both qualitative and temporal issues relevant to rocket engine testing (Ref. 8). Abductive reasoning for diagnosis also appears to hold some promise (Ref. 13).

CONCLUSIONS

Since 1984, effort has been underway at Rocketdyne, manufacturer of the Space Shuttle Main Engine (SSME), to automate much of the analysis procedure conducted after test firings. We thus report on progress in building the full Scotty system, after a noted 23rd century rocket propulsion expert.

Major progress has occurred on a technical front. Since the very inception of the program, it has been strongly believed that the intrinsic nature of SSME test analysis and character of inductive-based ESBTs represents an excellent match of problem and tool. The intuition has been confirmed by the relative ease with which expertise has been transformed to a structured system of modules composed of examples and thence to effective production rules. The structuring relies upon well-known software engineering techniques, and is aided by commercial CASE tools. The transformation from records in a data base to examples to production rules is accomplished automatically with Reliance Expert, a product combining a RDBMS and an inductive tool. The engineering staff responsible for building (and eventually maintaining) Scotty has consistently used examples as input. The knowledge-acquisition "bottleneck" is thus much wider than for most previously-reported expert systems. The end result is a software system which meets the real needs of Rocketdyne, and is deliverable in a cost-effective manner with less than usual maintenance requirements.

REFERENCES:

1. A-Razzak, R., T. Hassan, A. Ahmed, ExTran 7.2 Users Manual, Intelligent Terminals Ltd., Glasgow, Scotland, 1986.
2. Arciszewski, T., "Potential Applications of Induction in Engineering," Proceedings of the International Workshop on Inductive Programming, Inductive Programming Special Interest Group (IPSIG), Detroit, MI, 1989.
3. Asgari, D. and K. Modesitt, "Space Shuttle Main Engine Test Analysis: A Case Study for Inductive Knowledge-Based Systems Involving Very Large Data Bases," IEEE International Conference on Computer Software & Applications (COMPSAC), 1986, pp. 66-71.
4. Chen, K. and S. Lu, "A Machine Learning Approach to the Automatic Synthesis of Mechanistic Knowledge for Engineering Decision-Making," IEEE Conference on Artificial Intelligence Applications, 1988, pp. 306-311.
5. Concurrent Computer Corporation, Reliance Expert, Version 2 48-xxx F00 R00, Slough, England, 1988.
6. Concurrent Computer Corporation, Reliance DBMS, 04-338 F01 M99 R08, Oceanport, NJ, 1987.
7. Daumann, A. and K. Modesitt, "Space Shuttle Main Engine Performance Analysis Using Knowledge-Based Systems," ASME International Conference on Computers in Engineering, 1985, pp. 55-62.
8. Dietz, W. and M. Ali, "Qualitative and Temporal Reasoning in Engine Behavior Analysis and Fault Diagnosis," NASA Conference on Artificial Intelligence for Space Applications, 1987.
9. Koen, B., Definition of The Engineering Method. American Society for Engineering Education, Washington, D.C., 1985.

10. Modesitt, K., "Space Shuttle Main Engine Anomaly Data and Inductive Knowledge-based Systems: Automated Corporate Expertise," NASA Conference on Artificial Intelligence for Space Applications, 1987, pp. 203-212.
11. Modesitt, K., "Experience with Commercial Tools Involving Induction on Large Data Bases for Space Shuttle Main Engine Testing," Proceedings of the Fourth Expert Systems International Conference, London, 1988, pp. 219-230.
12. Modesitt, K., "Experts: Human and Otherwise," Proceedings of the Third Expert Systems International Conference, London, 1987, pp. 333-341.
13. Nau, D. and J. Reggia, "Relationships between Deductive and Abductive Inference in Knowledge-based Diagnostic Problem Solving," Proceedings of the First Expert Database Systems International Workshop, Benjamin-Cummings, 1986. pp. 549-558.
14. Shapiro, A., Structured Induction in Expert Systems. Addison-Wesley, 1987.
15. Sopp, G., "Scotty: Beaming Up a New Look at SSME Performance," Threshold, Rocketdyne Division, Rockwell International, Canoga Park CA, Number 2, 1987.
16. Tuckwell, R., "Scotty's Enterprise," Computer Systems, Bromley, U.K., August, 1987, pp. 18-20.
17. Warman, D. and K. Modesitt, "A Student's View: Learning in an Introductory Expert System Course," Expert Systems, An International Journal of Knowledge Engineering, Spring, 1988, pp. 30-39.
18. Warman, D. and K. Modesitt, "Learning in an Introductory Expert System Course," IEEE Expert, Spring, 1989, pp. 45-49.
19. Waterman, D., A Guide to Expert Systems. Addison-Wesley, 1986.

VIP: A Knowledge-Based Design Aid for the Engineering of Space Systems

Steven M. Lewis and Kirstie L. Bellman
Computer Science Laboratory
The Aerospace Corporation
Los Angeles, CA 90009-2957

Abstract

This paper describes the Vehicles Implementation Project (VIP), a knowledge-based design aid for the engineering of space systems. VIP combines qualitative knowledge in the form of rules, quantitative knowledge in the form of equations, and other mathematical modeling tools. The system allows users rapidly to develop and experiment with models of spacecraft system designs. As information becomes available to the system, appropriate equations are solved symbolically and the results are displayed. Users may browse through the system, observing dependencies and the effects of altering specific parameters. The system can also suggest approaches to the derivation of specific parameter values.

In addition to providing a tool for the development of specific designs, VIP aims at increasing the user's understanding of the design process. Users may rapidly examine the sensitivity of a given parameter to others in the system and perform tradeoffs or optimizations of specific parameters. A second major goal of VIP is to integrate the existing corporate knowledge base of models and rules into a central, symbolic form.

Introduction

VIP is a portion of Vehicles, a long-term research effort on the part of The Aerospace Corporation to develop artificial-intelligence tools for the conceptual design of spacecraft and other systems. Currently, much of the knowledge and tools used for such design are scattered throughout the corporation. The tools, while sophisticated, are often poorly documented and require a detailed knowledge of the tool itself, as well as of the system under consideration; in order to use the current tools effectively, the user of a model of spacecraft batteries needs a detailed knowledge of both the model as well as batteries. Complex and detailed setup procedures and relatively rigid structures often discourage engineers from examining alternative choices. This complexity precludes planners from using many of the existing models in the early stages of a system's design.

As older engineers retire and younger people enter the organization, a major problem has arisen regarding the maintainability of existing tools. Too often, organizations find themselves dependent on sophisticated tools that were developed by personnel no longer with the organization. These tools become difficult to maintain and, without the experts who developed the package, equally difficult to rewrite. A long-term goal is to reorganize the corporate knowledge base to make it more accessible and maintainable. Vehicles is an attempt to develop tools that allow engineers to concentrate on the rules and equations of a model, while leaving to a central system the details of implementation, solution, and interface. The hope is that models will be translated into a central, easily represented and documented form that can be useful to both experienced and casual users.

An object within the system, e.g. a frame or a slot, consists essentially of two parts, a **class** and a specific **instance**. The class holds information that is common to all instances. For example, a spacecraft may have multiple antennas. Each instance points to a common frame, of the class antenna. Within this class there is a **slot** that holds the antenna's diameter. The slot points to a template that holds common information: name, default values, limits, definitions, and default print units. The instance holds information specific to one frame: the value, the default print units, information about how the value was derived, and any user-supplied annotation. Similarly, relationships such as equations also exist in two forms: a template attached to the frame template, and a specific instance in which variable names are replaced by pointers to the variable instances.

Actions

All actions in the system are initiated by the user. Three basic user actions are possible: (1) The user may enter a value for an attribute, (2) the user may add a new subsystem to the design, or (3) the user may request a view of the current state of the design. When the user enters a new value, the effects of that change throughout the system are handled by the **propagator**.

The Propagator

When a new value is entered by the user, that value is tested against the limits imposed by the attribute template. If acceptable, the new value is passed to the relationships that are attached to that slot. Equations may be solved for any unknown; however, equations are reorganized and solved for a particular unknown only when all other unknowns have acquired values. Once an equation is solved for a given variable, subsequent changes in the independent parameters are automatically propagated through the equation to change the derived value. The derived value may be altered by the user in two ways. First, one of the independent values may be set to **UNKNOWN**; this causes the solution of the equation to be retracted and the equation to revert to a state of having two unknown values. In that state, the assertion of a value for the previously derived parameter causes the equation to be solved for the retracted value.

A second means of setting the value of a derived parameter is available under the tradeoff view. The system of equations in the model is repeatedly solved for differing values of an independent parameter. The resultant values of the dependent parameter are collected and displayed as an *x,y* graph showing how the independent parameter varies with the dependent parameter. The user may then use the mouse to select a value of the independent parameter that results in the desired value of the derived parameter. We prefer this latter approach because it preserves the causal ordering of derivation. That is, we do not say that the cost of a spacecraft was determined by the cost of a subsystem, but rather that the spacecraft's cost was determined by the weight of that subsystem, whose weight was in turn selected to give a specific cost [Simon 84].

Views

VIP contains a set of tools that allow users to visualize data. These tools provide the user with multiple **views** of a complex design. Different tools are available at the design, subsystem, and attribute levels of the hierarchy. Tools at the design level allow the user to visualize an overview of the components tree (Figure 3). Clicking on any subsystem displays that subsystem.

Attribute-level tools allow users to view the properties of individual attributes. These tools are accessed by clicking on the name of the attribute in the subsystem window (Figure 4). Where the views are appropriate to the current value of the slot, a menu of possible views is then displayed.

Choices Available for All Slots

- **Information:** This choice gives the definition of the item, tells how it was derived (if a value is known), and displays all the slots that depend on that value as well as all slots used to compute the value. In addition, any annotations are displayed. Information is displayed in a locked, scrollable text window.
- **Annotate:** This choice allows the user to view and edit annotation related to a given value. This is usually used to allow the user to add notes justifying his decision.
- **How:** If the item is set, this feature causes the system to tell how it is set, or, if unset, how it might be set; if unset, this choice lists all rules and equations that could be used to derive the item; if none exist, this choice tells the user that he must input a value for the item, as there is no other way to derive one. Note that many of the equations in the system normally use some items as inputs only, and thus may be inappropriate for deriving these items. For example, the cost of a satellite is normally proportional to its weight. It is rarely useful to suggest to the user that the satellite's weight could be estimated if the cost were known.

Choices Available Only When Appropriate

- **Sensitivity:** This choice displays the sensitivity of a derived value to all parameters used in its derivation. Each of the deriving parameters is varied by 5% of its current value and the resultant effects on the dependent parameter are computed as numerical derivatives. The derivatives are normalized as $\delta \log(y)/\delta \log(x)$ [Landauer].
- **Tradeoff:** This choice allows users to visualize the effects on a derived value of altering one parameter. Users select a parameter from a list of parameters used to derive the current value. The system automatically varies the selected parameter over a selected range and generates a graph showing the effects on the dependent parameter. Users may select on the dependency graph a point that causes that combination to become the new current values.
- **Recommend:** If a value is unknown but VIP can find a collection of equations based on known parameters designated "usually user entered" or on values derived from the above, this choice displays the potential dependency tree, highlighting the unknown values and allowing the user to enter the remaining unknowns.
- **Apportion:** If the value is the sum of N terms, this choice displays the relative importance of the additive terms as a pie chart.

Interface

A major feature of the philosophy underlying VIP is to enable users to interact with the system in a simple and intuitive fashion. The **interface** consists of a collection of windows; in the current implementation, only one window is displayed at a time. Some of these windows are described below. The two basic windows are the **subsystem**

The Vehicles project has the long-range goal of developing new capabilities and knowledge representations for supporting the conceptual design of spacecraft. VIP has the more immediate goal of utilizing methods developed in this project to provide an intelligent system that allows users to develop and manipulate space-system designs. To make VIP widely available, we developed it for use on a personal workstation (the Apple Macintosh II). This project has been concerned with developing a system that is simple and user-friendly enough to shift much of the use and extension of the knowledge base from the computer science lab to the working engineer.

VIP Architecture

The architecture of VIP follows a common practice in expert-system design, by making a clear separation between the knowledge in the system and the procedures used to manipulate that knowledge. VIP is divided into two basic parts -- an engine for manipulating knowledge and user selections, and a collection of relevant knowledge sources.

Figure 1 displays the architecture of VIP. The seven major components of the engine are as follows:

- **Knowledge Base:** This stores general knowledge about the systems the user is developing. Data from the knowledge base (currently stored as a series of textual files) are parsed by the system into working code. Storing knowledge as text allows the user to interact with easily understandable forms of the knowledge base, facilitating the reading, editing, and correction of the data. The generated code is used by the system but is not normally accessed by either users or developers. The knowledge base is described in detail below.

- **Knowledge Editor:** This tool allows the knowledge base to be updated. Users may add new subsystems, rules, or equations to the database. Knowledge that is added is tested for syntax and consistency with the current contents of the database. The editor tags each piece of information with the name of the person entering the knowledge, and encourages users to attach annotation detailing the technical underpinnings of each item.

- **Working Database:** This stores current designs. Designs are represented as frames that have slots containing information about parameter values, information about component hierarchies, information about how parameter values were derived, and user-supplied comments. Designs in the working database may be created and manipulated by the user. A design (for example, of a proposed spacecraft) may be stored in the knowledge base, permitting future manipulation.

- **Propagator:** This tool accepts changes in parameter values. Any change in a parameter's value is propagated through all constraints, rules, and equations in the system.

- **Report Generator:** The tool generates a report on the current design. The report is user-readable text formatted to allow VIP to read in the report, regenerating the working design.

- **Interface:** VIP has a built-in user interface that allows users to browse and alter information rapidly. The interface is described in detail below.

- **Toolbox:** A major component of the interface is the toolbox, which gives the user a number of views into the data. The toolbox also allows users to read, add, alter, or annotate any portion of the design.

Knowledge Base

The knowledge base stores four classes of information: templates, fixed systems, historical designs, and working designs. These are described below.

Templates hold knowledge about how to generate specific elements of an object, storing what is possible to know about a specific subsystem. Such information includes possible parent systems, possible component systems, attribute names, equations, rules, and constraints. Attribute templates store the type of attribute (if numeric), the type of measurement (e.g. distance), and default units (e.g. km). The attribute template may also store default values and upper and lower bounds. If an attribute involves a choice (e.g. a choice of material), then the template holds all possible choices.

Fixed systems are instances of subsystems. As such the attributes are assigned specific values. Fixed systems represent knowledge that may not be altered in the current design. A communications band such as X-band would be considered as a fixed system that holds the assignments for uplink and downlink frequencies, bandwidths, and the effects of adverse weather on the signal. The choice of launch vehicle is another class of fixed system that contains cost, availability, possible orbits, and throw weight.

Historical systems comprise data from complete systems that have already been developed. Like fixed systems, they comprise read-only data. Historical systems may be accessed to give the user perspective about what is possible. The system can access the historical system most similar to the current design and, in the absence of other information, can use parameter values from this system as defaults in the current design.

Working designs are also complete systems. Unlike historical systems, working designs may be modified.

The knowledge base is stored as a collection of textual files that hold information as keyword, value pairs. Figure 2 shows a portion of the current knowledge base describing a power subsystem. The current system parses these text files into Prolog code; the files are then added to the working database as required.

Working Database

A design such as a spacecraft or space system may be considered as a hierarchical frame system. The design is a root frame whose slots contain specific values called **attributes** and frames called **components**. The component hierarchy may be visualized as a tree (Figure 2). The design and all its components are a class of frame called a **subsystem**. In addition to components and attributes, a subsystem has slots for relationships: equations, rules, and constraints. All attributes and relationships are attached to a specific subsystem frame. Relationships can link to attributes in other frames; these links allow information to propagate through the design.

window, which gives a detailed view of a specific subsystem, and the **navigation window**, which shows all subsystems as parts of a tree.

• **Subsystem Window:** The subsystem window (Figure 4) displays all the slot values in a scrolling field. Each slot has four components: **Name**, **Value**, **Units** (if applicable), and **Source**. The name is simply the name of the slot. Values may be strings, numbers, or tuples. Every slot type points to code that generates an appropriate display of the slot's contents. If these contents are undefined, a blank rectangle is displayed. Numeric slots may have units associated with the slot's value. If these units are defined, they are displayed next to the data. The source type is displayed as an icon at the right of the slot. Currently, we support icons for **user**, **default**, **equation**, **rule**, and **system** (copied from a predefined system, such as the throw weight of the shuttle).

Navigation Window

The navigation window (Figure 3) displays the components as a tree, with the design at the root, various subsystems as the branches, and the components as the leaves. The user may open a selected subsystem by clicking on the appropriate box. The tree gives the user a sense of where the current system lies relative to the entire design. This window enables the user to develop insight into the architecture of the design under consideration. The navigation window also enables the user to move throughout the design.

Dependency Window

The dependency window displays trees that are generated for an individual parameter. Two possible trees can be derived. When the value of a parameter is derived, the dependency tree shows the current dependencies; for example, cost was derived from weight and availability, weight was derived from power and coverage, and coverage was entered by the user. The leaves in the dependency tree represent default values as well as values entered by the user. Dependency trees may be constructed for any derived parameter. When a parameter has no value, a potential dependency tree can be generated by seeking parameters that are noted as "usually user entered."

Other Windows

Other windows are generated to display graphical information, presented as an x,y graph. The relative contribution of terms to a sum is presented as a pie graph, and the sensitivity of a dependent parameter is presented along with its dependencies. Here the derivatives are normalized as described above and are presented as a bar graph.

Hardware

The system is implemented on an Apple Macintosh II with 5 MB of memory and using AAIS Prolog. A version runs on the Sun/3 under Quintus Prolog, but this version does not have the full user interface that is available on the Mac. The system requires approximately 2.5 MB of working memory.

Problems Studied

We have applied the system to a number of conceptual studies within the corporation. For example, we have developed models for electrical power systems,

constellations of communications, and configurations for SDI-related designs. The most complex models were able to handle 20 subsystems and approximately 100 equations.

In several cases the models were generated by engineers working directly with the system, rather than by personnel within the Computer Science Laboratory. The success of the system with relatively naive users was encouraging. The problems these people commonly encountered (misspelling the names of system parameters or referring to parameters not explicitly imported from the owning subsystem) have highlighted the need to provide more tools to verify the input data, both syntactically and in terms on the internal consistency of the generated model.

References

Bellman, K. and A. Gillam, "A knowledge-based approach to the conceptual design of space systems," *AI Papers 20, Proceedings of the Conference on AI and Simulation, 1988*, SCS International.

Landauer, C., "Sensitivity Analysis," Technical Report, The Aerospace Corporation (in process).

Iwasaki, Y. and H. Simon, "Causality in Device Behavior," *Artificial Intelligence 29:3-32* (1984).

Parsaye, K., M. Chignell, S. Khoshafian, and H. Wong, *Intelligent Databases* (Wiley, New York, 1989).

Figure Captions

Figure 1. VIP Architecture

Figure 2. Navigation Window. The components of a design may be viewed as a tree. The system highlights the currently active system. When this window is displayed, clicking on any box causes that system to become active and displays a subsystem window for that subsystem. This enable the user to change the active subsystem.

Figure 3. Subsystem Window. The basic view of a subsystem, this window allows a user to view and modify specific attributes. The Traverse button brings up the navigation window. Clicking on the name of any attribute (e.g. Time_Delay) pops up a menu of applicable tools for that attribute. Clicking on the value box allows the user to type in a new value. If the attribute has only a fixed set of alphanumeric values, clicking on the box pops up a menu of choices. The icon indicates the source, and clicking on the icon displays information about how the parameter was derived.

Figure 4. Knowledge Base. This section of the knowledge base illustrates how knowledge is entered as keyword text. A major goal of VIP was to keep the working knowledge base in a form that can be accessed and edited by the user.

Figure 1

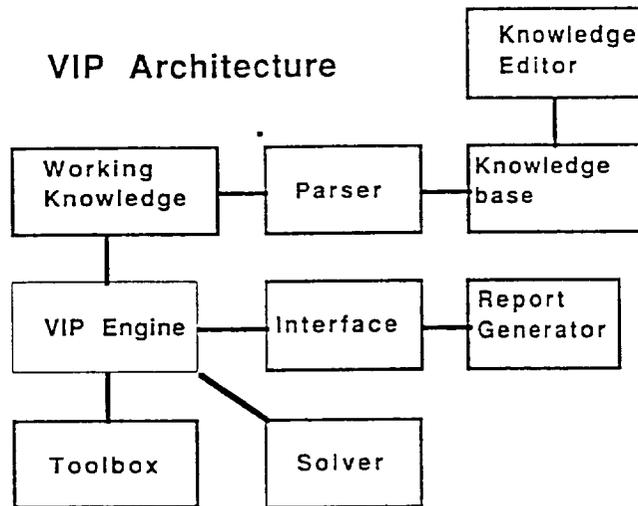


Figure 2
Sample System Specification File

Note sections in *bold italics* are the designator. Remaining text is merely commentary.

```

% *****
% power
%
% Begin Subsystem Specification
%
subsystem power
possible parents buss % must be a subsystem of the buss
% designate possible components
possible components regulator solar_panel power_buss

% *****
%
% Begin Designating Attributes
%
% *****
attribute type choice solar_panel solar_body nuclear solar_concentrator
default solar_panel

%
% Specify a numeric attribute
%
% name type units metric english

attribute eol_power numeric power watts watts
lower limit 100 watts
upper limit 5000 watts
what " eol_power is the end of life power. That is the power the power
system can deliver at the end of the design life of the spacecraft. Because
components in the system, specifically solar cells degrade over the
spacecraft lifetime, this will be significantly lower than the bol_power or
beginning of life power. Power systems are usually designed to deliver a
specific end of life power. "

attribute battery choice nicad nih lithium lead_acid
  
```

Figure 3 Navigation Window

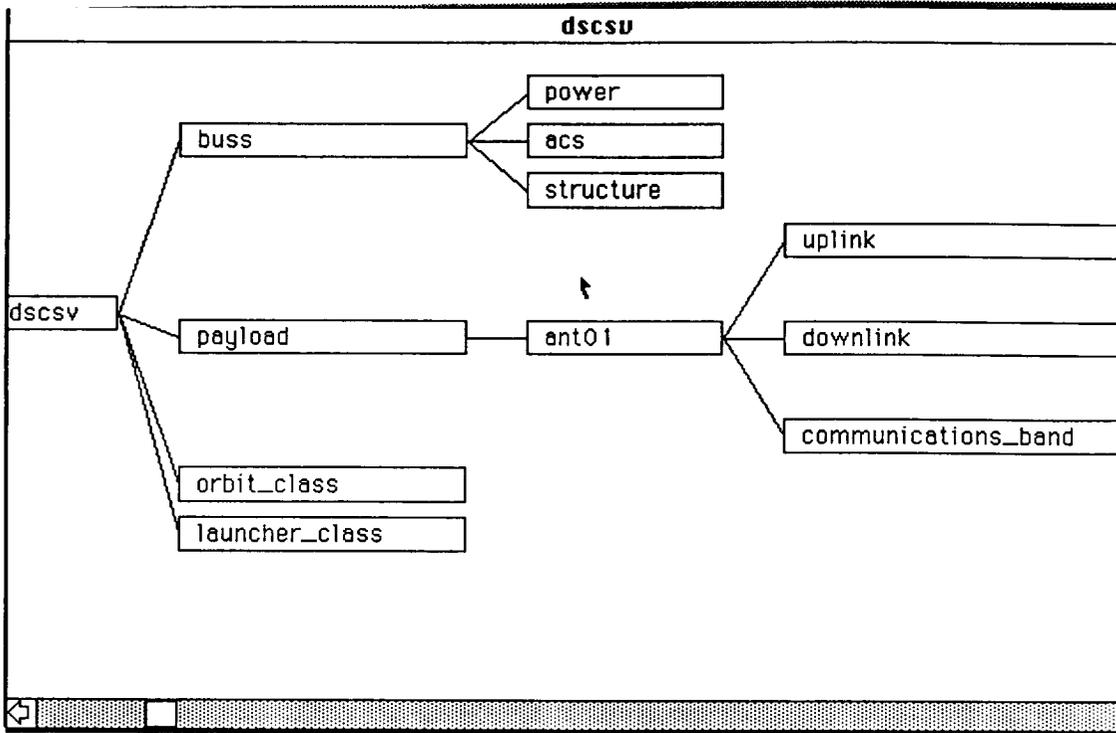


Figure 4 Subsystem Window

The subsystem window is titled "boost_kkw" and contains a toolbar with "Traverse", "Undo", "Undo Choice", and "Quit" buttons. Below the toolbar is a list of parameters, each with an input field and a unit. A vertical toolbar on the right side contains a home icon, a scroll bar, and several icons including a person's face, "e=mc² EQN", and "D Default".

Parameter	Value	Unit
Time_Delay	30.0	sec
Time_Comm	40.0	sec
Pk_Effective	0.0250	ratio
Time_Postboost	200.	sec
Absentee_Ratio_B	36.0	ratio
Time_Kkv_Flight	130.	sec
Kkv_Per_Booster_B	8.0	ratio
Boost_Leakage	0.800	ratio
N_Boosters	1000.	ratio
Weight		kg

Using Decision-Tree Classifier Systems to Extract Knowledge from Databases*

D.C. St. Clair ,
C. L. Sabharwal
University of MO--Rolla
Graduate Engineering Center
St. Louis, MO 63121

Keith Hacke,
W.E. Bond
McDonnell Douglas Research
Laboratories
St. Louis, MO 63166

ABSTRACT

One difficulty in applying artificial intelligence techniques to the solution of "real world" problems is that the development and maintenance of many AI systems, such as those used in diagnostics, require large amounts of human resources. At the same time, databases frequently exist which contain information about the process(es) of interest. Recently, efforts to reduce development and maintenance costs of AI systems have focused on using machine learning techniques to extract knowledge from existing databases. This paper describes research conducted at McDonnell Douglas Research Laboratories in the area of knowledge extraction using a class of machine learning techniques called decision-tree classifier systems. Results of this research suggest ways of performing knowledge extraction which may be applied in numerous situations. In addition, a measurement called the Concept Strength Metric (CSM) is described which can be used to determine how well the resulting decision tree can differentiate between the concepts it has learned. The CSM can be used to determine whether or not additional knowledge needs to be extracted from the database. An experiment involving "real world" data is presented to illustrate the concepts described.

INTRODUCTION

Applying AI techniques to solve diagnostic problems often requires that information contained in one or more databases be converted to knowledge. One common way of performing this conversion is to use domain experts. For example, when experts are asked to assemble a set of rules for diagnosing a particular system, they review information from sources such as schematics and existing maintenance databases. Then they develop a set of diagnostic concepts, generally stated as a set of rules, which correlate diagnostic inputs with the desired diagnosis(es). Since large amounts of human resources are required to perform this knowledge extraction, it is desirable to automate as much of this process as possible.

Databases use attributes, A_j , and their associated values, a_{jj} , to represent information about quantities of interest. These attributes may represent both numeric (discrete and continuous) and nonnumeric quantities. A database is an organized set of these attributes and their values, a set of relations among these attributes, and a language for manipulating attributes and the relationships among them. This structure transforms raw data into information (18).

While information contained in a database may be accurate and complete, it is not knowledge. Using information as knowledge requires identification of the pertinent logical entailments hidden in that information. It is these logical entailments that allow inferences to be made from information contained in the database. Identification of logical entailments is complex and is usually done by

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

special algorithms (2). This paper is concerned with the automated extraction of knowledge from databases and the representation of this knowledge in a structure that can be processed by logical entailment algorithms (18).

The first step in performing knowledge extraction is to determine how knowledge will be represented. While various knowledge representation schemes have been developed for expressing concepts and their relationships to other knowledge, the relationships modeled by all these schemes can generally be expressed in terms of first-order logic expressions (2). One well-known knowledge-representation scheme is the rule-based system characterized by its knowledge base of facts and rules. In this paper, all references to rules apply to any knowledge representation which is used to model first-order logic expressions. The actual physical knowledge representation is secondary in importance.

The second step in performing knowledge extraction is to determine which algorithm should be used in the extraction process. One of the more popular and successful classes of algorithms which are used for this process are called decision-tree classifier systems. These systems take training instances as input and produce a set of rules as output. The rules output by the system are represented in the form of one or more decision trees (3,10,11,).

Recent research has focused on the automated extraction of knowledge from existing databases in an effort to reduce the development and maintenance costs of AI systems. This is a complex problem since concepts may take many forms, the identification of appropriate attributes is difficult, and sufficient information may not be available to support the formation of clear and accurate concepts. Other factors which contribute to the complexity of this problem are the difficulty in determining when extraction is complete and the difficulty of evaluating the knowledge produced.

The following sections describe an approach for extracting knowledge from databases which addresses many of these difficulties. The approach described is applicable in cases where the extracted knowledge can be represented as a set of rules. The extraction techniques use a class of inductive machine-learning techniques called decision-tree classifier systems. The section entitled Evaluation of Knowledge Extracted describes a metric which is useful for measuring the results of the extraction effort. The last section shows the results obtained by extracting knowledge from a "real" database (6).

EXTRACTION OF KNOWLEDGE FROM DATABASES

Type of Concepts to be Learned: One of the first steps in knowledge extraction is to determine the type of concepts to be learned. For instance, if one is trying to extract diagnostic information from a database, it is usually desirable to express the concepts being learned as rules. Machine learning techniques, such as decision-tree classifier systems, are proficient at this form of extraction. This approach is applicable to both numeric and nonnumeric data. However, continuous numeric-valued attributes present special problems (13).

Uncertainty plays a major role in knowledge extraction. Uncertainty involves both the uncertainty of facts and of rules. Fact uncertainty may be the result of noisy training examples. Noise is hard to identify since it is difficult to differentiate noise from "exceptions to a rule." Although several different approaches have been tried for handling noisy data (3,11), noise still presents a difficult problem for knowledge-extraction techniques. Rule uncertainty not only concerns the certainty with which conclusions can be asserted within a rule, but also the way in which uncertainties are propagated along rule chains. Both types of uncertainty introduce serious problems in knowledge extraction and continue to be active areas of research.

If the concepts to be learned are in the form of mathematical equations, standard operations research and statistical techniques such as regression (linear and nonlinear), correlation, and hypothesis testing may produce more satisfactory results (7). Although operations research and

statistical approaches are used primarily with numeric data, nonnumeric attributes can be assigned numeric values. Once the appropriate operations research techniques have been applied, the resulting equation(s) must then be interpreted in light of their nonnumeric counterparts (5).

The extraction techniques described below assume that the database on which extraction is to be performed consists of a set of records. For the purpose of knowledge extraction, each record is viewed as a set of attribute-value pairs along with one or more associated conclusions. Whether or not the database is a single entity or a distributed database is not important. Richardson (12) has developed an algorithm for combining information from numerous relations into single records. Each of these records becomes a training instance to the machine learning technique.

Representation of Concepts: Learned concepts will be represented in the form of rules. For example, the rule:

$$A_1(a_{11}) \wedge A_4(a_{43}) \wedge A_2(a_{22}) \Rightarrow C_6$$

(1)

denotes the concept: if the value of attribute A_1 is a_{11} , the value of attribute A_4 is a_{43} , and the value of attribute A_2 is a_{22} , then C_6 may be concluded. (Boolean-valued expression $A_i(a_{ij})$ is true when a_{ij} is the value of A_i .) Fig. 1 shows how rule (1) would be represented by a decision tree.

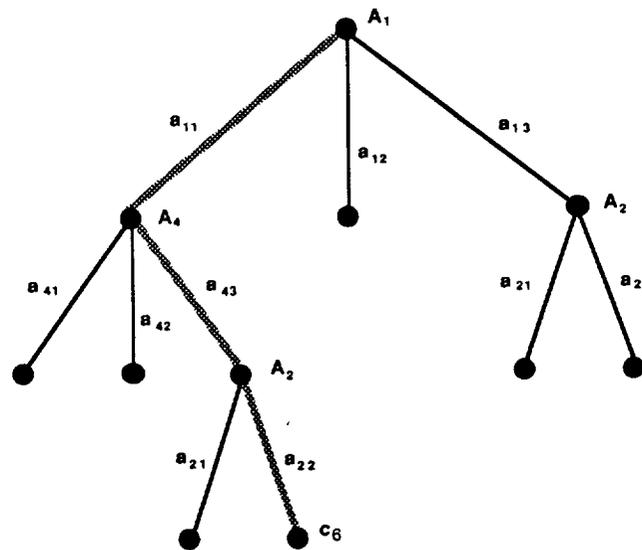


Fig. 1 Decision Tree Showing $A_1(a_{11}) \wedge A_4(a_{43}) \wedge A_2(a_{22}) \Rightarrow C_6$

Individual concepts are represented as paths in the decision tree. Each internal node represents an attribute A_i while each branch descending from A_i corresponds to a specific attribute value, a_{ij} . Each leaf node, c_j , represents a conclusion out of the set C of all possible conclusions.

Since more than one conclusion may exist at a leaf node, the concept shown in Fig. 1 will be represented by the tree shown in Fig. 2. The trees are identical with the exception of the label at the

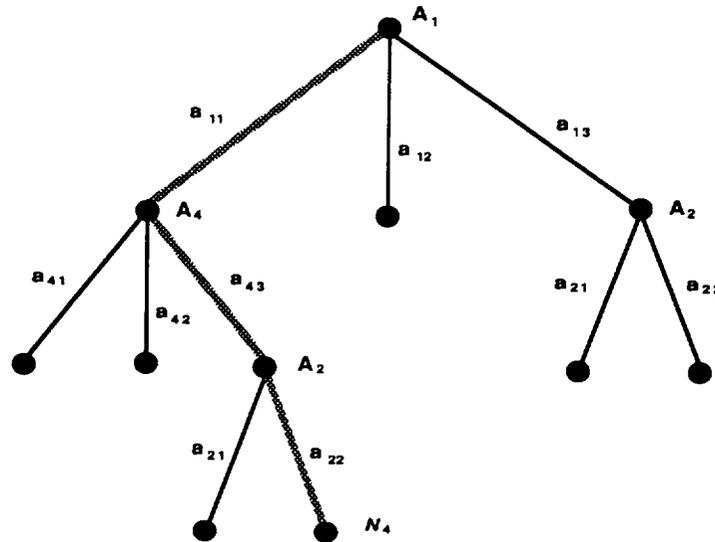


Fig. 2 Decision Tree Showing $A_1(a_{11}) \wedge A_4(a_{43}) \wedge A_2(a_{22}) \Rightarrow C(N_4)$

leaf node. The label, N_4 , represents the set of one or more conclusions, $c_j \in C$ occurring at this node. The expression:

$$A_1(a_{11}) \wedge A_4(a_{43}) \wedge A_2(a_{22}) \Rightarrow C(N_4)$$

denotes the rule where $C(N_4)$ is the conjunction of all $c_j \in N_4$. Note that: (1) multiple conclusions may be present at any leaf node, i.e., $|N_k| > 1$, and (2) any conclusion, c_j , may be present at more than one leaf node, i.e., $c_j \in N_k$ for more than one value of k . **Similar concepts** are concepts which have the same conclusion.

The problem of extracting knowledge in the form of decision trees reduces to the problem of constructing "correct" trees. Decision-tree classifier systems are a class of machine learning techniques which can be used to construct such trees.

DECISION-TREE CLASSIFIER SYSTEMS

Decision-tree classifier systems take training instances as input and output decision trees like that shown in Fig. 2 (3,10,11). These systems are called classifier systems because they separate input training instances into different classes. They are also referred to as induction systems since they induce knowledge from examples. Decision trees are frequently used to represent the results of this classification, hence the name decision-tree classifier systems.

During construction of the tree, the decision-tree classifier must determine the best attribute to be used to expand the tree at each node. It must also determine when no further attributes should be added to a path of the tree. Induction of decision trees may be incremental or nonincremental. In nonincremental induction, all training instances are processed at one time and the decision tree created. At this point, the learning process is considered completed. In incremental induction, learning is performed each time the decision tree is used to classify an instance. A well-known nonincremental induction technique called ID3 was developed by Quinlan (11) and is based on earlier work in induction by Hunt et al. (9). Two incremental versions of ID3 have been developed; ID4 by Schlimmer and Fisher (14) and ID5 by Utgoff (19).

Other types of machine learning approaches are applicable to knowledge extraction. These include case-based reasoning(15), explanation-based learning(4), and genetic algorithms(8). They will not be discussed in this paper.

DECIDING WHAT TO EXTRACT

The selection of the appropriate database attributes to participate in the extraction process is critical both to the quality of the knowledge extracted and to the efficiency of the extraction process. The choice of attributes depends on the type of concepts being learned. In many cases, the domain expert may be able to provide advice on which attributes are likely to be important. The attributes chosen to participate in the extraction process make up what is called the description space, viz;

$$D = \{ A_1, A_2, \dots, A_n \}.$$

In an effort to keep the description space as small as possible, statistical and mathematical programming techniques such as regression analysis and correlation can be used to help identify database attributes which are dependent on each other. When selecting database attributes to be included in the description space, it is seldom necessary to include attributes which are dependent on others already in the set. Mathematical programming techniques can be used to help identify linear and some types of nonlinear dependence among attributes. In some cases, simple plots of database values may help identify appropriate attributes.

Once a candidate description space is identified, the next step is to perform knowledge extraction using only a subset of the training instances available. This is desirable since the machine learning mechanism being used may also help identify relationships among attributes which have not been detected by earlier efforts. After evaluating these initial results, it may be possible to further revise/refine the description space.

All of these efforts are designed to keep the complexity of the extraction process to a minimum. Minimization of complexity is desirable because a database may contain a large number of attributes. Simply using all database attributes in the knowledge extraction process would only increase the complexity of the extraction process without adding additional knowledge.

EVALUATION OF KNOWLEDGE EXTRACTED

Since the accuracy of the concepts learned as well as the complexity of the tree constructed is determined by both the quality and quantity of training instances and by the way the classifier system chooses attributes for the tree, it is desirable to evaluate the "quality" of the knowledge extracted. Knowledge quality can be measured in different ways, including the correctness and thoroughness of the knowledge extracted and the certainty with which the knowledge structure can differentiate between the concepts learned.

Evaluating knowledge correctness is necessary to determine how well the concepts learned compare with what is known about the "real" world. Correctness evaluations are done in a manner similar to verification and validation of expert systems (17). A common set of test suites is evaluated first by using the extracted rules and then by domain experts. Next, these test results are compared. This approach helps verify that the set of concepts learned is consistent with domain experts' knowledge. Failure to adequately satisfy correctness tests may be the result of poor attribute selection, poor extraction techniques, or an inadequate number of training instances.

In many cases, domain experts discover that the knowledge extracted is correct but not thorough. This is evidenced by the fact that "pieces" of knowledge are found missing during the tests for correctness. This may indicate an inadequate number of training instances in the database. In these cases, additional knowledge may have to be added by domain experts.

To measure how well a decision tree differentiates between concepts, the authors developed an approach for evaluating the quality of a learned decision tree by measuring certain characteristics of the tree. This approach complements the work of domain experts and is especially useful in cases where multiple conclusions exist at a leaf node, i.e. $|N_k| > 1$, for some value(s) of k . The approach uses the Concept Strength Metric (CSM) described below.

The development of the Concept Strength Metric was motivated by the need to construct diagnostic advisors for use in aircraft maintenance (1,16). By utilizing inductive learning systems, it is possible to construct diagnostic advisors which can assist in maintaining their own knowledge bases. However, one of the problems arising from such learning systems concerns the quality of concept differentiation since it is rarely the case that all concepts will be learned perfectly.

The Concept Strength Metric value, $E(c_j)$, for each conclusion c_j , is the weighted measurement indicating that, given the decision-tree's current level of experience, it can uniquely differentiate conclusion c_j . The value $E(c_j)$ is the sum of individual weighted strengths, $E_k(c_j)$, for c_j at each leaf node, viz:

$$E(c_j) = \sum_{k=1}^{\lambda} E_k(c_j),$$

where λ is the number of leaf nodes in the tree.

The weighted strength, $E_k(c_j)$, for conclusion c_j at node N_k , is the weighted probability that conclusion c_j can be clearly differentiated by the path leading to node N_k . It is calculated by computing:

$$E_k(c_j) = \frac{\delta_{jk}}{\sum_{i=1}^{|C|} \delta_{ik}} * \frac{\delta_{jk}}{\sum_{h=1}^{\lambda} \delta_{jh}} = \alpha_{jk} \beta_{jk}$$

where δ_{jk} denotes the number of times conclusion c_j has appeared at node N_k and $|C|$ denotes the number of possible conclusions in the tree. The frequency of c_j occurrences in the tree is given by η_j . The frequency of all conclusions occurring at node N_k is ξ_k . Note that:

$$\xi_k = \sum_{i=1}^{|C|} \delta_{ik} \quad \text{and} \quad \eta_j = \sum_{k=1}^{\lambda} \delta_{jk}$$

The factors

$$\alpha_{jk} = \frac{\delta_{jk}}{\sum_{i=1}^{|C|} \delta_{ik}} = \frac{\delta_{jk}}{\xi_k} \quad \text{and} \quad \beta_{jk} = \frac{\delta_{jk}}{\sum_{h=1}^{\lambda} \delta_{jh}} = \frac{\delta_{jk}}{\eta_j}$$

are of interest. The first, α_{jk} , denotes the fraction of all conclusions at node N_k which are c_j . The

larger the value of δ_{jk} , the higher the likelihood of uniquely identifying conclusion c_j at this node. Since

$$\sum_{j=1}^{|\mathcal{C}|} \alpha_{jk} = 1$$

the larger the number of conclusions at node N_k , the smaller the likelihood that all conclusions will be uniquely differentiated at this node. If only one conclusion, say c_1 , is present at node N_k , then $\alpha_{1k} = 1$. If $c_j \notin N_k$, then $\alpha_{jk} = 0$.

The factor β_{jk} scales α_{jk} by the fraction of c_j occurrences at all leaf nodes. This factor scales the current knowledge about conclusion c_j at node N_k with respect to all the information about conclusion c_j . Hence, the product, $E_k(c_j) = \alpha_{jk} \beta_{jk}$, is the weighted measure that conclusion c_j can be clearly differentiated by the path leading to node N_k , given the current level of knowledge.

To illustrate the Concept Strength Metric, consider the tree shown in Fig. 3. The values at each leaf node N_k indicate the nonzero number of times each conclusion $c_j \in \mathcal{C}$ has occurred at that node. For example, at node N_1 , conclusion c_3 occurred twelve times. No other conclusions occurred at this node. This experiment contained 168 training instances with $|\mathcal{C}| = 5$, $\eta_1 = 65$, $\eta_2 = 50$, $\eta_3 = 20$, $\eta_4 = 23$, $\eta_5 = 10$, $\xi_1 = 12$, $\xi_2 = 75$, $\xi_3 = 60$, $\xi_4 = 13$, and $\xi_5 = 8$. Table I shows the values of $E_k(c_j)$ and $E(c_j)$ for each conclusion.

Several interesting results are observable from the table. Each $E_k(c_j)$ in the table has a value between zero and one. Each $E_k(c_j)$ value represents the weighted probability that conclusion c_j can be uniquely identified when tree traversal leads to N_k . Conclusion c_3 is of particular interest since it appears at two nodes and it does not appear with any other conclusions. Hence, based on the present knowledge level of the tree, it is possible to positively identify all similar concepts whose

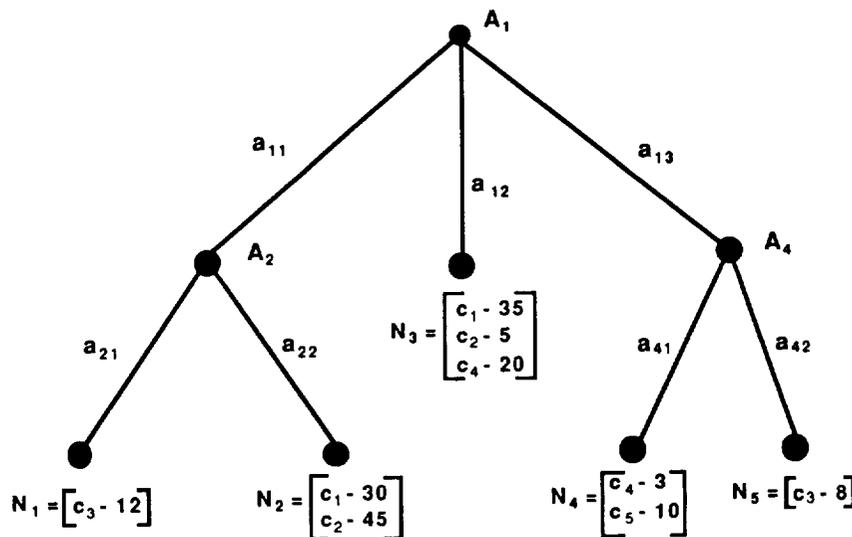


Fig. 3 Sample Tree Showing Conclusions at Each Node

k	$E_k(c_1)$	$E_k(c_2)$	$E_k(c_3)$	$E_k(c_4)$	$E_k(c_5)$
1	0	0	0.60	0	0
2	0.18	0.54	0	0	0
3	0.31	0.0083	0	0.29	0
4	0	0	0	0.03	0.77
5	0	0	0.40	0	0
$E(c_j)$	0.49	0.548	1	0.32	0.77

Table I. CSM Values for Fig. 3

conclusion is c_3 . This is reflected by the fact that $E(c_3) = 1$. It can be shown mathematically that $E(c_j) = 1$ whenever for each node N_k such that $c_j \in N_k$, then c_j is the only conclusion at N_k . Such conclusions are said to be **completely differentiated** by the learned decision tree.

The next most completely differentiated set of similar concepts are those whose conclusion is c_5 since $E(c_5) = 0.77$. Conclusions c_1 , c_2 , and c_4 are not as completely differentiated as c_3 and c_5 . These lower CSM values may imply that multiple concepts are present at several leaf nodes. Individual values of $E_k(c_j)$ provide additional information about each conclusion. For example, the fact that $E_3(c_2) = 0.0083$ indicates that this decision tree is a poor differentiator of c_2 at node three. The cause of this poor differentiation may be the result of noise or it may be due to a lack of training examples which contain this concept.

The $E(c_j)$ and $E_k(c_j)$ values can be used to determine when the decision tree has reached the desired level of concept differentiation. They can also be used to guide the learning process by indicating what types of additional knowledge are needed to improve the tree.

KNOWLEDGE EXTRACTION EXPERIMENT

The objective of this experiment was to extract rules from a database of iris flowers compiled by Fisher (6). The database contained four flower-description attributes with an associated flower type: *virginica*, *versicolor*, or *setosa*. Initial plots of attribute vs iris type suggested that neither sepal length nor sepal width alone were sufficient to predict iris type since several values for each of these attributes were associated with multiple iris types. Hence, these two attributes were chosen for the description space.

Utgoff's ID5 (19) was chosen as the extraction mechanism. ID5 was extended to calculate values of the CSM. Results from these experiments are shown in Table II. Given there were two attributes; sepal width with 26 values and sepal length with 41 values; and there were 94 leaf nodes, it can be seen that the resulting decision tree was wide and shallow. There were ten leaf nodes which contained multiple conclusions. At the conclusion of the learning process, the training set was used to test how accurately iris variety could be predicted. The tree produced was able to distinctly classify only 84% of the training instances. This means that 16% of the training instances fell into one of the ten leaf nodes containing multiple conclusions.

Evaluation of the knowledge extracted was based primarily on how well the decision tree differentiated between concepts. The CSM values shown in Table II indicate that the CSM value of clearly differentiating virginica is the same as the CSM value for versicolor. The fact that $E(\text{setosa}) = 1.0$ implies that the tree clearly distinguishes the setosa species, since all leaf nodes containing setosa as a conclusion do not contain any other conclusions.

# Training Instances	150
# Training Attributes	2
# Internal Nodes	20
# Leaf Nodes	94
# Leaf Nodes containing multiple conclusions	10
$E(\text{virginica})$	= 0.88
$E(\text{versicolor})$	= 0.88
$E(\text{setosa})$	= 1.00

Table II. Results of Iris Tests

CONCLUSIONS

This paper has described the general process of extracting knowledge from databases using decision-tree classifier systems. These learning mechanisms, based on induction, extract knowledge from input training instances and represent it in the form of decision trees. The Concept Strength Metric (CSM) was described for measuring the amount of concept differentiation in these decision trees. This result is important in helping to determine when sufficient knowledge extraction has been performed. By examining values of $E_k(c_j)$ and $E(c_j)$, it can be decided which conclusions require additional training instances to improve concept differentiation. The CSM may be effectively used to evaluate concept differentiation in any decision tree. Experimental results using the Concept Strength Metric are generating interest among practitioners in the diagnostic community.

REFERENCES

1. Bond, W. E., St. Clair, D. C., Flachsbart, B. B., and Vigland, A. R., *Integration of an Adaptive Diagnostic Expert System into an Avionics Test Environment*, **Proceedings of the Third Annual Expert Systems in Government Conference**, IEEE Computer Society, October 1987, pp. 132-136.
2. Brachman, R.J., *The Basics of Knowledge Representation*, **AT&T Technical Journal**, Vol. 67, No. 1, 1988, pp. 7-24.
3. Clark, P. and Niblett, T., *The CN2 Induction Algorithm*, **Machine Learning**, Vol 3, No. 4, 1989, pp. 261-284.
4. DeJong, K. and Mooney, R.J., *Explanation-based Learning: An Alternative View*, **Machine Learning**, Volume 1, 1986, pp. 145-176.
5. Draper, N. R., Smith, H., **Applied Regression Analysis**, 2nd ed. New York : Wiley, 1981.

6. Fisher, R. A., *The Use of Multiple Measurements in Taxonomic Problems*, **Annals of Eugenics**, Vol. 7, 1936.
7. Hillier, F. S. and Lieberman, G. J., **Introduction to Operations Research**, 4th ed., San Francisco : Holden-Day, 1986.
8. Holland, J.H., *Properties of the Bucket Brigade Algorithm*, **Proceedings of the First International Conference on Genetic Algorithms and Their Applications**, Pittsburgh, PA: Lawrence Erlbaum, 1985, pp. 1-7.
9. Hunt, E. B., Martin, J., and Stone, P. J., **Experiments in Induction**, Academic Press, 1966.
10. Michalski, R. S., Mozetic, I., Hong, J., and Lavrac, N., *The Multipurpose Incremental Learning System AQ15 and its Testing Application to Three Medical Domains*. **Proceedings of the Fifth National Conference on Artificial Intelligence**, Morgan Kaufmann, 1986, pp. 1041-1045.
11. Quinlan, J.R., *Induction of Decision Trees*, **Machine Learning**, Vol. 1, 1986, pp. 81-106.
12. Richardson, J. M., **Deduction of a Functional Dependency From a Set of Functional Dependencies**, M.S. Thesis, University of Missouri -- Rolla, 1988.
13. Sabharwal, C. L., St. Clair, D. C., Hacke, K., and Bond, W. E., *Representation of Continuous Attributes in ID_x Classifier Systems*, **Proceedings of the Fourth International Symposium on Methodologies for Intelligent Systems: Poster Session Program**, Oak Ridge National Laboratory, ORNL/DSRD-24, October 1989, pp. 167-176.
14. Schlimmer, J. C., and Fisher, D., *A Case Study of Incremental Concept Induction*, **Proceedings of the Fifth National Conference on Artificial Intelligence**, Vol. 1, Morgan Kauffman Publishers, Inc., August 1986 , pp. 496-501.
15. Slade, S., *Case-based Reasoning: A Research Paradigm*, Yale Department of Computer Science, Report # YALEU/CSD/RR #644, August 1988.
16. St. Clair, D. C., Bond, W. E., Flachsbart, B. B., and Vigland, A. R., *An Architecture for Adaptive Learning in Rule-Based Diagnostic Expert Systems*, **AI in Armament Workshop -- Diagnostics**, American Institute of Aeronautics & Astronautics, March 1988. (Reprinted from **1987 Proceedings of the Fall Joint Computer Conference**, IEEE Computer Society, October 1987, pp. 678-687.)
17. St. Clair, D. C., Bond, W. E., and Flachsbart, B. B., *Using Output to Evaluate and Refine Rules in Rule-Based Expert Systems*, **Proceedings of the Third Conference on Artificial Intelligence for Space Applications, Part I**, "NASA Conference Publication 2492, November 1987, pp. 9-14.
18. Ullman, J. D., **Principles of Database and Knowledge-base Systems**, Computer Science Press, Vol. 1, 1988, pp. 23-25..
19. Utgoff, P. E., *ID5: An Incremental ID3*, **Proceedings of the Fifth International Conference on Machine Learning**, Morgan Kaufmann Publishers, Inc., June 1988, pp. 107-120.

DEB: A Diagnostic Experience Browser using Similarity Networks

Cyprian E. Casadaban

Martin Marietta Manned Space Systems
Post Office Box 29304 Mail Stop 3691
New Orleans, Louisiana 70189

Abstract

This paper describes DEB: a fusion of knowledge base and data base that allows users to examine only the data which is most useful to them. The system combines a data base of historical cases of diagnostic trouble-shooting experience with similarity networks. A menu-driven natural language interface receives input about the user's current problem. Similarity networks provide the user with references to past cases that are most similar or most related to those they now face. The user can then choose the case that is most pertinent and browse its full textual description which, in turn, may include references to other related cases.

Introduction

The following describes some preliminary results of a NASA Mission Task being performed by the Machine Intelligence Group at Martin Marietta Manned Space Systems in New Orleans, where the External Tank for the Space Shuttle is assembled. The goal of the project is to increase productivity at weld stations by decreasing downtime (Pulaski/Casadaban 88).

When a downtime is reported a team of weld experts responds to the call. They work together to determine the cause of the problem. Their goal is to find out how to get the weld station operational as soon as possible and what to do to keep the problem from recurring. The result of a weld team call is a completed form that explains the path problem solving took from initial diagnosis through solution.

DEB grew out of a need to use this weld downtime trouble-shooting information that is gathered for each occurrence but not referenced afterward. Consequently, when a problem arises that is very similar to a previous problem, the diagnosis process must start from scratch. Instead, knowledge stored about similar downtime episodes could be consulted to lend advice about what to investigate first.

Once it was decided that the domain of aerospace hardware welding was too broad for a conventional expert system, an effort was started to create a history-based trouble-shooting assistant.

Data and Categorical Knowledge

The weld trouble-shooting team originally kept the historical data base in a limited form. A page or two of trouble-shooting information was recorded on paper but only the bare minimum of whom, what, and when was stored in the data base.

Our first task was to see that all data recorded at a weld downtime occurrence not only be written down, but typed into a dBase III plus data base as well (Figure 1). In this way all knowledge of an event was now retrievable. Data capture began in January of 1988.

Case_Number	code distinguishing each case. EX: 88/100 is the 100th case in 1988.
Tool_Number	tool on which the downtime occurred.
Date_of_Occurrence	date on which the downtime occurred.
Hours_Down	# of hours the tool was inoperable before the problem was fixed.
Vehicle_Effectivity	number of the assembly in the weld fixture EX: LW-51
Problem_Component	main component cited in the problem. EX: ROUTER
Problem_Subcomponent	subpart of the component cited in the problem. EX: MOTOR
Problem_Action	action cited in the problem. EX: NOT-OPERATING
Cause_Component	main component cited in the cause. EX: LIMIT-SWITCH
Cause_Subcomponent	subpart of the component cited in the cause EX: ARM
Cause_Action	action cited in the cause. EX: NOT-CONTACTING
Problem_Text	textual description of the problem.
Cause_Text	textual description of what the cause of the problem was.
Action_Text	textual description of what actions were taken for the fix.
Follow_Up_Text	any action items that will keep the problem from recurring.

Figure 1. The Weld Downtime Data Base Structure.

Once all the information about a downtime episode is collected, the case is categorized. This categorical knowledge lies at the heart of DEB and consists of a problem and a cause generalization.

Each generalization breaks down further using a BNF grammar to yield a systematic breakdown of allowable feature values for a given field. The problem and cause are described according to the following feature triplet:

COMPONENT / SUBCOMPONENT / ACTION

Our local expert assigns a category for the problem and cause of each case. For example, suppose the problem at a certain downtime incident is:

"Router motor is down at tool T01A5103"

The expert might categorize this problem as follows:

ROUTER / MOTOR / NOT-OPERATING

The cause of the downtime incident is categorized similarly.

Once the data is captured, it allows for the creation of informative data base sorts. Reports can then be generated showing which problems are occurring at particular tools, when, and what is causing them.

System Description

DEB, in essence, receives user input about a case and finds the top ten most similar cases. This happens in several steps. First, a C program pre-processes dBase III plus data records into the format of KnowledgePro topics. Topics are the data structure for representing similarity network knowledge. Next, these topics are read and used as frames depicting each weld downtime case, component, subcomponent, and action category used, as well as each weld tool. The user is prompted to describe their case by selecting the date of occurrence, the tool, and the component, subcomponent, action triplet which corresponds to the symptoms of their problem.

DEB then uses these parameters to find the ten most similar cases to the current problem. This is accomplished using similarity networks (Bailey 88) wherein a similarity value is retrieved for each aspect of a case as compared to the corresponding aspect of another case. These aspects are component, subcomponent, action, weld-gantry-type, weld-tool-type, weld-type, and recency of occurrence. This knowledge resides in the C-program-generated topic frames. The top ten most similar problems are derived and presented in a menu where the user can select a particular case and browse it. When a case is selected, another C program takes over and locates the case in a dBase III plus file, formats the contents of the case and sends it back to KnowledgePro in a text file which is then displayed for the user.

The system is written in KnowledgePro from Knowledge Garden Inc., and Turbo C from Borland International. Data and categorical knowledge are stored in a dBase III plus data base and similarity network knowledge is stored as KnowledgePro topics used as frames. A similarity network is a simple knowledge representation scheme which can be thought of as a set of objects bound by weighted links. The DEB system configuration is shown in Figure 2.

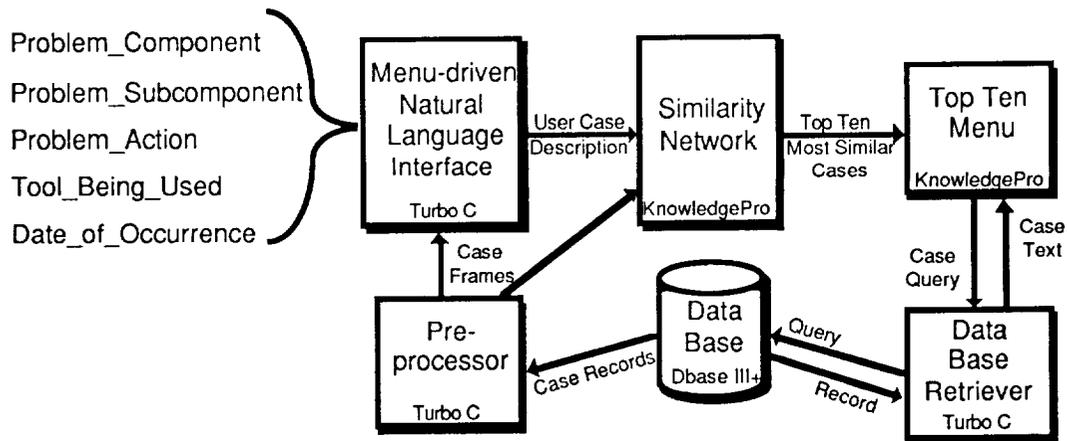


Figure 2. DEB System Configuration.

KnowledgePro was chosen for its ability to control hypertext with an underlying language, a rich data structure (topics), and a backward chaining inference engine which will be used for a future extension. KnowledgePro proved to be extremely slow communicating with the data base, so a faster data base retrieval system was written in Turbo C.

How It Works

First, the pre-processor systematically rummages through the data base and creates a summary frame for each case description (Figure 3). A frame is also created for each unique category. This feature frame contains information about the cases this particular feature has occurred in, which features it is similar to, and which other features it has been associated with in past case symptom triplets. In this way the pre-processor creates a similarity network for each problem feature.

```

Case 88/197
  case_number      88/197
  tool_number      T01A5002
  record_number    59
  date_of_occurrence 06/21/88
  problem_triplet  (ROUTER MOTOR NOT-OPERATING)
  cause_triplet    (ROUTER MOTOR DEFECTIVE)
end

component ROUTER
  is_similar_to    ((SAW 0.8))
  cases_where_seen (88/197 88/209 88/263)
  subcomponent_list (MOTOR HANDLE MUFFLER INCORRECT-INSTALLATION)
end
  
```

Figure 3. Case Summary and Component Feature Frame Examples.

Next, processing the degree of similarity of one feature to another is handled by C functions and is procedural (speed constraints on the delivery product required that this part be rewritten in C and integrated with KnowledgePro, after the initial KnowledgePro prototype). These similarity values reflect how similar in function or how related two problem features are.

At the start of the project our expert was consulted and supplied us with similarity measures in the form of fuzzy linguistic comparisons for all problem features (Schmucker 84). These measures of similarity or relatedness can be visualized as a network. The nodes are features being compared and the arcs are the results of the comparison (Figure 4). Arcs representing totally-different are not shown.

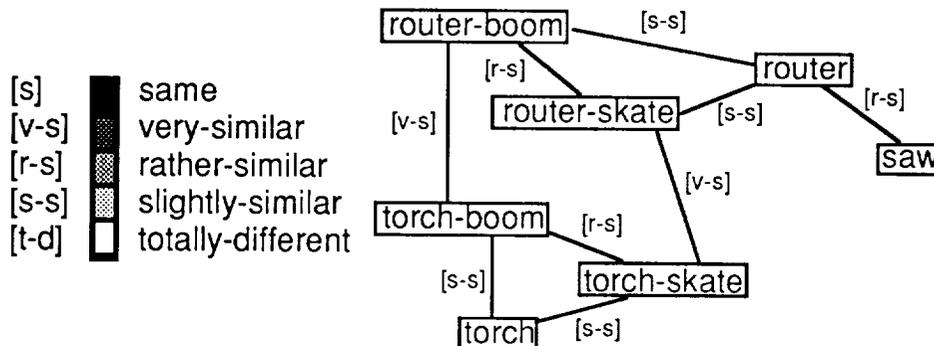


Figure 4. Component-feature Similarity Network Fragment.

Each problem feature type has a static weight associated with it based on its importance in the comparison of two cases (Figure 5). These weights were derived from interviews with our expert.

Once the similarities for each feature are found they are converted into numbers (same: 1.0, very-similar: 0.8, rather-similar: 0.6, somewhat-similar: 0.4, slightly-similar: 0.2, totally-different: 0.0). These numbers are then multiplied by the appropriate feature weight. In this way the total similarity of a similar case versus the base-case will fall in the range of 0.0 to 1.0, where the greater value reflects a greater similarity.

- (0.6) Symptom-triplet Feature
 - (0.6) Component
 - (0.2) Subcomponent
 - (0.2) Action
 - (0.3) Tool Feature
 - (0.7) Weld-gantry-type
 - (0.2) Weld-tool-type
 - (0.1) Weld-type
 - (0.1) Date Feature or Recency
- = 1.0

Figure 5. Feature Weights.

The weighted similarity value is the measure of how similar a feature is to the base feature, multiplied by the appropriate feature weight.

The way the networks are processed is as follows:

1. The user provides the system with a description of the problem (i.e., date, tool, component, subcomponent, action). This is called the *base-case*.
2. The frame corresponding to the *base-component* is accessed. The cases in which this component has been used are stored in the *similar-case-list* with a weighted *similarity-value* of "same" (since these cases have this component in common).
3. Any entries from the *is_similar_to* frame slot are stored as *similar-feature-entries* along with their weighted *similarity-values*.
4. For each *similar-feature-entry* (if any), access its frame and perform step 2 only, appending its case list to the *similar-case-list* along with the weighted *base-component-to-similar-feature-entry-similarity-value* found.
5. Repeat steps 2, 3, and 4 for the *base-subcomponent* and *base-action* just as for the *base-component* all the while appending to the *similar-case-list* or accumulating the weighted similarity of a *similar-case* to the *base-case* if that *similar-case* is already on the *similar-case-list*.
6. For the *base-case*, access the frame corresponding to its tool and find the tool's *weld-gantry-type*, *weld-tool-type*, and *weld-type* features (Figure 6).
7. For each case on the *similar-case-list* perform steps 8, 9, 10 & 11.
8. Access the frame corresponding to the case's tool and obtain the tool's *weld-gantry-type*, *weld-tool-type*, and *weld-type* features.
9. Access the frames for each *tool-feature* and store its weighted similarity measure against the appropriate *base-case-tool-feature*.
10. Compare the *similar-case-date* to the *base-case-date* and calculate *recency*(weighted) according to a formula in which a case is rated higher when it is more recent.
11. Calculate overall similarity versus the *base-case*.

12. Sort the *similar-case-list* by overall similarity and show the user a menu of the top ten most similar cases (Figure 7).
13. The user can now choose a case to browse. Once their selection is made, the full case description is displayed and related cases can be browsed.

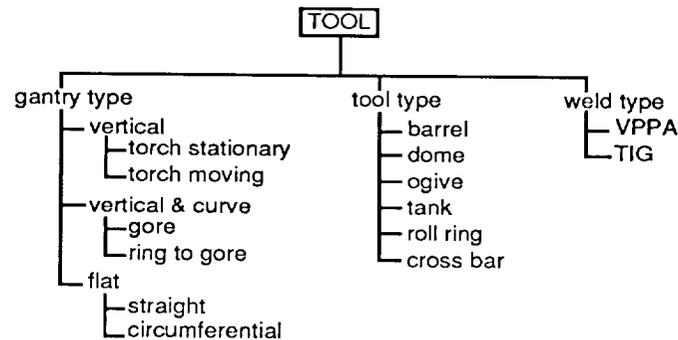


Figure 6. Weld Tool Features Hierarchy.

Interface Description

The DEB interface uses both menu-driven natural language and hypertext, which insures that every selection the user can make is valid. Both the initial user interface and the top ten browser were written in KnowledgePro. At the start of a DEB session, the user is presented with a sentence fragment in a natural language window and a menu of possible components to choose from (much like Texas Instrument's Natural Access). Once a component is chosen, the natural language window is updated to reflect their selection. Then they are shown a subcomponent menu which is made up of only the subcomponents related to the selected component. When this choice is made, similar things happen and a menu of related actions is shown. In this way the interface guides the user's selection and echoes their menu picks by way of an english sentence. Finally the user is asked to choose a tool and to enter the date of the downtime problem occurrence.

DEB - Diagnostic Experience Browser

The problem is that a router motor is not operating on tool T01A5103.

Top Ten Browsing

I would like to browse case ...

Similarity Value	Case	Date	Tool	Problem Component / Subcomponent / Actor
0.86	88/197	06/21/88	T01A5002	ROUTER/MOTOR/NOT-OPERATING
0.81	88/226	07/19/88	T03A5012	ROUTER/NOT-OPERATING
0.80	88/196	06/20/88	T01A5002	ROUTER/NOT-OPERATING
0.79	88/288	09/02/88	T01A5002	ROUTER/NOT-OPERATING
0.79	88/319	09/30/88	T01A5002	ROUTER/NOT-OPERATING
0.74	88/091	03/17/88	T01A5103	ROUTER/NOT-MOVING
0.74	88/076	03/03/88	T01A5103	ROUTER/NOT-MOVING
0.62	88/031	02/01/88	T02A5006	ROUTER/NOT-OPERATING
0.69	88/062	02/24/88	T03A5014	SAW/NOT-ENGAGE
0.62	88/066	02/26/88	T03A5014	SAW/NOT-ENGAGE
0.56	88/040	02/08/88	T01A5103	ROUTER/NOT-BACK-GROOVE
0.50	88/095	03/21/88	T01A5103	ROUTER/NOT-CUT-PER-ENG-SPECS
0.42	88/284	08/29/88	T04A5016	ROUTER/MOTOR/SPARKING

Figure 7. Top Ten Most Similar Cases Menu.

When DEB has processed the user's request, a menu appears showing the top ten most similar past cases (Figure 7). When the selection of a case to browse is made, the full textual episode description is retrieved from the data base and shown in a window. Using hypertext, the user can choose to generate another top ten menu based on this trouble-shooting account's problem feature-triplet or cause feature-triplet. If this case references another case (by mentioning its case number), the user can highlight this hypertext thread and DEB will show the corresponding case in the same manner as the case from which it was spawned. This browsing can continue until the user is done.

Issues and Lessons Learned

During the evolution of this task many issues were discussed, tested and evaluated. One of the more prevalent was the blurry line that separated problems and causes. At first thought this does not seem to be an issue, however, it was soon discovered that through the investigation and categorization of problems and causes there was a relationship much like a chain. A chain in that, depending on where a problem was discovered, different but related problem-cause pairs would be named. For example, there is a sensor that detects how far the weld torch is from the metal part. Suppose this sensor fails causing the torch to dive into the part. If the sensor was controlled by a computer and the controlling parameters were input by an engineer, there are more than one problem-cause combination for this anomaly. The choice depends on where in the chain of events the problem is discovered. Often a cause is found that followed from another factor that was at first hidden (perhaps a power surge, in this example).

For these reasons the problem and cause category features were not stored in separate similarity networks, but grouped together in one. This allows the user to follow the chain of events possible for a class of downtime occurrences by allowing the user to browse similar cases which may have been described or categorized differently. The similarity networks perform well in this task.

It was also discovered that a better batch of cases to browse could be attained by using the similarity networks on a large base of cases and assigning thresholds to improve performance. These thresholds may mean limiting the number of similar features each individual problem feature can have to only the most similar. Another way of instituting thresholds is to limit the number of cases that are fully processed and sorted for the top ten. In this way, only those above a certain similarity value after only the symptom-triplets are compared. These types of enhancements yield the best results for a large historical base of events.

Future Enhancements

Several enhancements have been decided upon to make DEB a more intelligent assistant. First, a report module to give statistical analysis of problem/cause trends and a natural language explanation system will be written. Also to be added is a weld team formation module that would be an expert system to decide which members of the weld team are needed to diagnose a problem. This task is currently performed by the weld team coordinator who calls members of the team based on the problem description called in. A weld team formation expert system would alleviate the problem of wasting an expert's time going to a tool site when they are not needed.

It would be useful for the system to perform the case problem and cause categorization automatically. A great amount of expert-derived rule-based knowledge about the domain would be needed along with is-a hierarchies and natural language keyword extraction routines.

Yet another enhancement our group is considering is to combine DEB with another AI system developed here called ELMO. ELMO is a case-based memory building tool and stands for Episodic Long-term Memory Organizer (Pulaski 88). Instead of a similarity network, ELMO uses hierarchies of knowledge to make generalizations about cases. The cooperation of the two knowledge representations would provide added reasoning capability for future applications.

Conclusion

This type of historical browsing capability lends itself to many memory intensive tasks whether they are reasoned off-line or in real-time. For an intelligent assistant, the easy and timely access of experience is essential for tasks in the domain of aerospace manufacturing, as well as other domains. Moreover, this ability is useful for domains in which events are not assembly-line proven but are similar enough that past experience can be used to produce a quality product.

References

- (Bailey 88) Bailey, D., Thompson, D., and Feinstein, J. "Similarity Networks." PC AI, July/August, 1988. pp 29-32.
- (Pulaski/Casadaban 88) Pulaski, K., and Casadaban, C. "Case-Based Reasoning: The Marriage of Knowledge Base and Data Base," in proceedings of Fourth Conference on Artificial Intelligence for Space Applications, Huntsville, Alabama, November 15-16, 1988, NASA Conference Publication 3013, pp. 183-190.
- (Pulaski 88) Pulaski, K., "ELMO: An Episodic Long-Term Memory Organizer for Case-Based Reasoning," AAAI Case-Based Reasoning Workshop, St. Paul, Minnesota, August 23, 1988, pp. 107-112.
- (Schmucker 84) Schmucker, K. J., Fuzzy Sets, Natural Language Computations, and Risk Analysis. Computer Science Press, Rockville, Maryland, 1984.

Artificial Intelligence Techniques for Modeling Database User Behavior

Steve Tanner and Dr. Sara J. Graves
Department of Computer Science
The University of Alabama in Huntsville
Huntsville, Alabama 35899

Abstract

This paper describes the design and development of the Adaptive Modeling System. This system models how a user accesses a relational database management system in order to improve its performance by discovering use access patterns. In the current system, these patterns are used to improve the user interface and may be used to speed data retrieval, support query optimization and support a more flexible data representation. The system models both syntactic and semantic information about the user's access and employs both procedural and rule-based logic to manipulate the model.

Introduction

The users of a database management system (DBMS) often repeat particular patterns of usage. If these patterns are known during the design phase of the database, they can be used to structure the data in an efficient manner. Some DBMSs allow users to manually incorporate information about these patterns into the database (e.g. create new views, indexes, etc.). However, very few systems are able to recognize and maintain a model of these patterns for the individual user's benefit.

The Adaptive Modeling System (AMS) is a tool that creates and maintains a model of a user's queries to a DBMS and the relationships between and within those queries. The AMS changes the model to constantly adapt to the way in which the DBMS is currently being accessed by a particular user. As the use of the database changes over time, the AMS is able to monitor and model these changes automatically. Furthermore, intelligent use of the knowledge stored in the model enables the system to recognize patterns and trends. This information can be used by both the users and the administrator of the DBMS.

This paper describes a system that models a user's accesses to a database. This knowledge of the user is maintained in the model's knowledge base (KB) to enhance the user's access to the DBMS. As a user's access to the data changes over time, the system adapts its knowledge base to reflect those changes. The system learns the types of actions that a user is performing and the relationships between the data in the database that those actions imply. The knowledge base models both semantic and syntactic information in a flexible manner. It is manipulated by both procedural and rule-based logic. This use of conventional methods and AI techniques (DBMS and

Knowledge-Based Information and procedural and rule-based logic) creates a powerful combination for extending a database. The system is designed to enhance the use of a SQL-based query language to access a relational database system.

Related Research

The system described in this paper integrates research from several areas including machine learning, object-oriented database design, and entity relationship design.

Much of the research in machine learning has centered around the idea that a system can deduce rules from examples [Michalski 1983] [Michalski 1986] [Winston 1984]. Since the AMS operates while users query a database, it has a ready source of example queries. However, one of the problems with many current learning systems is that the examples must be correct and static (i.e. if it's correct now, it will always be correct). The AMS acquires information that is dynamic and at times incorrect with regard to future information.

Intelligent tutoring systems is an area of adaptive learning that has been explored by the authors. In many of these tutoring systems a changing model of user behavior is established such that as incorrect information is given to them, they must respond accordingly [Visetti 1987] [Woolf 1987]. The tutoring systems also expect the model of the user to change, as the user becomes more skilled at the task being tutored.

The integration of object-oriented techniques and relational systems [Blaha 1988], and using modified types of object-oriented structures in the building of database systems [Chen 1976] are under consideration for the AMS.

Attempts to improve the speed of retrieval of information from databases through the use of knowledge base and semantic query optimization are also of interest to the designers of the AMS. Some current efforts in these areas are being undertaken by [Brunner 1988], [Chakravarthy 1987] and [Malley 1987].

The Basis for the AMS

The AMS is based on the following assumptions:

- When a user accesses data in a database, he does so in a manner that often leaves specific traceable patterns. These patterns represent relationships in the data that are important to the user.
- These patterns can be identified and retained by an intelligent modeling system.
- The AMS can use these patterns to enhance the performance of database systems in a number of ways.

The AMS is aimed specifically at improving the user interface by making query predictions of which the user can take advantage when accessing the database. This

and some of the other improvements that can be realized by using recognized patterns are listed below:

- **Improve The User Interface:** Knowledge as to what the user is probably about to do can be used to give the user meaningful prompts. This will eliminate keystrokes as well as help the user organize his thoughts.
- **Speed Data Retrieval:** If the system can accurately predict what data the user is likely to access next, the data can be retrieved before the user needs it.
- **Support Query Optimization:** Over time, the system can recognize ways in which users access the same information in multiple ways. This information could be used by a query optimization mechanism.
- **Allow Flexible Data Representation:** The system may be able to reorganize the structure of the data based upon relationships discovered between data. This would allow the system to adapt to new unforeseen uses without explicit user or administrator intervention.
- **Support Trend Analysis:** Trend analysis is based to a large extent upon recognizing patterns of behavior in a system. Any recognition of these patterns would be of great interest for trend analysis.

When a given user is accessing data in a database, there are often links between the individual data items that form patterns. These data patterns are used extensively when the initial database is being designed (e.g. associated items are often grouped into one table). In order for these patterns to be used however, they must be known when the database is designed. Also, some patterns may come into conflict with one another, and therefore all cannot be incorporated in the initial design.

When users access a particular piece of data, or a particular type of data, they will often take certain actions either prior to or just after the access of that data. If these actions occur often, they will mark a pattern in the use of that data, even when these patterns were not known during the initial database design. In addition, these patterns are likely to shift with time. The way in which a user accesses data may change as the needs of that user change.

The system can identify patterns by looking for repeated behavior that occurs with relation to particular types of data. Since the patterns are likely to change over time, the system can continue to identify new patterns, and discard old patterns that are no longer useful.

Basic Components of the AMS

The AMS is a system that may be used in several different ways and, therefore, it is flexible in its layout and structure. To allow experimentation with different techniques during development, the system is modular in nature. The AMS itself consists of four primary internal components (Figure 1): the AMS Interface, the AMS Control Center (ACC), the Abstract Relation Rule Base (ARRB) and the User Model Knowledge Base (UMKB). Brief descriptions of these components are given below.

The **AMS Interface** is the primary link between the AMS and the User/DBMS. This window-based interface accepts queries from the user and passes them on to the DBMS as well as on to the ACC. The interface can display information to the user from both the DBMS and from the AMS.

The **AMS Control Center** controls the actions of the AMS. It is where decisions are made about what information to display to the user, when to invoke the rule base and how to use the information generated by the AMS components. Since the ACC is the main control center of the system, its operations must be efficient and its interactions with the UMKB strictly procedural in nature. The slower rule-based operations are invoked only on an as-needed basis.

The **User Model Knowledge Base** is used to store knowledge about the users of the database. A model of each individual user is stored in the UMKB and is used to represent how that user accesses the database. The model is represented as tables in a relational DBMS, and consists of four primary levels: The Syntactic level, the Semantic level, the Inter-Query level and the Data level. The first two levels are used to represent syntactic and semantic information about queries of a given user. The last two levels are used to represent more abstract relationships between queries and between data items stored in the user's database.

The **Abstract Relation Rule Base** is used to store rule-based information about the system's model of the user's behavior. It consists of tables that represent the premises and conclusions of the rules and how they should work together. The rule base is generally reserved for the control of more abstract types of modeling and is the primary mechanism for the modification of the Inter-Query and Data levels of the knowledge base.

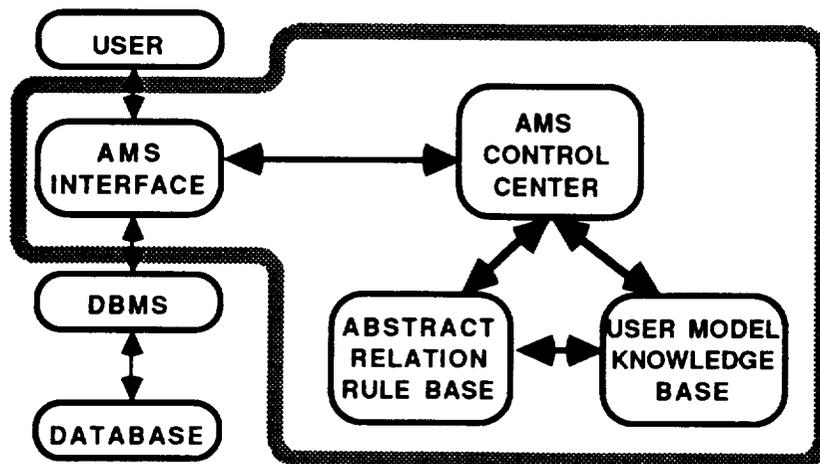


Figure 1: Layout of the AMS

The User Model Knowledge Base

The UMKB is structured as a set of relational tables with three levels of knowledge representation. The Semantic level represents knowledge about interactions within past queries. The Inter-Query level represents knowledge about relationships between separate queries. The Data level represents knowledge about relationships between database items.

In all of the levels, the system uses a unique identification (ID) number associated with each of the user's queries. By tagging information about each item in a query with the query's unique ID before storage, the entire query can potentially be recreated later. The IDs are also used in the precedence of relationships between queries since the IDs are assigned in sequential order. This ordering is exploited when looking for relationships between queries. For example, with the unique IDs, the knowledge base can be searched for such things as "What are the attributes of any relation that was created within the last five queries?" or "Has relation A been modified within the last several queries?" Also the tuples of some tables have weights associated with them. These weights represent how frequently and how recently the particular tuple was used. This information is accessed and updated during the traversal of the tables by the ACC and the ARRB. It is also used to flush old information from the system.

Tuples in the UMKB are subject to being flushed after a period of dormancy. If a tuple is not accessed by the system after an appropriate period of time, it is removed from the system in order to keep the size of the tables manageable. If a tuple is used frequently (i.e. the user takes the same path frequently) the query ID associated with that tuple remains current, and where appropriate, the weights remain high. This information is updated each time the tuple is accessed. If a tuple is not used frequently, the ID becomes old and any weights are decreased. A bookkeeping mechanism of the ACC will periodically eliminate tuples with lower weights.

The Semantic level is a representation of past queries made by the user. It consists of a set of tables that represent the possible SQL queries (CREATE TABLE, CREATE VIEW, CREATE INDEX, SELECT, etc.). Each command is structured as a set of tables that represent the important components of that command. For example, the SELECT command is represented by a set of tables that contain the table names that the SELECT command is operating on, the attributes selected, the ORDER BY clause information and others. Each tuple of a table contains information about one particular query and is tagged with that query's unique ID. By using the ID of a query to select information from the appropriate tables, the query can be recreated in its entirety. This level is used by both the ACC for straightforward traversal as well as by the ARRB for more complex analysis. Since represented queries are broken out into their component parts, the AMS is able to look for patterns that are close but do not necessarily match.

The Inter-Query level is used primarily by the ARRB for making abstract connections between queries. The structure is fairly simple and consists of only one

table. Each tuple of the table represents one relation between queries. Each tuple consists of a pair of query IDs, each with a code that marks the type of query (e.g. SELECT, NESTED SELECT, UPDATE, etc.) and a textual string that designates the relation between the ID pair. The rule system is able to use this table to store and retrieve information about relations between query pairs by either looking for the type of relationship necessary, or by looking for queries related to some specific query (via the query's ID). The primary key of the table includes both of the query IDs as well as the relation type. This allows the AMS to keep track of more than one relation between the same two queries.

The Data Level is also used primarily by the rule base and works in a similar way to the Inter-Query Level. It too consists of only one table. Each tuple consists of a pair of textual attributes that represent some type of data in the relational system (e.g. a table name, column name, etc.), a code that marks the data type of data and another textual string that designates the relation between the pair of data items. The rule system is able to use this table to store and retrieve information about relations between any pair of database entities by either looking for the type of entity or by looking for specific entities (e.g. all entries for TABLE-A).

The ACC and AMS Interface

The ACC is responsible for the control of the other three components of the AMS. Whenever the user performs any sort of action such as entering a word or pressing a function key, the AMS Interface passes this information on to the ACC which performs two primary actions: 1) It invokes the ARRB and gathers the results and 2) It makes any necessary changes to the Semantic level of the UMKB. Once these actions have been taken, the results are examined by the ACC to determine what set of information should be displayed to the user via the AMS Interface.

This information is presented to the user via a series of windows and menus. The user is first presented with a menu containing a list of the potential SQL commands he may choose (SELECT, INSERT, etc). After choosing one of these commands the Interface brings up a set of windows that represent that command. Each SQL command has three types of windows: the Initial window, the Current Choice window, and the Potential Choice window. Figure 2 shows the initial window for the SELECT command. The boxes represent fields that the user is allowed to move the cursor into. Each box represents one component of the command and map directly into the tables that represent the Semantic level of the UMKB. In other words, these boxes represent the major components of a complete SQL command.

When the user moves the cursor into one of these boxes, the system shows the user two additional windows (Figure 3). This first window (the Current Choices window) shows the user the choices he has already made for this component. The second window (the Potential Choices window) show the user the predicted choices that the AMS has chosen. The user may then pick choices from this window, or may type in a component on his own. In either case, the new component is added to the

current choices list and the user's choice is logged into the various locations of the UMKB for future reference. The user is allowed to move to and from any box in the Initial window. This means that the user is allowed to build the SQL command components in any order he wishes. For example he may choose Table names before he chooses Attributes, which is of course not valid in standard SQL.

When the user has built a SQL command, he can then send the command on to the database management system for execution. Since the AMS still retains the information about the last command, the user can then go back and make any changes to this command without having to reenter the entire query. Since the user can enter the components of a SQL command in any order, the AMS can use the ordering to its advantage. For example, if the user enters the Table names before he enters the Attribute names, the system will have more knowledge about what Attributes to predict. Once the user has filled in any appropriate components (enough for a complete SQL command) he may then have the command sent on to the RDBMS for execution.

SELECT FROM

WHERE

CONNECT BY START WITH

GROUP BY HAVING

ORDER BY

FOR UPDATE OF

Figure 2: Initial SELECT Window

SELECT FROM

WHERE

CONN START WITH

GROU HAVING

ORDE

FOR UPDATE OF

CURRENT CHOICES	POTENTIAL CHOICES
CITY	COLOR
COST	SIZE
LOCATION	TOTAL
AMOUNT	.
.	.
.	.
.	.

Figure 3: SELECT Window with Subwindows

The Abstract Relation Rule Base

The ARRB is the rule base that does most of the abstract reasoning with the user model stored in the knowledge base. It is based on a set of tables that represent information about the rules and some routines in the ACC that represent when and how the rules are invoked. The ARRB tables contain information about the rules, their premises and their resultant actions. Each rule is represented in three tables: the PREMISE Table, the ACTION Table and the RULE_SET table. In addition, each premise and action of every rule may have a resultant table associated with it.

Since every rule is made up of a set of premises and actions, these groupings are represented in the RULE_SET table. It contains four fields: RULE_NUMBER, PREMISE_ACTION_NUMBER, PREMISE_ACTION_FLAG and LOGICAL_CONNECTION. The primary key is the RULE_NUMBER, PREMISE_ACTION_NUMBER and the PREMISE_ACTION_FLAG. Each rule has a unique rule number and is made up of a set of premises and actions, represented by the PREMISE_ACTION_NUMBER. Premises are distinguished from ACTIONS by the PREMISE_FLAG. In addition, each premise has a logical connection between it and the other premises represented by the LOGICAL_CONNECTION attribute.

In the case of premises, the PREMISE_ACTION_NUMBER is used to reference the PREMISE table. This table is made up of several attributes which represent how the premise should be tested and where the results of that test will be located. The table has the following fields: PREMISE_NUMBER, PREMISE_TRIGGER, PREMISE_RESULT_LOCATION and PREMISE_RESULT_FLAG. The PREMISE_NUMBER is the primary key in the PREMISE table and (if it were supported) is a foreign key on the PREMISE_ACTION_NUMBER in the RULE_SET table. The PREMISE_TRIGGER contains the name of a user routine that should be invoked to do the processing that is required to see if the premise is valid. This routine will be invoked whenever the premise needs to be tested. In other words, whenever the rule system needs to test a given rule, the PREMISE_TRIGGER for each premise contained in that rule will be invoked. Also, the routine will be responsible for flagging either a success or failure in the PREMISE_RESULT_FLAG. If there is an associated table containing further results of the premise test, the routine is also responsible for placing any results in the table. The PREMISE_RESULT_LOCATION attribute contains the name of this table so that other functions may reference the results.

When an event causes the rule set to be invoked, the ACC clears the PREMISE_RESULT_LOCATION and begins firing the premises associated with the rules. It combines the results of the premises in the logical manner defined for each rule. If one of the sets of premises results in a true result, then the actions for that rule are invoked. These actions are executed in the same manner as the premise. The ACTION table contains the following fields: ACTION_NUMBER, ACTION_TRIGGER, ACTION_RESULT_LOCATION and ACTION_RESULT_FLAG. The ACTION_NUMBER is the primary key. Each action associated with a firing rule (the association is defined in the RULE_SET table) is invoked by having the routine name

stored in the ACTION_TRIGGER attribute executed. The routine is responsible for taking any action necessary, and returning a result into the ACTION_RESULT_FLAG and, if appropriate, results into the table defined in the ACTION_RESULT_LOCATION attribute.

The rule structure of the rule base system allows the AMS a great deal of flexibility, although this comes at the expense of speed. The premises and actions of rules are not limited by some specific rule syntax or even a specific language or set of possible events. In addition, if a defined action is used as a premise in more than one rule, it will only be invoked once per rule firing. Furthermore, defined actions can be used as both premises or actions. In the future, this could be the basis for a truly forward or backward chaining system.

The rule base makes extensive use of the UMKB and is allowed to modify both the Inter-Query and the Data levels of the KB structure, but it is only invoked if the ACC determines that it is necessary. In this way, the ACC can retain some level of control over the knowledge base. The rule base is expected to generate any appropriate query component predictions based on heuristic assumptions that are contained in the rules. These predictions are passed to the ACC along with weights that represent the likelihood of the correctness of each prediction. The ACC will compare these predictions with those of its own and choose the most likely candidates to display to the user.

The Use of the AMS Interface and Semantic Level

Before there are any queries made by the user to the database, the AMS Interface is able to represent the "bare bones" information necessary to support the query language syntax, but no semantic information. The three levels of the knowledge base are empty at this point. The tabular structure is there, but the tables contain no tuples.

As a user makes queries to the system, new tuples are added to the tables of the Semantic level, and the weights of the currently existing tuples are modified. As the information in the Semantic level accumulates, the ACC is able to make more accurate predictions. When the user begins a query, the ACC examines the information in the Semantic structure to help fill in the specific information necessary. The information is filled by taking the set of tables that represent the given command (e.g. SELECT, DROP, etc) and looking for patterns that match the current command's structure on a superficial level. For example, it can find what WHERE clauses were used in the past whenever the user selected information from a given column in a given table.

The system first determines what the syntactic layout of the command is likely to be and uses that layout to extract patterns from the Semantic level. Since all tuples have an associated weight, the system can assign a level of confidence in the patterns generated. As mentioned before, these ACC generated components predictions are compared with those generated by the ARRB. The predictions that have the highest weights are then shown to the user.

Conclusions

A prototype AMS system is currently being developed and evaluated. It is being implemented on an IBM compatible PC using the C language and a commercially available relational DBMS [Tanner 1989]. Early results of the user modeling look promising, although speed can be enhanced. The rule set stored in the ARRB is being examined and modified to find the best set for consistent prediction success. In the current system, both the AMS and the DBMS use the same processor. A migration of the system to a dual processor environment in which the AMS has a dedicated processor is planned for the future.

References

- [Blaha 1988] Blaha, Michael R., William J. Premerlani, James E. Rumbaugh, "Relational Database Design Using An Object-Oriented Methodology", *Communications of the ACM*, 31, 4, April 1988.
- [Brunner 1988] Brunner, K.P., R.R. Korfhage, "An Automatic Improvement Processor for an Information Retrieval System", *Proceedings from the Second International Conference on Expert Database Systems*, George Mason University, 1988.
- [Chakravarthy 1987] Chakravarthy, Upen S., Jack Minker, and J. Grant, "Semantic Query Optimization: Additional Constraints and Control Strategies", *Proceedings from the First International Conference on Expert Database Systems*, George Mason University, 1987.
- [Chen 1976] Chen, P.P. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems* 1,1. March 1976.
- [Codd 1972] Codd, E.F. "Relational Completeness of Data Base Sublanguages." In *Data Base Systems*, Courant Computer Science Symposia Series, Vol. 6. Englewood Cliffs, N.J.: Prentice-Hall 1972.
- [Date 1986] Date, C.J. *An Introduction to Database Systems*, Volume I, Addison-Wesley, Reading, Massachusetts, 1986.
- [Malley 1987] Malley, Christopher V. and Stanley B. Zdonik, "A Knowledge-Based Approach to Query Optimization", *Proceedings from the first International Conference on Expert Database Systems*, George Mason University, 1987.
- [Michalski 1983] Mickalski, R.S., J.G. Carbonell and T.M. Mitchell, *Machine Learning, An Artificial Intelligence Approach*, Morgan Kaufmann, Los Altos, California, 1983.
- [Michalski 1986] Mickalski, R.S., J.G. Carbonell and T.M. Mitchell, *Machine Learning, An Artificial Intelligence Approach*, Volume II, Morgan Kaufmann, Los Altos, California, 1986.
- [Minsky 1975] Minsky, Marvin, "A Framework for Representing Knowledge," in *The Psychology of Computer Vision*, edited by Patrick H. Winston, McGraw Hill Book Company, New York, 1975.
- [Tanner 1989] Tanner, Steve and Sara J. Graves, "*Modeling User's Access to the Oracle Relational Database Management System*", *Proceedings of the Oracle International User Week*, Dallas, Texas, 1989.
- [Visetti 1987] Visetti, Y.M., "Plan inference and student modeling in ICAI", *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, Washington, 1987.
- [Winston 1984] Winston, Patrick Henry, *Artificial Intelligence*, Addison-Wesley, 1984.
- [Woolf 1987] Woolf, Beverly, "Intelligent Tutoring Systems: A Survey", Invited talk given at the Sixth National Conference on Artificial Intelligence, Seattle, Washington, 1987.

**RESOLUTION OF SEVEN-AXIS MANIPULATOR REDUNDANCY-
A HEURISTIC ISSUE**

I. Chen

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

ABSTRACT

This paper presents an approach for the resolution of the redundancy of a seven-axis manipulator arm from the AI and expert systems point of view. This approach is heuristic, analytical, and globally resolves the redundancy at the position level. When compared with other approaches, this approach has several improved performance capabilities, including singularity avoidance, repeatability, stability, and simplicity.

1. INTRODUCTION

With an addition of one more degree of freedom (d.o.f.), a yaw motion of the forearm, to the typical six d.o.f. articulated arms, the shoulder of the three-link manipulator will have pitch, yaw, and roll motions (Figure 1). These seven-axis articulated manipulator arms when compared with the typical six d.o.f. robot arms may have many advantages, including flexibility, manipulability, obstacle avoidance, singularity avoidance, stability, and optimal control. In addition, because of the similarity to human arm configuration, the seven-axis, three-link manipulator arm is the best candidate for the master-slave teleoperated robot useful in hazardous environments. However, the resolution of the kinematics, control, and dynamics with redundancy is not an easy task. With the redundant d.o.f., there will be an infinite number of arm configurations kinematically for each desired hand position. The motion that maximizes a specific performance index is the optimal motion for that special condition. However, the optimal solutions are not easily accessible due to a highly nonlinear relationship between the joint angle space and the hand position. Because of the complex nature of this problem, each approach seems to have its own drawbacks and limitations. In order to resolve this redundancy with the already complicated problem, the author employed ([5], [6]) human arm motion heuristics to analytically derive the

Support for this research is provided by a grant NAG 023 from NASA Marshall Space Flight Center.

inverse kinematics (see the Appendix). The purpose of this paper is to present this heuristic concept and to compare the approach with other existing methods.

2. STATEMENT OF THE APPROACH

By observing the human arm motions, two heuristic rules are concluded:

- 1) The travel distance of the wrist joint from a hand position to a new hand position should be a minimum for each arm motion.
- 2) The travel distance of the elbow joint from an arm position to a new arm position should be a minimum for each arm motion.

In addition, there are two meta-level rules:

- 1) The first rule can be applied only when the second rule is satisfied.
- 2) If the movement of the robot arm violates the joint limit constraints or obstacle constraints, then a configuration which satisfies the above two domain specific rules and the constraints is the solution.

The first two rules state the fact that both the joint travel distances have to be minimized in order to obtain the kinematic solution. The first meta rule defines the relationship between these two domain-specific rules. The first rule is subject to the constraint of the second rule, and the first rule will be applied only when the second rule is satisfied. The purpose is to minimize the use of the lower arm as much as possible since each movement of the lower arm causes the movement of the forearm and hand together. This consumes more energy because a bigger moment of inertia is involved in the movement. The same principle is also applied to the movement of forearm and hand. In other words, the travel distance of the wrist joint should also be maintained to a minimum to ensure the minimization of the energy consumption. The main idea is to minimize the lower link of the manipulator since the motion of the lower link causes movement of all the upper links together. A bigger movement of the lower link will in turn consume more energy because of the bigger moment of inertia involved in the motion. The motion with the least amount of energy consumption is the most comfortable to the body. Besides, for a given kinetic energy, the optimal arm motion which obeys these rules is the minimum-time motion and is the fastest way to reach the new hand position. The second meta rule governs the condition whenever the arm motions are prohibited because of the violation of the joint limit constraints or obstacle constraints. Although this strategy has not yet been developed, a suboptimal configuration, which relaxes the constraint free minimum-distance solution according to the imposed constraints, is accessible. Note that from the point of view of AI and expert systems the second meta rule is of second-order since it governs all the rest of the

rules, including the first meta rule. This relaxation strategy may be an iterative process by relaxing the related joint angles to the subsequent neighbors according to a prescribed small joint interval.

The motion control of this seven-axis manipulator arm may be restated as follows:

Minimize the following performance index,

$$J = K_1 * D_1^2 + k_2 * D_2^2 \quad (1)$$

subject to the kinematic equations of the manipulator arm and the joint limit and obstacle avoidance constraints.

Here, D_1 is the elbow joint travel distance and D_2 is the wrist joint travel distance. K_1 and K_2 are the corresponding weighting factors with $K_1 \gg k_2$ to make sure that the minimization of the travel distance of the elbow joint D_1 before minimizing the travel joint distance of the wrist, D_2 . The squares of D_1 and D_2 assure that D_1 and D_2 will be positive throughout the converging process.

3. COMPARISONS

This approach is compared to other approaches in the following manner:

3.1 INVERSE KINEMATIC METHOD VS. RESOLVED MOTION METHOD

Research has been carried out on the control of redundant arms mostly through a pseudoinverse matrix, also known as the Moore - Penrose generalized inverse. The instantaneous joint displacements are computed from the joint velocities by using the pseudoinverse J^+ of the Jacobian matrix J . For a given hand position, the resolved motion methods cannot provide the corresponding joint angles. This implies that the resolved motion technique cannot directly map the workspace to the joint angle space of the arms. In other words, the approach is numerical and not analytical. Because of the instantaneous resolution of the redundancy, this type of technique also inherits other drawbacks (to be discussed later). The proposed approach has advantage over the resolved motion methods by resolving the redundancy analytically and globally (not incrementally). Benati's approach [2] is recursive, partly analytical, and partly numerical. The application of the heuristics rules to the derivation of the inverse kinematics of a seven-axis manipulator arm is shown in the Appendix.

3.2 RESOLUTION LEVEL

This approach resolves the redundancy at the position level rather than resolving the redundancy at the velocity level or at

the acceleration level.

The resolved motion method has many interesting characteristics [4]. A desired performance criterion function can be incorporated in the general solution for the avoidance of joint limits ([13], [17]), the improvement of the repeatability for repeated end effector motions ([1], [11]), and the obstacle avoidance in work space (e.g. [7], [12], [14], [17]). Another advantage is the least square property [3] of the pseudoinverse method which minimizes the sum of squares of joint velocities which in turn approximately minimizes kinetic energy. However, this approach has an intrinsic inaccuracy because of the error accumulation of the linear approximation of the Jacobian matrix. Therefore, lack of repeatability is the major drawback of this method (e.g. [4], [8]). The approach involves the instantaneous resolution of the redundancy at the velocity level where the sum of the squares of the joint velocities is minimized. This means that the kinetic energy is approximately minimized. This method kinematically resolves the redundancy at the velocity level. Chang [4] proposed a method called the criterion function method which has resulted in improved repeatability for end effector motions. Many researchers (e.g. [8], [10], [16]) extended the kinematic resolution method from the velocity level to the acceleration level by incorporating the generalized inverse into dynamics. While this method resolves the redundancy at the kinetic level, it may lead to stability problems. Local tampering with the energetics of movement has led to global disaster [8].

3.3 SINGULARITIES AND AVOIDANCE

Some arm configurations are singular where the joint angles or instantaneous joint velocities are impossible to realize. For the resolved motion approach or other approaches, the joint velocities for some manipulator configurations sometimes approach mathematical infinity in the derivation. The joint velocities that close to the singular points are also too large and are very difficult to realize. This establishes forbidden regions around the singular points. A substantial fraction of the workspace is lost and the degree of freedom of the manipulator is functionally reduced ([9], [15]).

Unlike other approaches, no singularity and singularity avoidance consideration are necessary for this approach since the resolution of redundancy is at the position level, and joint angles are determined globally not incrementally (Refer to the Appendix.) This approach is a geometrical approach where only three inverse matrices appear in the derivation, two rotational inverse matrices, R_z^{-1} and R_x^{-1} , and one translation inverse matrix, T^{-1} . These three are nonsingular matrices since their determinants are non-zero.

That no singularity exists in this approach implies improvement of the performance capabilities of the manipulator arm because singular points functionally reduce manipulator degree of

freedom. Besides, there is no need to consider the singularity avoidance in the control or arm movement planning.

3.4 COMPLICATION OF APPROACH

Another major attraction of this approach is the simplicity of the derivation (Refer to the Appendix.) The resolution of the redundancy at the position level does not require incorporation of the dynamics, pseudoinverse, etc. in the derivation. Compared to other approaches, the derivation is drastically reduced. In addition, this approach is analytical and geometrical. This means that the resolution of the redundancy is not incremental or local, but global. Thus, control and trajectory planning of the manipulator are much more simplified than those of the other approaches.

3.5 OPTIMALITY

This approach geometrically applies the heuristic rules for the resolution of the redundancy. For the resolved motion method, the kinetic energy is approximately minimized; however, it has the drawback of poor repeatability. Other extended methods of this type which attempt to realize the energy minimization have other problems also, such as stability. It would be a significant research task to verify the assertion that heuristic application of these rules does result in minimization of the energy consumption for manipulator arm movements.

4. SUMMARY AND CONCLUSION

Resolution of seven-axis manipulator redundancy is a very interesting and important research topic. Unlike most of the approaches, the redundancy is resolved through the implementation of heuristic rules in the derivation. It seems to the author that this approach resolves the very complicated seven-axis redundancy very easily and has no known drawbacks. In addition to this, although yet to be verified theoretically, this approach may have the most desirable feature of true minimization of the kinetic energy. Since the resolution of the redundancy is at the position level, the corresponding joint angles are derived analytically and geometrically. Thus, the resulting joint angles are determined globally not incrementally. This implies that, unlike other approaches, the joint angles for a given hand position are given directly or the joint angle space is a direct mapping from the workspace. Another very significant advantage of the approach is that there is no singular point in the derivation. Therefore, compared with other approaches, this approach has a greater degree of freedom functionally and has no singularity avoidance problem in the workspace.

5. APPENDIX

Since the forward kinematics for redundant arms is always feasible, these heuristics are applied only to the derivation of inverse kinematics for the seven-axis anthropomorphic arms.

According to the arm motion characteristics as described above, the inverse kinematics of a two-link manipulator will be derived. The two-link strategy for lower arm and forearm is then extended to a three-link strategy with hand included ([5], [6]).

5.1 THE TWO-LINK STRATEGY

According to Figure 2, the arm positions are represented by the following three points: the origin of the coordinates $P_0(0,0,0)$, $P_1(x_1, y_1, z_1)$ and $P_2(x_2, y_2, z_2)$. The point $P_2'(x_2', y_2', z_2')$ represents the desired position. When P_2 reaches P_2' , the collection of all possible locations of P_1 forms a circle (denoted by C) with center at $P_c(x_c, y_c, z_c)$ and radius r . The objective is to find the point $P_1^*(x_1^*, y_1^*, z_1^*)$ in C such that the distance between P_1 and P_1^* is minimized. Homogeneous coordinates should be employed to simplify the required calculation. First of all, $P_c P_2'$ coordinates of P_1^* with respect to the original coordinates.

$$[x_1^*, y_1^*, z_1^*, 1]^t = T^{-1} R_z^{-1} R_x^{-1} [x_1^{\wedge}, y_1^{\wedge}, z_1^{\wedge}, 1]^t$$

5.2 THE THREE-LINK EXTENSION

By referring to Figure 3, let $L_p =$ distance (P_0, P_3^*). If $L_p > l_1 + l_2 + l_3$ or, $L_p < l_1 - (l_2 + l_3)$ and $l_1 > l_2 + l_3$, then there is no solution. Evaluate $d_1 = \text{ABS}(l_2 - l_3)$, $d_2 = l_2 + l_3$, and, $D =$ distance (P_1, P_3^*), If $d_1 \leq D \leq d_2$, then do not move P_1 , i.e., $P_1 = P_1^*$. Apply the "two-link" method (Section 5.1) to find P_2^* . If $D < d_1$, then apply the "two-link" method with length of link₁ = l_1 and length of link₂ = $\text{ABS}(l_2 - l_3)$ or d_1 to find the new location P_1^* . If $D > d_2$, then apply the "two-link" method with length of link₁ = l_1 and length of link₂ = $l_2 + l_3$, or d_2 to find new P_1^* .

Because the joint limits are not imposed, $P_1^* P_2^* P_3^*$ is always in a straight line for the last two cases..

6. REFERENCES

- [1] J. Beillieul, J. M. Hollerbach, and R. Brockett, "Programming and Control of Kinematically Redundant Manipulators," 23rd IEEE Conf. on Decision and Control, Dec, 12-14, 1984.
- [2] M. Benati, P. Morasso, and V. Tagliasco, "The Inverse Kinematic Problem for Anthropomorphic Manipulator Arms," ASME J. of Dynamic Systems, Meas., and Control, Vol. 104, pp. 110-113, Mar. 1982.
- [3] A. Ben-Israel, and T. N. E. Greville, "Generalized Inverse Theory and Applications," New York, Robert E. Krieger Publishing Co.

07

- [4] P. H. Chang, "A Closed-Form Solution for the Control of Manipulators with Kinematic Redundancy," IEEE Int'l Conf. on Robotics and Automation, pp. 9-14, 1986.
- [5] I. Chen, T. C. Chen, and H. Saha, "Motion Simulation of an Anthropomorphic Arm," Intelligent Autonomous Systems, Eds. Hertzberger and Groen, North-Holland Pub. Co., pp 151-156, 1987.
- [6] I. Chen, T. C. Chen, and H. Saha, "A Strategy for Deriving Inverse Kinematics for a Seven-Axis Manipulator Arms," 17th Southeastern Symposium on System Theory, pp 124-127, March 24-26, 1985.
- [7] H. Hanafusa, T. Yoshikawa, and Y. Nakamura, "Analysis and Control of Articulated Arms," XIV-78- 83, Aug. 1981.
- [8] J. M. Hollerbach, and K. C. Sub, "Redundant Resolution of Manipulators through Torque Optimization," IEEE Int'l Conf. on Robotics and Automation, pp. 1016-1021, Mar. 25-28, 1985.
- [9] J. M. Hollerbach, "Optimal Manipulator," 2nd Int'l Symp. on Robotics Research, Kyoto, Japan, Aug. 20- 23, 1984 .
- [10] O. Khatib, "Dynamic Control of Manipulators in Operational Space," 6th Congress on Theory of Machines and Mechanism, Dec. 1983, New Delhi.
- [11] C. Klein, and C. Huang, "Review of Pseudoinverse Control for Use with Kinematically Redundant Manipulators," IEEE Trans. on System, Man and Cybernetics, Sol . SMC-13, pp.245-250, 1983.
- [12] C. A. Klein, "Use of Redundancy in the Design of Robotic Systems," Preprints of the 2nd Int'l Symp. of Robotics Research, Kyoto, Japan, pp. 58-65, Aug. 20-23, 1984.
- [13] A. Liegeois, "Automatic Supervisory Control of the Configuration and Behavior of Multibody Mechanisms," IEEE Trans on System, Man and Cybernetics, Vol. SMC-7, No. 12, 1977.
- [14] Y. Nakamura, and H. Hanafusa, "Task Priority Based Redundant Control of Robot Manipulators," Preprints of the 2nd Int'l Symp. of Robotics research, Kyoto, Japan, pp.357-364, August 20-23, 1984.
- [15] R. P. Paul, and C. N. Stevenson, "Kinematics of Robot Wrists," The Int'l Journal of Robotics Research, Vol. 1, No. 2, 31-38, 1983.
- [16] M. Vukobratovic, and M. Kircanski, "A Dynamic approach to Nominal Trajectory Synthesis for Redundant Manipulators," IEEE Trans. Systems, Man, Cybern., SMC-14, pp. 580-586, 1984.
- [17] T. Yoshikawa, "Analysis and Control of Robot Manipulators with Redundancy," The 1st Int'l Symp., ed. Brady, M. and Paul, R., Cambridge, Mass., MIT Press, pp.735-748, 1984.

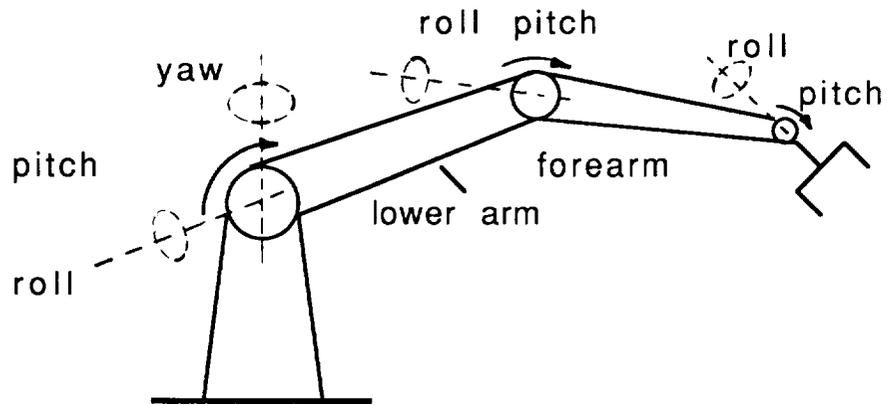


Figure 1. An anthropomorphic arm

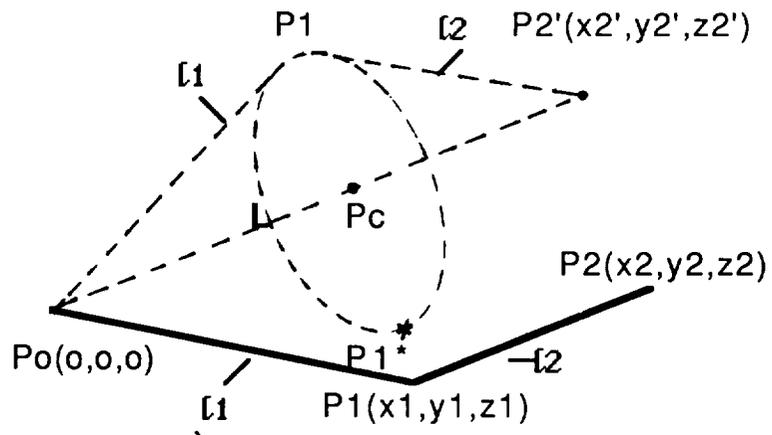


Figure 2.

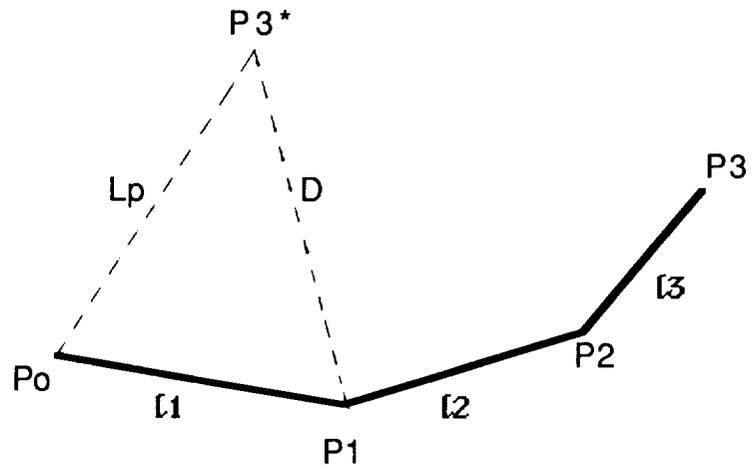


Figure 3.

THE SIXTH GENERATION ROBOT IN SPACE

A. Butcher, A.Das, Y.V.Reddy & H.Singh

Artificial Intelligence Laboratory
Dept.of Statistics & Computer Science
West Virginia University
Morgantown, W.V. 26506

Abstract

The knowledge based simulator developed in the artificial intelligence laboratory at the West Virginia University has become a working test bed for experimenting with intelligent reasoning architectures. With this simulator, recently, small experiments have been done by the authors' group with an aim to simulate robot behavior to avoid colliding paths. An automatic extension of such experiments to intelligently planning robots in space demands advanced reasoning architectures. This paper explores one such architecture for general purpose problem solving. The robot, seen as a knowledge base machine, goes via predesigned abstraction mechanism for problem understanding and response generation. The three phases in one such abstraction scheme are : (i) abstraction for representation, (ii) abstraction for evaluation, and (iii) abstraction for resolution. Such abstractions require multimodality. This multimodality requires the use of intensional variables to deal with beliefs in the system. Abstraction mechanisms help in synthesizing possible propagating lattices for such beliefs. The machine controller enters into a sixth generation paradigm.

1. INTRODUCTION

" All new technologies develop within the background of a tacit understanding of human nature of human work. The use of technology in turn leads to fundamental changes in what we do, and ultimately to what is to be human. We encounter the deep questions of design when we recognize that in designing tools we are designing ways of being. By confronting these questions directly, we can develop a new background for understanding computer technology --- one that can lead to important advances in the design and use of computer systems." This is a quotation from Winograd and Flores (6).

The already announced program for the sixth generation world of computation (4) actually touches the basic content of the above equation. In one form of expression (3), the sixth generation computational activities of the next few years explores learning and emotion, parallel knowledge systems, audio and visual sensors with multimodal modeling. In this game, part of the supporting hardware is optical storage and logic, organic processor elements and routine engineering and maintenance of full-sized knowledge based systems.

This paper attempts to project views on a sixth generation robot. The robot is intelligent to generate plans and take actions independently. The original idea comes from some

computational attempt made in this school's artificial intelligence laboratory to simulate robot behavior while avoiding colliding paths. The programming environment is a knowledge based simulation system developed earlier (1). In this system of knowledge based simulation frames have been used to represent objects and their relationships, and rules to represent procedural behaviors of objects. Model making in this system becomes explicit, understandable, modifiable and self-explanatory. By using a frame language to represent domain concepts such as object structure and goals, there is a one-to-one correspondance between the domain and the simulation model Also, by using rules to represent behavior, the specification and modification of the behavior become easier. Explanation generation techniques developed around the rule based system provide basis for explaining event behaviors. Recently, in the artificial intelligence laboratory of in this school attempts are being made to understand the process of encoding objects in frames side by side in terms of intensional and extensional variables. The extensional approach is not new. It is also known as rule based, procedure based or production systems. The well-known predicate calculus logic works here strongly. The other system, the intensional system is just the opposite. Here, non-predicate calculus logic enters automatically. We deal with *possible worlds*. Objects in frame carry *beliefs* as has been suggested for the sixth generation computations. They arise because of uncertainties present in the computation. It is well-known that intensional systems are semantically clear but computationally clumsy. It is proposed that for a smart robot capable of intelligent decision making and plan generation, intensional approach is worth attempting. In the following sections very briefly is described how the world appears in the eye of one such sixth generation robot. Also, this robot's smooth living with intelligent decision making and plan generation is aided by a three-phase abstraction hierarchy process. And, explanations have been provided on how such abstractions help the robot in simulating its actions in an uncertain world full of *plausibility*.

2.THE WORLD IN THE EYE OF THE ROBOT

To the robot, the world is always in a particular state at a particular time. A series of such states will appear as history. The robot is capable of witnessing and recording such world states. The robot's knowledge base classifies history in the following form:

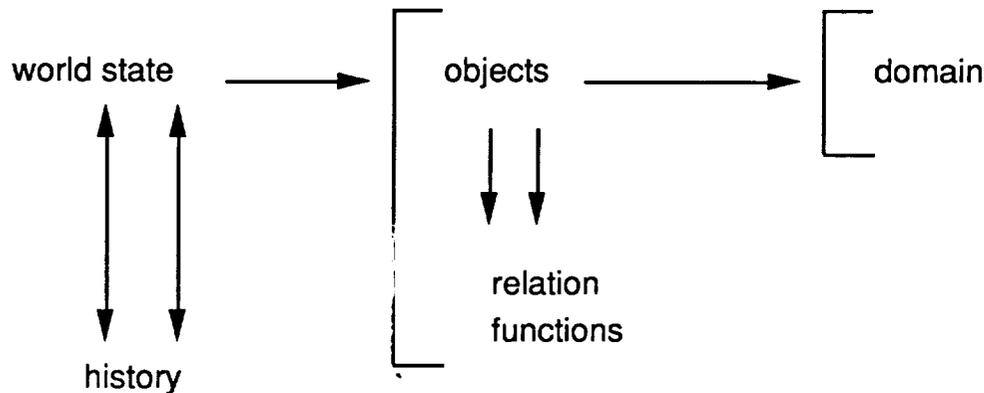


Fig.1

The world identifies a particular world state (s) by an event (e). This identification is characterized by a belief function which has a well-maintained architecture of propagation. For example, let there be two belief functions Z1 and Z2 characterizing two world states W and s. If Z1 is true when event occurs, then Z2 will be true in the resulting states. That is,

$$\forall W, S. Z1(s) \wedge OCCURS(e)(s) \supset Z2(SUCC(s))$$

In the present case, belief functions Z1 and Z2 are actually characteristics of predefined metalevel analysis before indexing a world state by a belief function. The overall architecture is as follows:

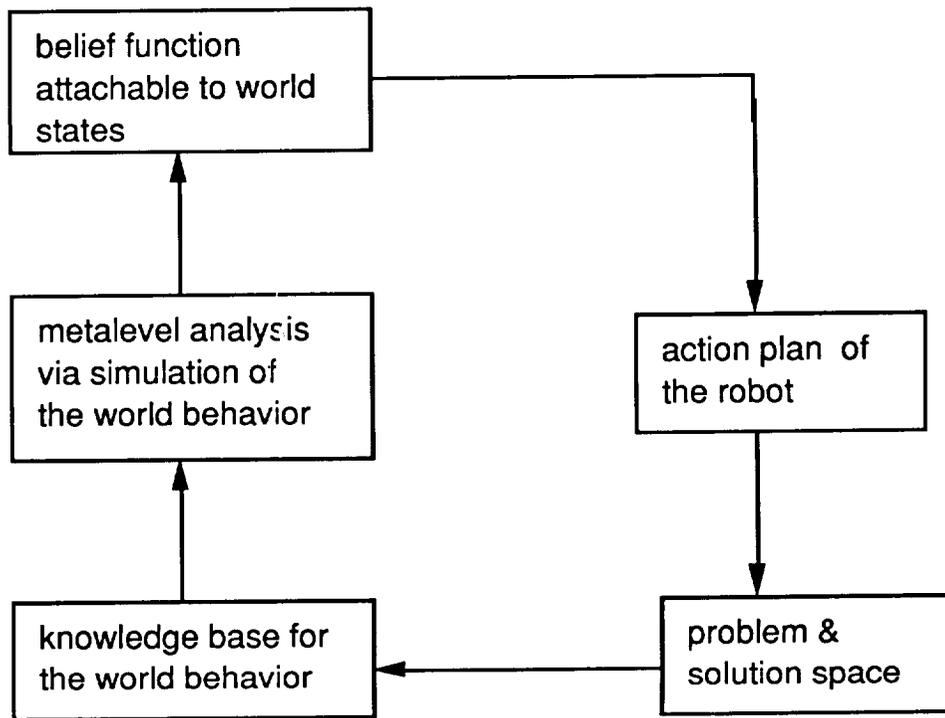
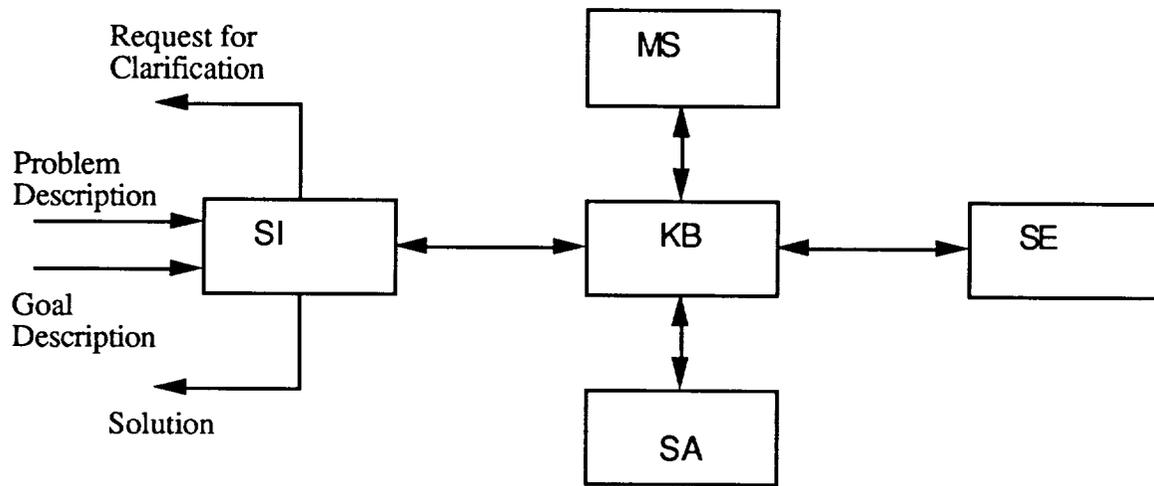


Fig. 2

The present author's group has implemented a substantial amount of work on the metalevel analysis via simulation of the world behavior. The work is being implemented in an object oriented knowledge based system highlighting encapsulation and inheritance (5). Problem formulation, model building, data acquisition, model translation, verification, experiment planning, analysis of result ---- all have been treated in the program with an aim to bring a closed world subject to the domain of an open world reasoning process. Figure 3 describes the implemented knowledge based simulation environment (1).

In the simulation environment described in figure 3, the user will describe the system he or she wishes to model as well as the results he expects from the simulation interface (SI). The interface will translate this description into a representation used internally. The

representation will also indicate what results to measure while the model is executed as well as which of these results to display graphically. The model synthesizer (MS) will interpret



KB : Knowledge Base
 MS : Model Synthesizer
 SA : Simulation Analyzer
 SE : Simulation Engine
 SI : Simulation

Fig.3

the scenario representation into the simulation target language and build a simulation model. Once the simulation engine (SE) has executed the model, the model analyzer will translate the results of the simulation back into the representation of the results to determine if the goals have been satisfied. If they have, then the simulation analyzer reports back to the simulation interface, which in turn, translates the results to the user. If the goal has not been satisfied, then the simulation analyzer will make modifications to the model by creating one or more scenarios. This process continues until the desired goals are met or some time limit has reached.

3. THREE PHASES IN ABSTRACTION HIERARCHY

To survive in one such aforementioned world of events and states, the robot needs a good planning mechanism. Such mechanisms are supported by a strong program in the science and craft of knowledge base development. It has been sketched below.

To create a knowledge base appropriate discipline needs to be followed that preserves consistency. Obviously, knowledge will come from distributed sources. Retrieval and modification thus become complex. Encoding facts, beliefs and relationships with structured learning should be generated too. Any action, taken by the robot, then is done from a flexible and accessible repository of the shared knowledge of various participants. In order to achieve this, an architecture of the three-phase abstraction process hierarchy will be narrated now. These three hierarchies are : representation, evaluation and resolution.

Phase-1:Representation

The robot is essentially a knowledge base machine. Knowledge acquisition for such a machine is very difficult. A very stringently tested policy must, therefore, be guaranteed for this. In this phase-1 level, the robot is able to generate a series of reasoning process. This is based upon an automatic problem recognition and model based approach to the solution of the problem. In the current system, for knowledge representation, a network of schema representation language is used. This represents the physical/abstract entities of the system to be modeled. Past research in this laboratory showed that, this representation scheme provides a good simulation environment in addition to simulation. The robot is waived of maintaining multiple models which eliminates extra cost.

Phase-2:Evaluation

Before adventing any action, the robot needs an assessment of possible alternative items of their relative applicability. This has been facilitated mostly because of the use of object oriented approach to model representation. Here, objects have one-to-one correspondance with domain entities and they have methods explaining their behavior. This has provided flexibility in creating and altering entities and their behavior without altering the simulation model interpreter.

Phase-3:Resolution

In this abstraction, the robot is trying to design the most appropriate action under certain specified goals and constraints. The specification on constraints is achieved by simulation. One possible architecture is shown in figure 4.

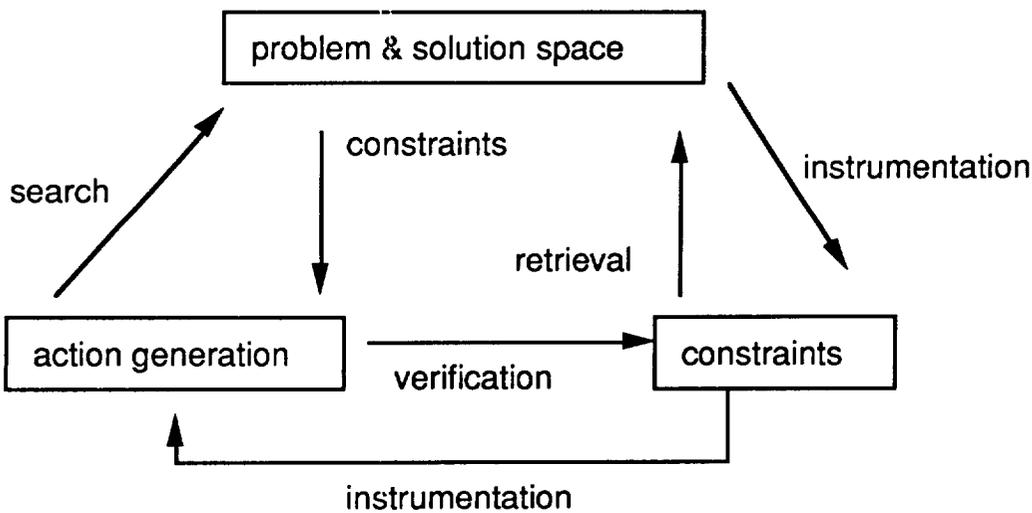


Fig.4

The above phase-3 architecture simulates a strong knowledge system support for intelligent plan generation by robots.

The three steps in abstraction helps in updating and revision of belief which is necessary to strengthen evidential support. This establishes a strong goal directed reasoning mechanism for the knowledge based system.

4.CONCLUSION

The whole environment of knowledge based simulation comprises of a sequence of experiments, where, each experiment measures how well a scenario (i.e., altered version of an original model) optimizes one or more goals. The purpose of introducing a three phase abstraction scheme is to let the robot justify, rectify, analyze, assemble its own beliefs before planning an action leading to a goal. This attitude is expected from a sixth generation robot. It, while taking actions, begins simulations first with the specification of a set of goals which result in the instrumentation of a scenario in order to gather data. Uncertainties in data are dealt with intensional mechanism, with beliefs as constraints on the simulation trajectory. Currently, we are in search of effective instrumentations to manage these constraints appropriately.

Acknowledgements: The work has been supported by a grant from SDC-USArmy(Huntsville,Alabama), contract # DASG60-86-0094. It, however, refelects the authors' own view. The authors are grateful to Kachesh Pathak of SDC for inspiring an investigation in this typical problem.

REFERENCES

- [1] Documentation on Knowledge Based Simulation,The, AI Lab, W.V.University, Morgantown, West Virginia, 1989.
- [2] Fiksel, J. & Hays-Roth, F., Knowledge Sysrtems For Planning Support, IEEE Expert, Vol.4,No.3 (1989) pp16-23.
- [3]Gaines, B.R. & Shaw, M.L., From Time Sharing to the Sixth Generation : the Development of Human-Computer Interaction, Pt.1, Int.Journ.Man-Machine Studies,Vol.24(1986) pp 1-24.
- [4] Promotion of R&D on Electronics & Information Systems That May Compliment or Substitute for Human Intelligence, Tokyo: Science & Technology Agency, 1985.
- [5] Reddy, Y.V., Fox,M.,Husain, N. and McRoberts, M., The knowledge - Based Simulation System, IEEE, March(1986) pp 26-37 (and refernces cited therein)
- [6] Winograd, T. and Flores, F. in Understanding Computers and Computation : A New Foundation for Design, Albex Pub. Co., N.J. (1986) p xi.

ROBOT DYNAMICS IN REDUCED GRAVITY ENVIRONMENT

Gary L. Workman and Tollie Grisham
University of Alabama in Huntsville
and
Elaine Hinman and Cindy Coker
Marshall Space Flight Center

ABSTRACT

Robot dynamics and control will become an important issue for productive platforms in space. Robotic operations will be necessary for both man tended stations and for the efficient performance of routine operations in a manned platform. The current constraints on the use of robotic devices in a microgravity environment appears to be due to safety concerns and an anticipated increase in acceleration levels due to manipulator motion.

The robot used for the initial studies was a UMI RTX robot, which was adapted to operate in a materials processing workcell to simulate sample changing in a microgravity environment. The robotic cell was flown several times on the KC-135 aircraft at Ellington Field. The primary objective of the initial flights was to determine operating characteristics of both the robot and the operator in the variable gravity of the KC-135 during parabolic maneuvers.

This study demonstrated that the KC-135 aircraft can be used for observing dynamics of robotic manipulators. We also observed the difficulties associated with humans performing teleoperation tasks during varying G levels and can provide insight into some areas in which the use of artificial techniques would provide improved system performance.

Additionally a graphic simulation of the workcell was developed on a Silicon Graphics Workstation using the IGRIP simulation language from Deneb Robotics. The simulation is intended to be used for predictive displays of the robot operating on the aircraft. It is also anticipated that this simulation can be useful for off-line programming of tasks in the future.

INTRODUCTION

Most texts and papers dealing with kinematics and dynamics of robots assume that the manipulator is composed of joints separated by rigid links. However; in recent years several authors have published papers dealing with the dynamics of flexible manipulators, particularly for the application of robots in space.^{1,2} Additional concepts which have to be worked out in any robotic implementation for a space platform include teleoperation and degree of autonomous control.³⁻⁵

Lightweight arms are necessary for space, primarily for the costs benefits derived from their reduced weight.⁶ However, lighter weight arms have to necessarily flex during movement. Flexure of the arms performing a task requiring precision requires some control mechanism to insure that the end-effector is at the proper place and orientation with respect to the workpiece. Flexing motions of the arm will cause (1) accelerations to feed back into the base support of the robot, (2) transmit accelerations into the sample being transported, or (3) take forever to perform a task. The first effect will obviously destroy the microgravity environment of the Space Station, while the second will impact experiments such as the delicate protein crystals which are to be grown in space. In some cases slow movements may be acceptable for (3); however it certainly will not be suitable for all tasks aboard the Space Station.

One must also include the reasoning that for man and robots to co-exist in the space environment, the robot must be non-threatening to man. Lightweight arms are the only ones which satisfy that criteria. For space applications, a Cincinnati-Milacron T3 is not only overweight, but also may be threatening to humans trespassing in its working volume. Artificial Intelligence will also have to provide a major role for robots to enter into the space activities. Current systems such as the Flight Teleoperator System (FTS) and the Orbital Maneuvering Vehicle (OMV) will certainly develop some intelligence with time.⁷ Allowing some degree of autonomy due to time delay communications for teleoperation over large distances is necessary. The robot controls then will have certain motions embedded in the control software that do not need explicit operator communication, except for emergency control, such as ABORT. AI will have to be part of the embedded software control. Time delays make for precarious situations in performing teleoperation from large distances. Tasks may be accomplished in a more reasonable manner and more success using some AI techniques.

This study was initiated in response to the need to determine the adverse parameters which might exist for robotic/telerobotic operation in a microgravity environment. The first part of this study has been to build up hardware to perform reduced gravity observations on a small robotic arm and to develop a graphical simulation of the robotic workcell. We anticipate combining these two efforts at a later time for predictive display capability in a remote control environment. Also at that time we will begin to develop control strategies using AI for remote teleoperation. The initial experiments will be performed on the KC-135 aircraft since TDRSS satellite telemetry will soon be available to onboard experiments.

REDUCED GRAVITY EXPERIMENTS

A major goal was to evaluate a small robot system, such as the UMI RTX, for materials processing applications in low gravity and determine the characteristics of a robot arm in a space environment, particularly with respect to accelerations which might impact materials grown on a space platform.

A materials transfer workcell was assembled to simulate the changing of sample ampoules as might occur aboard a space laboratory. Accelerometer packages were included for determining

the G levels of the workcell and the at the end-effector. Several flights were taken with the workcell, improving some data taking capabilities each time. One experimental concern was whether a short enough, but meaningful task could be developed. The task needed to take no longer than around 22 seconds to fit within the microgravity portion of the parabolas. However, on each flight somewhere between 25 and 40 parabolas are flown allowing a short task to be repeatedly run and measured, or a long task possibly to be broken into consecutive steps.

The task for the first flights was a simple ampoule exchange from one rack to another. During the higher gravity portion of the parabola the robot could be reset, and another task run readied. The human operator was able to train the robot to perform a materials transfer function within the 20 seconds desired. The first computer used with the experiment did not allow for both control of the robot and reading of the accelerometer package at the same time. The next upgrade was to a 386 based computer, which enabled some improvements in the data acquisition process; but the multi-tasking software used at that time still did not permit the I/O commands to the robot to operate properly. Consequently we never did get to control the robot and take acceleration data simultaneously in these flights. Future task upgrades should allow this to be done.

A number of lessons were learned with this series of experiments. The RTX robot uses plastic belts for actuation of the links and optical encoders for position and velocity control. The slippage and flexing of the belts caused excessive jitter and accelerations at the end-effector. We believe that the belt-driven actuation would not be acceptable for experiments such as the protein crystal growth studies due to the lack of control of accelerations at the end-effector. The control system; however, is PID and appeared to work well whether the task was learned in 1 G and performed in low G or vice-versa. However, it was tedious to teach the robot during parabolas, mainly because we had few visual aids to assist in the correct orientation of the end effector; particularly for inserting the sample ampoule into its holder. A borrowed fiber-optic borescope provided little depth perception and was not useful for this study. In addition teaching a robotic device for precision movements can certainly be improved through more innovative approaches using embedded sensors or vision systems with some autonomous local control.

A major concept which might be important in terms of promoting telescience experiments to use the KC-135 would be to implement the above experiments using remote manipulation from the ground. The KC-135 aircraft facility at Johnson Space Center has indicated interest in these types of experiments using the TDDRS satellite for transmission purposes. Teaching the robot remotely and then performing the task from the ground would be an ideal simulation for space teleoperation.

IGRIP SIMULATION RESULTS

The RTX robot workcell was modelled on a Silicon Graphics IRIS work station using the IGRIP off-line programming language and simulation tools developed by Deneb Robotics. The simulation was generated with version 1.0. Later versions offer some enhancements which were not available in our simulation. In fact no off-line programming is actually available at the

moment; however, it can be performed with later versions and would certainly be a necessary tool for the space-based activities.

In IGRIP, a geometric model is designed using through the creation of individual parts. These parts are attached in relation to each other and saved as a device. As a device, several non-geometric properties are entered, such as kinematics, velocities, and acceleration. The final workcell consists of the robot model and the workcell model.

A PART is made up of one or more OBJECTS. Simple OBJECTS are created through the use of cad primitives. Examples of Cad primitives used on the RTX and workstation model are the BLOCK, WEDGE, and CYLINDER. Additional OBJECTS are made through the use of the MIRROR and CLONE commands. OBJECTS which are connected and move together are saved as a PART.

The PARTS are assembled in the CREATE DEVICE module by invoking the NEW DEVICE command and specifying the number of degrees of freedom. Beginning with the base, each PART is attached to the previous PART and translated (x,y,z) or rotated (roll, pitch, yaw) to its new position. Once all PARTS are in place, it is saved as a DEVICE.

The ATTRIBUTES of the DEVICE are defined next. LINK TYPES, LINK LIMITS, Max Velocity, and Max acceleration are then entered. The Degrees of Freedom are specified by choosing the axis for rotation or translation corresponding to each link. Selecting the KINEMATICS command gives the option of jogging by links or through a kinematics routine. The RTX model has a user-kinematics routine defined.

In the CREATE WORKCELL module, the models of the robot and workstation are placed together in relation to one another and saved. It is here in the workcell that a device is programmed. With the user-defined kinematics specification, a sequence of entered link movements are saved in a program. The MOVE command enables the selection of a link and input as to the translational or rotational distance. An option is available as to whether or not the link movement will be simultaneous with other link movements. The program is then saved for later recall.

To view the program, one uses the LOAD command. The DEVICE must be activated before simulation can occur. Simulation time step size (SIM STEP SIZE) is an option adjusting the time increment between simulation updates. This alters the speed of the simulation.

During a simulation, it is useful to check for collisions. A primary DEVICE is selected. ADD TO QUEUE puts the devices selected into the Collision Queue of the primary DEVICE. Checks for Collisions and Near Misses are available. The primary DEVICE has the option to stop or not upon detection of a Collision, Near Miss, or both.

As these studies continue, we hope to be able to upgrade to a newer version of IGRIP and continue with the development of the predictive display capability for the teleoperation experiments on the KC-135.

Future plans call for the incorporation of a lightweight video camera placed on or near the arm to provide viewing of the workspace. Graphic overlays, giving supporting information, or command cues may be added. An on-board operator could try using primarily the video view to attempt to alter the robot's task. Simple task altering commands such as "go to rackspace x" would be pre-programmed to allow easy, and fast, task modification. This would also be necessary for remote telepresence experiments in general, as there may be no on-site person to implement or program complex command sequences.

CONCLUSIONS

It has been demonstrated that the KC-135 aircraft can function appropriately as a testbed for robotic tasks in reduced gravity environment, provided that simple and short duration tasks are used. In many cases, larger tasks can be broken into several short tasks, allowing relevant test measurements to be made over a short period of time. For space tasks, the advantage is the ability to view and study manipulator actions in a low gravity environment before going into space. Simple PID controllers designed for ground-based operation do work in reduced gravity, although low cost robots, such as the UMI RTX exhibit excessive jitter and accelerations due to the actuators used in their construction. However, the possibility of using off-the-shelf, or slightly modified laboratory manipulators in the pressurized modules may still provide access to affordable remote experimentation opportunities.

Much can be learned by performing teleoperation demonstrations on the KC-135 aircraft. However, improved sensing is needed for precision training of tasks performed in the reduced gravity of the aircraft. With time, a testbed for remote microgravity simulation could be developed, which combines both the graphic simulation and on-board manipulation. While the UMI RTX is not suitable for precise work, other laboratory robots, which have been designed for precision motion, can be tested.

BIBLIOGRAPHY

1. Toshio Fukuda, "Flexibility Control of Elastic Robotic Arms," *Jour. of Robotic Systems*, 2 (1985) 73-88
2. W.H. Sunada and S. Dubowsky, "On the Dynamic Analysis and Behavior of Industrial Robotic Manipulators with Elastic Members," *Trans. ASME Mech. Transmission Automat. Design* 105, (1983) 42-51
3. E. Heer and A.K. Bejczy, "Control of Robot Manipulators for Handling and Assembly in Space," *and Machine Theory* 18, (1983) 23-35
4. R. A. Clift, "Program Plan for the Astronaut's Apprentice," *Robotics* 1, (1985) 251-264

5. D. R. Criswell, "Robotics and the Space Station," *Robotics* 1 (1985) 205-222
6. M.M. Clarke, "Recent Advances in Teleoperation: Implications for the Space Station," *SPACETECH* 85
7. J. Varsi and D. Herman, "Space Station as a Vital Focus for Advancing the Technologies of Automation and Robotics", *Conference on Artificial Intelligence for Space Applications*, 1986, p 1-6.

Compiling Knowledge-Based Systems from KEE to ADA

Robert E. Filman
Conrad Bock
Roy Feldman

IntelliCorp Inc.
1975 El Camino Real West
Mountain View, California 94040

January 11, 1990

Abstract

The dominant technology for *developing* AI applications is to work in a multi-mechanism, integrated, knowledge-based system (KBS) development environment (e.g., KEETM). Unfortunately, systems developed in such environments are inappropriate for delivering many applications — most importantly, they carry the baggage of the entire Lisp environment and are not written in “conventional” languages. One resolution of this problem would be to “compile” applications from complex environments to conventional languages. Here we report on our first efforts to develop a system for compiling KBS developed in KEE to ADATM. We call this system KATYDID, for KEE/Ada Translation Yields Development Into Delivery. KATYDID includes early prototypes of a run-time KEE core (object-structure) library module for ADA, and translation mechanisms for knowledge structures, rules, and Lisp code to ADA. Using these tools, we have (not quite automatically) compiled part of a simple expert system to run in a purely ADA environment. This experience has given us various insights on ADA as an artificial intelligence programming language, potential solutions of some of the engineering difficulties encountered in early work, and inspiration on future system development.

1 Overview

Our long-term goal is that after a KEE prototype has “settled down,” one pushes a few buttons and the system produces several files of ADA subprograms (and data files for these programs) that encode the application. After transferring this code to the target machine, it is compiled and linked with a library of KEE support subprograms

and with the application or interface in which it is to be embedded. (Thus, in ADA terms, the product of the translation process is an ADA library subprogram that uses an ADA-language, KEE-core library subprogram.) This optimum environment would include facilities for examining and modifying the code produced by the translation, for further development of the application in ADA, and for incorporating these changes in the KEE environment for repeated translation.

Here we report on our first efforts to develop a system for compiling KBS developed in KEE to ADA. We call this system KATYDID, for Kee/Ada Translation Yields Development Into Delivery. KATYDID has components that run in the KEE and REFINETM Lisp-based environments and in arbitrary ADA environments. The components of the KATYDID system are:

- **The KATYDID core.** The core is an ADA-language library which supports the underlying KEE-like object-system base functionality.
- **The knowledge-base dumper.** This system, written in KEE and REFINE, generates structures that can be used by the KATYDID core to recreate a version of an application knowledge base.
- **The rule compiler KRICKIT (Kee/ada Rule Invocation Compiler KIT).** KRICKIT is a REFINE program that takes a set of backward chaining KEE rules and a set of pattern-directed queries on those rules. It compiles them into ADA functions that, when used in conjunction with the KATYDID core, have the same effect as running those queries in KEE.
- **The Lisp-To-ADA translator CIKADA (Commonlisp In Kee to ADA).** CIKADA is a REFINE program that translates the Lisp code of a KEE application to ADA.

As part of our early efforts, we have succeeded in compiling part of a NASA prototype system called CS-1/FIXER [8]. CS-1/FIXER, by reasoning from a structural model, diagnoses and suggests repair actions for a space-based air purification system. Our demonstration system lacks the graphics and display panel mechanisms of the original; we have also hit some severe size limitations of our ADA compiler. However, using our typescript, menu-based interface, we can break the system components and obtain diagnoses of these breakages from their symptoms using the CS-1/FIXER rules. Additionally, we are able to dynamically construct object systems from the keyboard or through ADA programs.

Article overview. This article contains four more sections. In Section 2, we discuss the run-time KATYDID core. In Section 3, we turn to mechanisms for translation from Lisp-based to ADA-based programs, discussing the compilation process itself, compiling Lisp code to ADA, and compiling rules. In Section 4 we discuss our observations on the KBS compilation process and in Section 5, present our plans for extending this work.

2 The KATYDID core

The KEE system is an environment for building knowledge-based systems. KEE is object-centered and integrates several AI problem-solving paradigms, including (1) frames, (2) inheritance, (3) access-oriented programming (demons), (4) object-oriented programming, (5) multiple-worlds, (6) truth maintenance, and (7) production rules, with (8) facilities for querying, altering, and displaying the resulting structures. The first six of these form the KEE core. Unfortunately, space limitations preclude our giving a detailed description of KEE. Additional detail can be found in several of the references [2,5,6].

The KATYDID core is an approximately 10,000 line ADA program that implements the functionality of much of the KEE core. (In certain interesting respects, its functionality exceeds that of KEE.) The gross structure of the KATYDID core is perhaps best illustrated as having three layers: (1) *Structural operations*: The basic types of the KATYDID core and the primitive operations on these types; (2) *Core functionality*: The code that implements the “KEE-like” behavior of the KATYDID core. This includes the functions that store and retrieve slot values, create units and slots, invoke active values, index slots for the rule system, and perform inheritance; and (3) *Core interface*: The presentation of the functionality of the core in the conventional KEE form to applications. That is, the functions of the user’s manual are at this level.

Most of the application code rests on the user level. However, ADA’s lack of dynamic method binding requires that application methods and demons be inserted (by the compiler) under the “core functionality” layer.

Primitive types and data structures. In a strongly-typed language such as ADA, there is benefit to not multiplying types. Towards this end, in KATYDID we implemented a uniform treatment of knowledge bases, units, slots, facets and so forth. Thus, as in Opus [3,4], KATYDID implements a fully recursive unit structure. The data type layer supports primitives for accessing and modifying the fields of an object.

As an (effectively) embedded system, the KATYDID core needs a language for communicating with its callers. It is not enough just to provide objects, since much of the access to KEE objects is symbolic. This leads us to our second datatype, the *symbol*. Like Lisp, KATYDID has the equivalent of oblists and symbol properties.

ADA is strongly typed. If objects are to have values, ADA must know their type. We selected six types as appropriate for slot values: integers, floats, booleans, objects, symbols, and lists. We defined a *box* as a variant record of two fields, one of which indicates the type of the content of the box and the second, the actual data. Lists are *box.lists*; the *car* of such a list is itself a box, the *cdr*, a *box.list*. In essence, this duplicates the “tagged values” that are invisibly supported by Lisp systems. In KATYDID we have to carry the tagging along with each box and explicitly unwrap the box to get to its data.

The datatype layer supports a moderate range of functions for manipulating *box.lists* — functions for placing a value into a box, taking it out, comparing boxes and values for equality, consing values onto *box.lists*, *car*, *cdr*, *rplaca*, and *rplacd*, and a small complement of list manipulation primitives such as membership and set union.

Core functionality. The KATYDID core implements the following important classes of

KEE functionality: object creation; retrieval from symbolic name to data structure; the specification of inheritance links, including the retrieval of the closure of the inheritance link relationships; inheritance, including several predefined inheritance roles; indexing slots of a specified names for the rule system; object properties; and the storing and retrieving of values of slots, complete with coercion between data types and active values. We comment on some of the more interesting aspects of these mechanisms below.

- **Inheritance links and shared structures.** Like KEE, KATYDID supports two varieties of links, superclass/subclass and class/instance. One can make any object be of subclass (subset) or instance (set element) of any other. Such a declaration establishes a *base link* between the two. A link between two objects, parent and child, causes every member sub-object (e.g., member slot) of the parent to exist in the child. If the link is a subclass link, the child has a member sub-object; if the link is an instance link, the child has an own sub-object. If the child already has a sub-object of that name, the two sub-objects are merged. If not, the child shares the sub-object with its parent. We implement this by creating a *derived link* between a non-shared sub-object and its parent sub-object. This allows the normal inheritance process to work between them. The process is repeated recursively down the sub-object hierarchy.

The greatest complexity of the object manipulation comes through the manipulation of objects that share substructures. More precisely, if a parent, G , has a member slot, S , and G 's descendant C does not have any local information about that slot, C shares most of the data structure of S . This saves considerable storage at the expense of dramatically increasing program complexity. When a shared object acquires some local information (such as a value on one of its slots or facets or multiple parents that do not share the same structure) it becomes necessary to "make that object local" to its superior. This may involve search, as the process needs to split the children formerly noted on the parent between the parent and the newly-local child.

- **Inheritance of values.** Slots acquire values both from the explicit, local placement of that value on the slot (*local values*) and because the slot is a descendant of other (member) slots which have values. The slot's visible values are based on combining these through an inheritance role. In KATYDID we implemented several of KEE's inheritance roles: *union*, which does a set union of all parent values with the child's local values; *unique* (or *variable values*), which ignores all parent values; and *override values*, which uses local values, if any, otherwise the values of *some* parent. We extended these roles with a *prototype* role which acquires its parent's values at object creation or linkage but then acts like the role unique.
- **Slot values.** KATYDID includes the full range of functions of storing into and retrieving from the values of slots. The core supports two kinds of active values, AVGETs and AVPUTs, but not the active values associated with attaching and detaching from a slot.

User layer. KATYDID presents in the user layer most of the significant functionality of the KEE core described above. The semantics of KEE often specify that a particular argument can be either a symbol or an object. When such a function has a single such parameter, this leads to the user layer providing overloaded versions of such functions, for symbol, object, and boxed values.

Knowledge-base construction. Rather than develop a symbolic representation of a knowledge base, we chose to have the knowledge-base dumper create ADA procedures that, by calling functions such as `create_unit` and `put_values`, realize the appropriate structures. This is accomplished by constructing Lisp functions to re-create the knowledge base and then running those functions through the Lisp to ADA translator discussed below. This “compiled KB” mechanism has the advantages that (1) knowledge base loading is extremely quick, (2) the code itself can be modified, and (3) it was straightforward to implement. This approach has the disadvantages that (1) the code for creating the KB remains part of the running core image (2) there is no natural mechanism for saving changed knowledge bases, and (3) the size of the files produced can cause problems for some compilers.

Omissions. The KATYDID core, as currently implemented, lacks some of the KEE core functionalities. Most critical of these are (1) the lack of functions for eliminating structure, i.e., deleting objects or removing inheritance links, (2) mechanisms for symbolically loading and saving knowledge bases, (3) valueclass and cardinality checks, (4) obscure inheritance roles, and (5) the ATMS and KEEworlds.

3 System translation

In this section, we (1) overview the nature of the translation process and then discuss how we apply translation to problems of (2) compiling Lisp code and (3) compiling KEE rules.

KATYDID Translation. The purpose of the KATYDID system is to take a KEE application and deliver it in ADA. Almost all KEE applications include some Lisp programs; many of them include KEE rules. The KATYDID approach to rules and programs is to *compile* them into ADA.¹ We call the KATYDID rule compiler KRICKIT (KEEADA Rule Invocation Compiler KIT) and the Lisp-To-ADA translator CIKADA (Commonlisp In KEE to ADA).

Instead of writing our compiler in a procedural language like C or Pascal, we have chosen to use a *transformational implementation* [1]. A program transformation system takes other programs as input and manipulates them by iteratively applying rules. Typically, a rule replaces a program structure in the source language with one in the target language. It is also common that individual rules are *correctness-preserving*: the new program structure computes the same results as the one it replaced.

¹A contrasting approach would be to build, in ADA, both Lisp and rule interpreters. However, since the delivery of KEE applications in ADA is the primary goal of this project, we chose to translate rules and Lisp code directly into ADA. This approach enables us to produce significantly more efficient systems.

Because writing a compiler is a relatively complex task, a number of tools have been built to support compiler development. These tools are often called *compiler-compilers*, even though they usually deal only with part of the problem of compiler construction. In this work, we have used the Refine system [7,9] as our compiler-compiler. Refine is characterized by a grammar facility for expressing language syntax and a tree structure of knowledge objects to represent a program's evolving semantics. In KATYDID we use four different Refine grammars, one for each of ADA, Lisp, the KEE rulesystem, and an intermediate language in KRICKIT. Given a grammar, REFINER performs as both a scanner and a parser generator. Whenever an expression is parsed, it is done with respect to a specific grammar.

Translating Lisp into Ada. The most critical problem in translating Lisp into Ada is that of typing. Lisp is weakly typed—any variable can be dynamically bound to any object. The system tags or remembers the type of all data. Ada is strongly typed and requires types to be specified at compilation. We identified three approaches to the type problem: type inference, overloading, and explicit declaration.

Type inference. Type inference is the inference of the type of Lisp variables from information implicit in the program. In the example below, a variable is set to the result of a function. The return type of the function can be deduced and used as the type of the variable `ch`.

```
(defun test1 (j)
  (let (ch)
    ...
    (setq ch (integer-to-character j))
    ...
  ))

(defun integer-to-character (i)
  (cond
    ((= i 1) 'a')
    ((= i 2) 'b')
    ((= i 3) 'c')
    (t      'd')))
```

The complete transformation cannot be made until the type of `j` is determined and we know whether `test1` is a function or a procedure.

The above example shows two characteristics of type inference. The first is that some inferences are not formally correct. In the example, we are deducing the type of `ch` from the type of the value to which it is set. This could be incorrect, as another assignment statement to the same variable may use a different type. It is often the case that we need to make such *heuristic* decisions about type inference. We can make our inference formal by proving that the variable is not set to another type, or by restricting our use of this deduction to the code that occurs between one setting of the variable and another.

Similarly, we might use heuristic inference to deduce the type of `i` in the function `integer-to-character`. In that case, all comparisons of `i` are to integers, so we can conclude that in normal programming practice `i` will be an integer. However, in Lisp `=` may be redefined by the user, so this deduction cannot be made formal without further analysis.

A second characteristic of type inference is that it can make use of the flow of data through a program. This is shown in the deduction of the return type `integer-to-character`. We can deduce that the result will be in the set `{'a', 'b', 'c', 'd'}`. All these are of type `character`, so we know the result type is `character`. We call this *data flow* type inference. A critical component of data flow inference is identifying the return points of a subprogram and the values returned at those points.

Data flow type inference can be heuristic. For example, if `integer-to-character` lacked its “otherwise” clause (`'d'`), we might still infer that the result type is `character`, even though formally it could include return the `list` or `symbol nil`.

Overloading. The second approach to handling the difference in type strength between Lisp and Ada is to use Ada’s overloading—having multiple definitions of a single subprogram, discriminated on the basis of their argument and result types. This pushes the type discrimination problem off onto the ADA compiler.

There are three main difficulties involved with relying on overloading

1. Many versions of the same function are needed to cover all the possible cases of argument types. With ten arguments of seven types we must write (on the ADA side) 7^{10} ($\approx 2.8 \times 10^8$) versions of the same function. (Such a function was contemplated for the KATYDID core.) Few ADA compilers can handle this.
2. It is possible, with overloading, to create ambiguous programs that fail to compile.
3. Overloading cannot completely replace type inference in Lisp. It cannot, for example, determine the return type of function `test1`.

Thus overloading can only be an element, combined with dataflow analysis, in the resolution of the type inference problem.

Explicit declarations. The third approach to type differences is simply to rely on the user to make Lisp type declarations. Many CommonLisp type declarations can be translated directly into Ada declarations with only a minimal amount of type inference. While *formally* reliable, this approach is both tedious and prone to *human* error.

Rule Compilation. The KATYDID system supports KEE rules with KRICKIT, the backward chaining rule compiler. KRICKIT takes KEE rules and a set of pattern-directed queries as inputs. Its output is a set of semantically equivalent Ada functions. KRICKIT has four components:

- **Rule parameterization.** Rule parameterization is the process of determining the “directionality” or “flow” of the variables in a rule. A given rule can be parameterized in different ways, based on its use in different contexts. Each parameterization results in a separate function.

- **Intermediate language.** The syntax of rules is significantly different than that of a procedural language. KRICKIT translates the rules into an intermediate language, so as to replace the implicit enumeration of rule languages with iterative and enumerative constructs of conventional languages.
- **Ada code generation.** From the intermediate rule language, KRICKIT translates to ADA code, using processes similar to the translation mechanisms described in the last section. A relatively small number of ADA constructs are used in the resulting code.
- **Rule Compiler Runtime.** The only additional facility required in the KATYDID core to support the rule system is an index of those units with slots of interest.

The functionality of the KATYDID rule compiler parallels that of the KEE backward chaining rule compiler—it can compile any predefined query, but is unable to handle dynamically created query forms nor deal with some of the more obscure features of KEE’s rule system (e.g., alternative agenda mechanisms.) In the future we plan to extend KRICKIT to forward and mixed chaining compilers. Such compilers are no more difficult to program than the backward chainer.

4 Observations

In the development of the KATYDID system, we needed to handle several issues in the general transition from a Lisp to an ADA environment. Of primary interest among them are the topics of translating from weak to strong typing, automatic storage management, and indirectly invoked subprograms.

Issues in typing. ADA is a strongly-typed language—the data type of every object must be unambiguous at compile-time. As the experience with Pascal demonstrated, a language that is too rigid about its typing discipline is unworkable. ADA therefore includes three mechanisms for type escape: variant records, overloading, and generics. These features make the language more amenable to symbolic computing, but are inadequate for representing the complexity required by KBS. The problem with the ADA’s notion of type is that it identifies type with implementation. In the real world (which is, after all, what we’re trying to model) the set of “interesting” things for operations is independent their implementation. What programmers need is a facility for grouping things into classes and then checking or restricting operations on those classes, not one that starts with an overly concrete notion of class and makes it pervasive throughout the program.

Automatic storage management. Lisp programs are developed with the luxury of garbage collection. ADA places the responsibility for storage management in the hands of the programmer, who is almost certain to louse it up. We, who are trying to present a library of dynamically accessed objects to a program written with no thought of storage management have an especially hard time. We did nothing particularly clever in the

KATYDID system for storage management. However, we believe that while most applications develop considerable circular object structures, few use circular list structures. This suggests that future versions of KATYDID may be able to manage storage through reference counts.

Indirect program invocation. ADA eschews late binding of program bodies — the compiler always knows what function is called under what conditions. KEE programs, on the other hand, specifically enjoy the ability to change the active value associated with a slot, change the method in a slot, and even more grandiosely, to dynamically redefine new code through method inheritance. While the last is clearly inappropriate in the ADA environment, we have inelegantly simulated the former two by demanding the application (i.e., the application compiler) provide specific functions that map between KATYDID symbols and user code.

5 Future directions

We have developed an initial prototype of a system for translating KEE structures, rules, and programs to ADA and a prototype ADA run-time library for the use of this translated code. Our experience leads us to believe that this translation process should be possible for many applications. While the individual elements of this system have their strengths and weaknesses, considerable effort in translation, library development and the integration of the two is still needed. In Phase II of this effort, we plan to include work on the following topics.

The KATYDID core. The core is incomplete. It lacks not merely the frills like truth maintenance and graphics but important computational components like storage management and the ability to delete structures. Working with the translation mechanism, we need to resolve the appropriate data types and the occasions for their boxing and unboxing. We plan to develop a more hierarchical type structure for KATYDID components. And, of course, this critical code has not been optimized or bulletproofed.

In the long term, the ADA environment offers various opportunities for a more interesting KEE core. For example, ADA includes a multitasking mechanism, suggesting that the next version of the KATYDID core should include locking and transaction mechanisms. It may also be possible for the compiler to optimize accessing functions and data structures when it can prove suitable restrictions on the application program behavior.

The compilation systems. The compilation system can be improved in several dimensions. Within the paradigm of pure translation, we could do a much better job of type inference and type checking and employ a less ad hoc representation of ADA — discovering and representing, for example, the language's semantic constraints. As a compilation system, we also have the opportunity for optimizing the resulting system — performing functions such as data structure optimization, synthesizing variants of core library modules for particular varieties of applications and dead code elimination.

Viewed from a larger perspective, there is considerable opportunity in this work for improving the overall quality of knowledge-based applications. We expect the ultimate

KATYDID system to include utilities for interactive dialog with the developer about the hidden assumptions and behavior of the application, mechanisms to ensure that an application stays within the subset of KEE the core supports, verification and validation mechanisms to critique the structures and programs of the application, and an integrated solution of the problem of system maintenance in a translation environment.

Acknowledgments

We would like to thank the National Aeronautics and Space Administration, Marshall Space Flight Center for their support of this work under NASA Contract NAS8-38036. KEE is a registered trademark of IntelliCorp Inc. ADA is a registered trademark of the U.S. Government, AJPO. Refine is a trademark of Reasoning Systems.

References

- [1] Balzer, R., "Transformational Implementation: An Example," *IEEE Transactions on Software Engineering*, v. 7, no. 1, January 1981, pp. 3-14.
- [2] Fikes, R., and Kehler, T., "The role of frame-based representation in reasoning," *Communications of the ACM*, v. 28, no. 9, September 1985, pp. 904-920.
- [3] Fikes, R., and Nado, R., "Saying more with frames: slots as classes," Working Notes of the 1989 AAAI Spring Symposium Series, Stanford, California, 1989, pp 10-14.
- [4] Fikes, R., Nado, R., Filman, R., McBride, P., Morris, P., Paulson, A., Treitel, R., and Yonke, M., *OPUS: A New Generation Knowledge Engineering Environment*. IntelliCorp, Mountain View, California, 1987.
- [5] Filman, R. E., "Reasoning with worlds and truth maintenance in a knowledge-based programming environment," *Communications of the ACM*, v. 31, no. 4, April 1988, pp. 382-401.
- [6] Filman, R. E., Bock, C., and Feldman, R., "Compiling Knowledge-Based Systems Specified in KEE to ADA," Phase I Report, IntelliCorp, Mountain View, CA, 1989.
- [7] Green, C., and Westfold, S., "Knowledge-Based Programming Self-Applied," *Machine Intelligence 10*, Chichester, Ellis Horwood Ltd. Halsted Press, 1982, pp. 339-359.
- [8] Malin, J. T., and Lance, N., "Processes in Construction of Failure Management Expert Systems from Device Design Information," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-17., No. 6, November 1987, pp. 956-967.
- [9] Smith, D., Kotik, G., and Westfold, S., "Research on Knowledge-Based Software Engineering Systems at Kestrel Institute," *IEEE Transactions on Software Engineering*, v. 11, no. 11, November 1985, pp. 1278-1295.

AUTOMATED UNIT-LEVEL TESTING WITH HEURISTIC RULES

W. Homer Carlisle, Kai-Hsiung Chang,
James H. Cross, and William Keleher
Auburn University

Keith Shackelford
G. C. Marshall Space Flight Center

ABSTRACT

Software testing plays a significant role in the development of complex software systems. Current testing methods generally require significant effort to generate meaningful test cases. The QUEST/Ada* system is a prototype system designed using CLIPS (7) to experiment with expert system based test case generation. The prototype is designed to test for condition coverage, and attempts to generate test cases to cover all feasible branches contained in an Ada program. This paper reports on heuristics used by the system. These heuristics vary according to the amount of knowledge obtained by preprocessing and execution of the boolean conditions in the program.

INTRODUCTION

There are many approaches to software testing, and most require considerable human interaction at a great cost in man hours. The goal of automating this activity is to provide for more cost effective software testing and to avoid human bias or oversight. One class of automated testing tools, the dynamic analysis tools, is characterized by direct execution of the program under test (3). A test data generator is a dynamic analysis tool designed to assist the user in achieving goals such as statement coverage, condition coverage, or path testing. The difficulties of test data generation are due to the computation efforts, sometimes wasted, in computing infeasible paths or solving arbitrary path predicates, especially if a predicate contains non-linear terms or function calls. Consequently AI approaches must be utilized to avoid these problems.

QUEST/Ada* is a prototype system that is designed to experiment with expert system based test case generation. This system seeks to achieve its goals using heuristic rules to choose and generate new test cases. This paper reports on various rule sets designed to achieve condition coverage of Ada programs with increasing amounts of knowledge about the conditions in the Ada program. Knowledge can vary from little information about the input data (requiring random case generation of the appropriate type of input data), to complete symbolic solutions for variables in the conditions under test.

*Research and development of the QUEST/Ada system has been supported by the National Aeronautics and Space Administration (NASA). Ada is a trademark of the United States Government, Ada Joint Program Office.

BACKGROUND

Testing

The reliability of software is critical to space applications. One of the most common ways of ensuring software reliability is through program testing. There are three major categories of software testing: domain testing, functional testing and structural testing.

Domain testing

Programs run on finite state machines over finite input sets. Consequently it is theoretically possible to prove a program correct by testing it over its input domain. However in general these domains are too large for this type of testing to be feasible. It is therefore assumed that programs of arbitrary large storage requirements run on machines of arbitrary large size and precision. Unfortunately this assumption leads to results that demonstrate the impossibility of an algorithm to determine correctness of a program (4).

Functional testing

Functional testing is the process of attempting to find discrepancies between the program's output and its requirements specification (6). In functional testing (1) (4) a program is executed over selected input and the results are compared with expected output. Normally nothing is assumed about the internal structure of the program. Rather, test cases are constructed from knowledge of "what the program is supposed to do", i.e. its "function". This is known as the "black box" approach to testing

Structural testing

Structural or "white box" testing uses the source code control structure of a program to guide the selection of test data (1). One metric for the selection process is coverage, which is concerned with the number of structural units exercised by a test case. Examples of this metric are

Statement Coverage -	execute all statements in the program graph;
Branch Coverage -	encounter all exit branches for each decision node in the program graph;
Path Coverage -	traverse all paths of the graph.

Attempts to develop a practical test generation methodology for branch coverage have suggested approaches ranging from random test generation to full program path predicate solutions. Howden (4) has formalized test generation rules to help programmers test their code. Consequently such rules can be considered "expert knowledge" required for effective and automatic test case generation in an expert system test case generator.

Test case generation

The success of test data generation depends on knowledge of the internal structure of the program. Indeed, in the absence of any such knowledge, the only known testing method is random generation of test data and probabilistic determination of the equivalence of the function under test with desired behavior. On the other hand, if the structure of the program is well understood then by testing, complete validation over a limited domain may be possible. Consider for example a program consisting of a single input variable containing only assignment and increment operations. Such a restriction of a program determines that it can only compute a constant function $f(x) = c$ or a linear function $f(x) = x + c$ for some constant value c . With this knowledge two test cases are consequently

sufficient to identify and validate the program.

Branch coverage is currently regarded as a minimal standard of achievement in structural testing (5). Thus, the goal of an expert system test case generator is to achieve branch coverage by using heuristic rules with execution feedback to generate test cases sufficient to insure that each branch in a program is invoked at least once. Figure 1 gives a system overview of such a test case generation methodology.

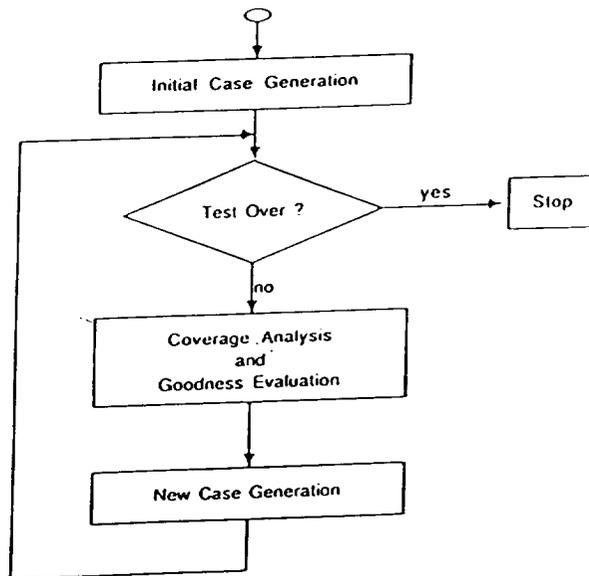


Figure 1

To avoid exponential searches, the analysis may be supported by a search strategy such as that proposed by Prather and Myers (5). This strategy views a software package as a flowgraph with each condition containing a true and false branch. The goal for test cases is to maximize the number of covered branches as recorded in a branch coverage table. The strategy is to select the first condition in a path from the start for which the condition has not yet been tested in both directions, and to generate (if possible) a test case that will drive this condition in the other direction. The idea behind this strategy is that, since some previous test case has reached the condition, it is already "close" to a test value required to drive an alternate branch of the condition.

AN INTELLIGENT TEST DATA GENERATION SYSTEM

QUEST/Ada is a prototype automated software testing tool presently implemented to support expert system based coverage analysis. The framework of QUEST/Ada will however support other rule based testing methods. Figure 2 gives an overview of the relationships among the major components of the system. An instrumented Ada module is supplied as input to a parser scanner that gathers information about the conditions being tested. Using compiled output of the parser/scanner, the test coverage analyzer executes the program for a test case and analyses the result. Based on this analysis, the test data generator uses rules to create new values for variables that are global to or are parameters to the unit under test. These variables are called "input variables".

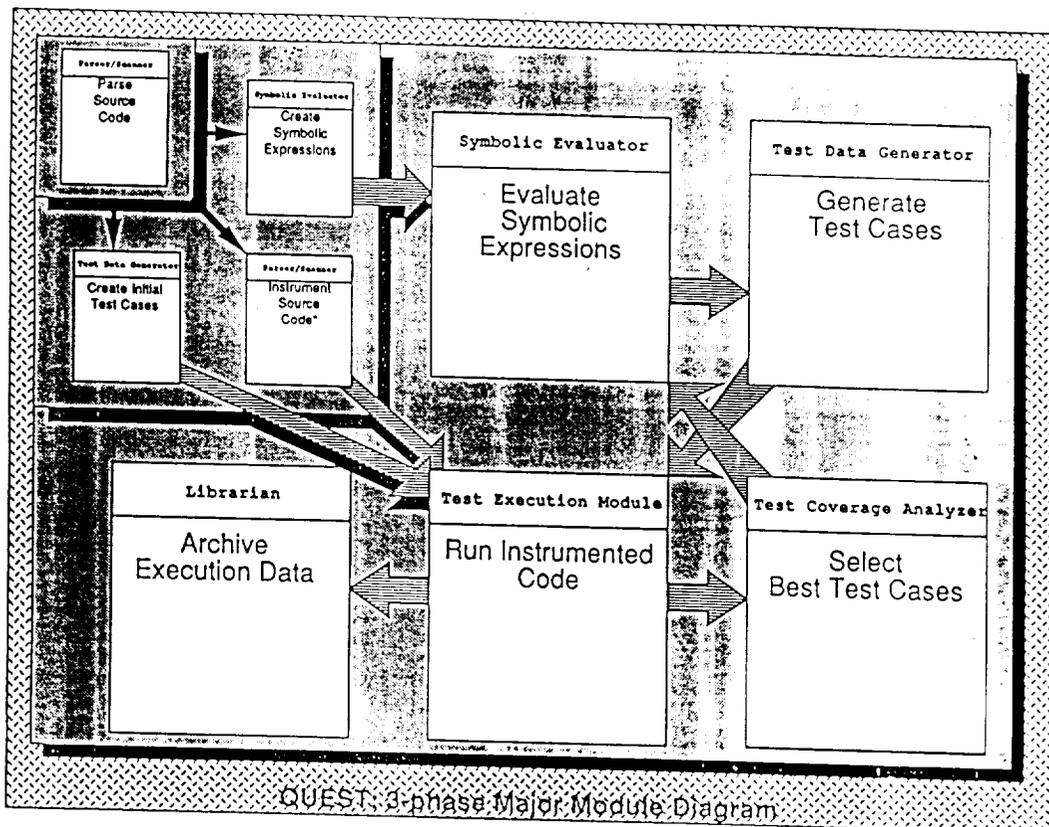


Figure 2

Initial test cases are needed to start the process. These may be provided by the user or generated by the system using an initial test case generation rule. Upon execution of the program on test cases, coverage analysis determines what branches have been covered and which branches need further testing. Coverage analysis is basically a table filling process recording the execution of each condition of the program. The expert system generates new test cases by applying rules based on knowledge about both the conditions not yet fully covered, and previous conditions in the execution path that lead to the condition not fully covered. New test cases are generated, and the testing continues. Execution stops when full coverage is indicated, or when a test case limit is reached. Implementation details of the QUEST/Ada system are described in (2).

Rule Based Test Case Generation

As designed, the QUEST/Ada system's performance is determined by the initial test case, rules chosen to generate new test cases, and the method used to select a best test case when there are several test cases that are known to drive a path to a specific condition.

Initial cases

If the user does not supply an initial test case, then initial test cases are generated by rules that require knowledge of the type and range of the input variables. For these variables test cases are generated to represent their mid-range, i.e. $(\text{upper-limit} - \text{lower-limit})/2$, lower and upper values.

Best test case selection

When there are several test cases that drive a condition in a particular way, a rule is used to select from among these test cases a best test case. Experiments are being conducted with two "best test case" selection rules, with the second rule intended to be more knowledgeable than the first. In the

first rule, the best test case represents a measure of the closeness of the left hand side (LHS) and the right hand side (RHS) of the condition as determined by the formula

$$|LHS - RHS| / (2 * \text{MAX}(|LHS|, |RHS|)).$$

The idea is that test values closer to the boundary of the condition are better. Problems arise in the search algorithm's attempt to cover all branches when a change in values of input variables change an execution path, and execution no longer reaches the condition. In order to decrease the likelihood of such unanticipated branching, a second approach to best test case selection has been designed. This approach utilizes information about the conditions in the execution path leading to the condition under consideration. In this situation, the formula for best test case selection takes into account the closeness of previous conditions. The heuristic idea is that for previous conditions in the execution path, the left hand side and right hand side of these conditions should be further apart. This heuristic assumption is based on the idea that small changes in the values affecting the condition under consideration will have a smaller impact on previous conditions when the left hand side and right hand side are far apart.

As an example, if two conditions c_1, c_2 precede condition c_3 in the execution path, and t_1, t_2, t_3 represent the "closeness" values associated with a test case t , then for weights w_1, w_2, w_3 a value determined by

$$w_3 * t_3 + w_2 * (1/t_2) + w_1 * (1/t_1)$$

represents a better measure of the test case than does the value t_3 . Note that the values of t_1, t_2, t_3 are in $[0, 1]$.

In general, if c_1, c_2, \dots, c_{n-1} represent a path of conditions leading to a condition c_n , and for each $i = 1..n$

$$t_i = |LHS \text{ of } c_i - RHS \text{ of } c_i| / (2 * \text{max}(|LHS \text{ of } c_i|, |RHS \text{ of } c_i|))$$

then for some weights w_1, \dots, w_n , the best test case for condition n is chosen by a minimum value of

$$v = w_n * t_n + w_{n-1} / t_{n-1} + \dots + w_1 / t_1.$$

For testing in QUEST, weights of 1 for w_n and $1/(n-1)$ for $w_1 \dots w_{n-1}$ were chosen.

Test case generation

In order to experiment with the effects of altering the knowledge about the conditions of a program under test, three categories of rules have been selected. The rules are in the syntax of "CLIPS" (7), a forward chaining expert system tool used by the QUEST/Ada prototype. Comments (lines beginning with ;) are intended to explain the action of the rule. The first category of rule reflects only "type" (integer, float, etc.) knowledge about the variables contained in the conditions. These rules generate new test cases by randomly generating values. The following listing provides an example of this type of rule.

Listing 1.

```
(defrule generate_random_test_cases ""
  (types $?type_list)
```

```

;use only type and
  (low_bounds $?low_bounds_list)
;boundary info
  (high_bounds $?high_bounds_list)
;to avoid run error
=>
;set up a loop to generate n test cases for the
;n input variables
  (bind ?outer_pointer 1)
  (while (<= ?outer_pointer (length $?type_list))
;get test case number
  (bind ?test_number (test_number))
  (format test-case-file " %d" ?test_number)
;step thru each variable
  (bind ?inner_pointer 1)
  (while (<= ?inner_pointer (length $?type_list))
;get the type of the variable
    (bind ?type      (nth ?inner_pointer $?type_list))
;assign it a random value
    (bind ?random_value (rand()))
;get range information
    (bind ?low_bound
          (nth ?inner_pointer $?low_bounds_list))
    (bind ?high_bound
          (nth ?inner_pointer $?high_bounds_list))
;be sure random value is within bounds
    (if (> ?random_value ?high_bound) then
      (bind ?test_value
            (* (/ ?high_bound ?random_value) ?high_bound))
    else
      (bind ?test_value ?random_value))
    (if (< ?random_value ?low_bound) then
      (bind ?test_value
            (* (/ ?low_bound ?random_value) ?low_bound))
    else
      (bind ?test_value ?random_value))
;write value for the variable to the test case file
;in appropriate format
  (if (eq ?type int) then
    (format test-case-file " %d" ?test_value))
  (if (eq ?type fixed) then
    (format test-case-file " %f" ?test_value))
  (if (eq ?type float) then
    (format test-case-file " %e" ?test_value))
;next variable in test case

```

```

    (bind ?inner_pointer (+ ?inner_pointer 1)))
  (fprintout test-case-file crlf)
;next test case
  (bind ?outer_pointer (+ ?outer_pointer 1)))
)

```

The second category of rule attempts to incorporate information that is routinely obtained by a parse of the expression that makes up a condition (such as "type" and "range"), information about coverage so far obtained, and best test cases for previous tests. This particular example uses the best test case associated with a condition, and for n input variables, generates n test cases by altering each variable one percent of its range. Listing #2 gives an example of this category of rule.

Listing 2.

```

(defrule generate_increment_by_one_percent_test_cases ""
  (types $?type_list)
  (low_bounds $?low_bounds_list)
  (high_bounds $?high_bounds_list)
;match any condition that is only half covered
  (coverage_table ?decision ?condition truefalse)
;get the best test case for each condition
  (best_test_case ?decision ?condition $?values)
=>
  (bind ?outer_pointer 1)
  (while (<= ?outer_pointer (length $?values))
    (bind ?test_number (test_number))
    (format test-case-file " %d" ?test_number)
    (bind ?inner_pointer 1)
    (while (<= ?inner_pointer (length $?values))
      (bind ?type      (nth ?inner_pointer $?type_list))
      (bind ?high_bound
              (nth ?inner_pointer $?high_bounds_list))
      (bind ?low_bound
              (nth ?inner_pointer $?low_bounds_list))
      ;increment the current variable by one percent of
      ;its range
      (bind ?one_percent (/ (- ?high_bound ?low_bound) 100))
      (bind ?increment
              (+ (nth ?inner_pointer $?values) ?one_percent))
      ;if this is the variable we want to alter
      (if (= ?outer_pointer ?inner_pointer) then
        (if (<= ?increment ?high_bound) then
          (bind ?test_value ?increment)
        else
          (bind ?test_value ?low_bound))
      else
        (bind ?test_value ?low_bound))
    else

```

```

;and the other variables are written as is
  (bind ?test_value (nth ?inner_pointer $?values)))
  (if (eq ?type int) then
    (format test-case-file " %d" ?test_value))
  (if (eq ?type fixed) then
    (format test-case-file " %f" ?test_value))
  (if (eq ?type float) then
    (format test-case-file " %e" ?test_value))
  (bind ?inner_pointer (+ ?inner_pointer 1)))
  (fprintout test-case-file crlf)
  (bind ?outer_pointer (+ ?outer_pointer 1)))
)

```

The final type of rule utilizes information about the condition that can be obtained by symbolic manipulation of the expression. The given rule uses a boundary point for input variables associated with the true and false value of a condition. This value is determined by using symbolic manipulation of the condition under test. Many values can be chosen that cross the boundary of the condition and, as with best test case selection, we seek to choose a value that will not alter the execution path to the condition. In addition to best test case selection we now have additional knowledge to generate new test cases. We use the values of variables at a condition and compare them with values of the variables that reach the condition. This added information is incorporated in the generation of new test cases. To achieve this, the following approach has been taken by the above rule.

Suppose that for an input variable x appearing in a condition under test, the value of x at the condition boundary has been determined to be x_b and the input value that has driven one direction of the condition has been x_i . Although we do not know how x is modified along the path leading to the condition (the value of x_i on input may be expected to differ from the value of x at the condition) we are able to establish that the value of x at the condition is x_c . In this situation we choose as new test cases (provided the values lie in the limits allowed for values of x)

$$x_b * (x_i / x_c) + e$$

where e is 0 or takes on a small positive or negative value. Listing 3 is an example of this heuristic.

Listing 3.

```

(defrule generate_symbolic_approximation_plus_increment_test_cases ""
;type information here
  (types $?type_list)
  (low_bounds $?low_bounds_list)
  (high_bounds $?high_bounds_list)
;knowledge about the condition here
  (coverage_table ?decision ?condition truefalse)
  (best_test_case ?decision ?condition $?values)
  (value_at_cond ?decision ?condition $?vacs)
  (symbolic_boundary ?decision ?condition $?boundaries)

```

```

=>
  (bind ?outer_pointer 1)
  (while (<= ?outer_pointer (length $?values))
    (bind ?test_number (test_number))
    (format test-case-file " %d" ?test_number)
    (bind ?inner_pointer 1)
    (while (<= ?inner_pointer (length $?values))
      (bind ?type (nth ?inner_pointer $?type_list))
;for the variable under consideration
      (if (= ?outer_pointer ?inner_pointer) then
;for its range
        (bind ?high_bound
          (nth ?inner_pointer $?high_bounds_list))
        (bind ?low_bound
          (nth ?inner_pointer $?low_bounds_list))
;get its input value
        (bind ? (nth ?inner_pointer $?values))
;and its value at condition
        (bind ?Xc (nth ?inner_pointer $?vacs))
;and the boundary of the condition
        (bind ?Xb (nth ?inner_pointer $?boundaries))
;generate a guess as to an input value leading to boundary
        (bind ?approximation (* (/ ?Xi ?Xc) Xb))
;generate a small amount to move around boundary
        (if (< (abs ?high_bound) (abs ?low_bound)) then
          (bind ?small_bound ?high_bound)
        else
          (bind ?small_bound ?low_bound))
        (bind ?digit 0)
        (while (!= (trunc ?low_bound) ?low_bound)
          (bind ?digit (+ ?digit 1))
          (bind ?low_bound (* ?low_bound (** 10 ?digit))))
;call it e
        (bind ?e (** 10 (* -1 ?digit)))
        (bind ?incremented_approximation
;increment the approximation by e
          (+ ?approximation ?e))
        (if (<= ?incremented_approximation ?high_bound) then
          (bind ?test_value ?incremented_approximation)
        else
          (bind ?test_value ?high_bound))
        else
          (bind ?test_value (nth ?inner_pointer $?values)))
;write to test case file in appropriate format
        (if (eq ?type int) then

```

```

    (format test-case-file " %d" ?test_value))
  (if (eq ?type fixed) then
    (format test-case-file " %f" ?test_value))
  (if (eq ?type float) then
    (format test-case-file " %e" ?test_value))
  (bind ?inner_pointer (+ ?inner_pointer 1)))
  (fprintout test-case-file crlf)
;next test case
  (bind ?outer_pointer (+ ?outer_pointer 1)))
)

```

CONCLUSION

The objective of the research has been to achieve more effective test data generation by combining software coverage analysis techniques and artificial intelligence knowledge based approaches. The research has concentrated on condition coverage and uses a prototype system built for expert system based coverage analysis. The success of this approach depends on the search algorithm used to achieve coverage and the heuristic rules employed by the search. The effectiveness of rules vary according to the knowledge about the source and the knowledge obtained by previous test cases. The QUEST/Ada prototype provides an extendible framework which supports experimentation with rule based approaches to test data generation. In particular it facilitates the comparison of these rule based approaches to more traditional techniques for ensuring software test adequacy criteria such as branch coverage, and allows for modification and experiments with heuristics to achieve this goal.

REFERENCES

- (1) Beizer, B., *Software System Testing and Quality Assurance*, New York: Van Nostrand Reinhold Co., 1984.
- (2) Brown, David B., *Quest/Ada: Query Utility Environment for Software Testing of Ada*, Phase 1 Report, Contract NASA-NCC8-14, Department of Computer Science and Engineering, Auburn University, Alabama, 1989.
- (3) DeMillo, R.A., McCracken, W.M., Martin, R.J., Passafiume, J.F., *Software Testing and Evaluation*, Benjamin/Cummings Publishing Co., Inc. Menlo Park, California, 1987.
- (4) Howden, W.E., *Functional Program Testing and Analysis*, McGraw-Hill, New York, 1987.
- (5) R.E. and Myers, P., Jr., *The Path Prefix Software Testing Strategy*, IEEE Transactions on Software Engineering, Volume SE-13, Number 7, July 1987, pp. 761-765.
- (6) Meyers, G.J., *The Art of Software Testing*, John Wiley and Sons, New York 1979.
- (7) National Aeronautics and Space Administration, *CLIPS Reference Manual*, Artificial Intelligence Section Johnson Space Center, Version 4.1 September 1987.

SDI SATELLITE AUTONOMY USING AI AND ADA

Harvey E. Fiala

SBI Software Engineering
Rockwell International
Strategic Defense Center
2800 Westminster Blvd.
P. O. Box 3089, MC/EA20
Seal Beach, CA 90740-7644

Telephone: (213) 797-4496

ABSTRACT

This paper describes the use of artificial intelligence (AI) and the programming language Ada to help a satellite recover from selected failures that could lead to mission failure. An unmanned satellite will have a separate AI subsystem running in parallel with the normal satellite subsystems. A satellite monitoring subsystem (SMS), under the control of a blackboard system, will continuously monitor selected satellite subsystems to become alert to any actual or potential problems. In the case of loss of communications with the earth or the home base, the satellite will go into a SURVIVAL mode to reestablish communications with the earth. The use of an AI subsystem in this manner would have avoided the tragic loss of the two recent Soviet probes that were sent to investigate the planet Mars and its moons.

The blackboard system works in conjunction with an SMS and a reconfiguration control subsystem (RCS). It can be shown to be an effective way for one central control subsystem to monitor and coordinate the activities and loads of many interacting subsystems that may or may not contain redundant and/or fault-tolerant elements. The blackboard system will be coded in Ada using tools such as the ABLE development system and the Ada Production system.

INDEX TERMS—Ada, autonomy, blackboard, expert system, frame, global data base, inference engine, knowledge base, and rule base.

INTRODUCTION

Two Soviet probes, Phobos 1 and Phobos 2, were recently sent to investigate the planet Mars and its moons, but were both lost. A ground controller had sent an unverified command that caused loss of communication to the earth. Command verification was not possible because the Soviet's ground control computer, responsible for validating uplink command sequences was down. As a result, no further uplinked ground commands could be received, and the batteries went dead after the spacecraft lost solar panel orientation. This was truly a profound loss to space science. The Spacecraft Autonomy Group criticized the Soviets for allowing unverified uplinks, and for having an on-board computer that so easily accepted such single-transmission command sequences. The United States' Voyager spacecraft will only accept a command sequence that has been

repeated three times. An autonomous system design that included an AI subsystem, as described in this paper, would have enabled the Soviet probes to recover from their tragic circumstances.

Examples of expert systems that provide some degree of autonomy for satellites include (1) the Expert System for Satellite Orbit Control (ESSOC) [Reference 2], which provides autonomous processing for satellite maneuvering operations, (2) the Autonomous Satellite Control [Reference 3], which allows a satellite to operate on its own for up to 30 days; and (3) Spacecraft Control Resolution Expert System (SCARES) [Reference 4], which handles anomalies in a satellite's attitude control system. Other spacecraft with varying degrees of autonomy are reported in [Reference 11].

The GIOTTO spacecraft, which successfully encountered Halley's comet in March 1986, had a number of autonomous facilities on board. These ranged from the simple switching of heaters, to the autonomous reconfiguration of on-board subsystems, extending to the full autonomous recovery of contact with earth [Reference 9].

The Indian Remote Sensing Satellite (IRS), launched in March 1988 into a polar sun-synchronous orbit, achieves a high degree of autonomy by possessing many fault-tolerant features, including automatic reconfiguration logic for the attitude control system [Reference 5].

The Infrared Space Observatory (ISO), planned for launch into a 24-hour orbit in 1993, will require considerable autonomous operation and reconfiguration capability [Reference 12]. Autonomy features will permit recovery of the satellite in good health and enable a quick restart of scientific operations after a period of up to 3 days without earth contact.

Rockwell International has prepared a final report to NASA titled, "Research On Advanced Engineering Software for In-Space Assembly and the Manned Mars Spacecraft" [Reference 1]. This report identifies a strong need for advanced engineering software to support spacecraft autonomy and subsystem health maintenance. It identifies Intelligent Communicating Agents (ICA), which is a form of intelligent distributed software processing, as one example of Advanced Engineering Software directly applicable to future NASA space missions and objectives.

Graceful degradation of overall spacecraft performance takes place as various subsystems fail by using prioritized loading charts contained in the blackboard systems knowledge base. For example, if a failure of a power supply takes place and no spare power supply exists to replace it, then the various loads on any remaining power supplies are either turned off, reduced, or switched to a duty cycle tolerable to the remaining power supply. According to a prioritized table, the least critical functions are either removed or placed on a low duty cycle consistent with minimum mission objectives.

Power supply degradation can occur due to component failures, aging, distance of the solar panels from the sun, half-life of radioisotope power sources, and other potential reasons. Designing independently redundant subsystems could result in an overload of the power system during degraded power system performance. However, if all subsystem reconfiguration is coordinated through a central subsystem, then various subsystem loads can be reconfigured to accommodate a reduced available power, while still meeting overall mission objectives. Under conditions of reduced power, data collection is reduced resulting in less required transmitter power. The result is a graceful degradation of system operation. It is shown here that a blackboard system is a good way to accomplish this reconfiguration management.

AI BLACKBOARD SUBSYSTEM

Figure 1 in the Appendix gives a high-level block diagram for a blackboard system for an autonomous satellite. Blackboard systems provide a mechanism to implement cooperation between a collection of expert systems or knowledge sources. Blackboard systems consist of an explicit global data base (called the blackboard) and knowledge sources that effect and react to changes on the blackboard. Differences among blackboard systems involve mainly control algorithms and mechanisms for determining when knowledge sources should be executed.

The blackboard system works in conjunction with an SMS and an RCS. In the event that a mission-threatening condition is taking place or has occurred, then a corrective action is taken by the RCS. The corrective action is based on various redundant and fault-tolerant features that are integrated into the satellite as part of the overall autonomous design.

The blackboard system is a logical way for one central control subsystem to monitor and coordinate the activities and loads of many interacting subsystems that may or may not contain redundant and/or fault-tolerant elements.

A system or satellite reconfiguration might go as follows: Suddenly the satellite power drops to the 90-percent level. The SMS detects this power drop and sets the relevant flag on the blackboard. The inference engine (scheduler), which monitors the blackboard, is alerted and examines the knowledge sources in the knowledge base and finds a rule that reconfigures the satellite according to priority loading Table 1 in the Appendix. The RCS, alerted from the blackboard, then studies priority Table 1 from the knowledge base and the current system configuration from the blackboard and identifies any required changes. It studies the frame data for the affected subsystems from the blackboard to determine any constraints. In case of a conflict between two units that should not be on together, or between two units that must be on together, an overall priority table in the knowledge base is consulted. The RCS then schedules or reconfigures the affected subsystems according to the new priority table. It then updates the system configuration data on the blackboard.

KNOWLEDGE BASE (LONG-TERM MEMORY)

The knowledge base contains the rule base and the various priority tables.

Examples of two rule templates of the type used in the knowledge base are:

RULE_SPARE_UNIT: If one UNIT fails, and at least one spare UNIT does exist; then switch out the failed UNIT, and switch in the spare UNIT.

RULE_NO_SPARE_UNIT: If one UNIT fails, and a spare UNIT does not exist, and at least one UNIT is still operating, and a prioritized loading table does exist; then reduce the load on the operating UNIT(s) according to the prioritized loading table for the UNIT.

A partial rule base for autonomous SDI satellite subsystem reconfiguration is given in Table 1 in the Appendix.

Load types are classified into types A through D as follows:

Type A: Loads that must be run at 100-percent power and 100-percent duty cycle (for example, the executive or an IMU). An estimated 10 percent of the loads falls into this category.

Type B: Loads that must be run at 100-percent power when they are on, but can be run at a duty cycle of less than 100 percent (for example, if an output is not needed all the time, as in a radio receiver that needs to be powered on only during reception, a transmitter that needs to be powered on only during transmission, an attitude control system that needs to be powered on only during attitude control, and an on-board signal processor that periodically processes data prior to transmission to the earth). An estimated 75 percent of the loads falls into this category.

Type C: Loads that can run at less than full power, but must run continuously (for example, volatile memory, timing sources). An estimated 5 percent of the loads falls into this category.

Type D: Loads that can run at less than full power and at less than 100-percent duty cycle (certain heaters, coolers, etc). An estimated 10 percent of the loads falls into this category.

An example of a simplified prioritized loading table is given in Table 2 in the Appendix.

GLOBAL DATA BASE (THE BLACKBOARD)

The global data base, which is the blackboard, contains the status of all subsystems, frame data on all the subsystems, and the flags for the various subsystems to communicate with each other. Frames are used to collect all required data on each subsystem. In this application, executable procedures are not attached to each slot in the frame, as is the case in the typical blackboard system.

An example of a frame for one of the subsystems is given in Table 3 in the Appendix.

SATELLITE MONITORING SUBSYSTEM (SMS)

The SMS monitors excursions beyond temperature and voltage limits, attitude deadbands, and spin rate limits; monitors selected status flags, time since contact with the satellite's home base, duration of thrust pulses, verification of command sequences; and other mission-critical functions. In the event that a mission-threatening condition is taking place or has occurred, then the SMS sets the appropriate flag on the system blackboard to alert the Inference Engine and the RCS.

RECONFIGURATION CONTROL SUBSYSTEM (RCS)

The RCS monitors the blackboard for changes in the system status. It checks the priority tables in the knowledge base, and it has the ability to reconfigure each of the reconfigurable subsystems.

Reconfiguration of a subsystem can be achieved by turning it off, by turning it off and bypassing it, by changing the duty cycle of the power supplied to it, or by lowering the power supplied to it. After the RCS reconfigures a subsystem, it updates the subsystem's status on the system blackboard.

Table 4 in the Appendix contains a listing of generic satellite subsystems with what is probably their most likely load type.

SURVIVAL MODE

If communication is not reestablished after a certain time, or if the battery charge falls below a prespecified limit, then the SURVIVAL mode is engaged. Power is removed from all possible subsystems and, at a predetermined time, the home base, on or near the earth, will start to transmit a very strong signal, according to a prearranged plan. Or as an alternative, the satellite will go into a star- and sun-tracking mode. At the predetermined time, the satellite will then do an attitude scan with its receiver to lock onto the strong signal from the home base.

The home base will alternate transmission of the strong signal with listening with its receiver until communication with the home base has been reestablished.

IMPLEMENTATION IN ADA

The Ada language has been mandated as the official software language by the DOD, and NASA is now rapidly moving in that same direction for new software programs. There is an advantage to implementing AI programs in Ada from the standpoint of standardization, life cycle maintenance, and customer acceptance.

The blackboard system would be developed first on the Generic Blackboard System (GBB) and then recoded in Ada. The rule base portion of the system would be converted into Ada code using the Ada Production System (APS) [Reference 7]. The APS is a development tool that was developed at Rockwell International and used to develop knowledge-based applications written in Ada. To the author's knowledge, there currently are no commercially available expert system shells that produce executable Ada code. Benchmark testing of Ada code produced by the APS shows that the code executes at approximately 75 percent of the speed of OPS83 code. OPS83 code is based on the C language and is considered to be the fastest production system in use.

Examples of AI systems that have been coded in Ada include the LATEST expert system, the ABLE blackboard system, and the Embedded Rule-Based System (ERS).

LATEST is a very successful rule-based expert system coded in Ada. It gives the reason for a hold or an abort of a Shuttle launch within 3 seconds, where this process normally takes experts several hours of analysis. Knowledge base rules were generated from example sets by a process called rule-induction using the RuleMaster Expert System development tool. It kept the software conventional by avoiding an inference engine. Quoting from Reference 14, "GHC had already made compiling rules for real-time execution possible by developing RadAda to translate RuleMaster interpretive code into Ada source code."

The ABLE system consists of a development system (compiler or constructor) and a library. To create the blackboard system using ABLE, either Erasmus and the ABLE compiler, or the ABLE Constructor would be used. Paraphrasing from Reference 13, "The object of the ABLE compiler is to accept Erasmus source code and produce an equivalent Ada program. The object of the ABLE constructor is to provide the application developer with a graphic interface for defining the structure of a blackboard application. The constructor then performs most of the code generation automatically, producing Ada source code that the developer may fine tune at will."

Quoting from Reference 13, "The ABLE development system may not have been able to fully resolve certain data typing questions, nor may it know details of communicating with other software or devices, so the user may manually extend and/or optimize the code at this point. Code produced will already contain references to all necessary ABLE library units, as well as pertinent sublibrary units, but the user may of course add references when extending the code."

The ERS was successfully recoded in Ada. Quoting from Reference 6, "The project evolved into a major redesign of ERS that exploits Ada's facilities for data abstraction and object-oriented development. The resulting Ada implementation has all of the functionality of earlier versions of ERS (with hooks for many additional features), maintains upward compatibility with existing rule bases, is significantly more efficient than previous versions, and is of higher overall quality by any software engineering standards. Most important, the project demonstrates, convincingly, Ada's suitability and utility for developing knowledge-based systems and embedded AI applications in general."

Ford Lisp-Ada Connection (FLAC) [Reference 8] is a tool designed to support direct entry of knowledge by experts into a Lisp machine environment to help develop expert systems. Paraphrasing from Reference 8, "The knowledge is then downloaded to an inference engine that has been implemented in the Ada programming language. FLAC consists of two subsystems, the Knowledge Editor Graphics System (KEGS) and the Ford Ada Inference Engine (FAIE). The inference engine is written in Ada. It supports both forward and backward chaining modes of inference. FLAC is an application independent system that is generic in the sense that any knowledge that can be represented in rule format can be entered using KEGS, and any Ada program can embed FAIE for expert system capabilities."

That expert system shells, blackboard systems, and other rule-based systems can be successfully coded in Ada can be seen from References 6, 7, 8, 13, and 14, as described above. Rockwell has a very broad set of Ada capabilities [Reference 10], has produced numerous avionics systems coded in Ada, and a blackboard system for an unmanned autonomous satellite is well within its present capabilities.

CONCLUSIONS

This paper has described the architecture for a blackboard system that will coordinate between various satellite subsystems and expert systems to result in the high degree of autonomy required for important defense systems. Sufficient evidence currently exists to demonstrate that AI subsystems can be coded in Ada and embedded in conventional Ada code. In the interests of justifying expenditure of natural resources on important space missions, it should be considered a requirement that each satellite contain an autonomous system that is able to reestablish both vehicle attitude and communications in the event that one or both were lost. Agreeing on certain minimum, internationally developed software standards for satellites and spacecraft can help lead to international cooperation on large-scale missions, such as the Manned Mars Mission.

ACKNOWLEDGEMENTS

The author wishes to express his appreciation to Don DeLaquil for his critical review of this paper and for his helpful suggestions and comments.

AUTHOR BACKGROUND

Harvey Fiala has an MS in electrical engineering from the California Institute of Technology, a Certificate in AI from UCLA, and a BS in electrical engineering from North Dakota State University. He is a Registered Professional Engineer in the State of California and has been a software engineer for 19 years at Rockwell International. He is a member of IEEE, American Association for Artificial Intelligence (AAAI), the Computer Society, the Robotics and Automation Society, and the Planetary Society.

Mr. Fiala is currently responsible for the development of seeker software for the Space-Based Interceptor (SBI) at Rockwell's Strategic Defense Center in California. Prior accomplishments at Rockwell include the development of a two-way Laser Communication System, an Electronic Warfare Jammer, and a multiband pulse sorter. He performed software verification testing on the Shuttle program. He holds patents in the field of laser communications, PRR signal sorters, and DC-to-DC converters.

Prior experience includes project engineer on an infrared seeker-guided missile and experiment engineer on the Surveyor Spacecraft at Hughes Aircraft. He worked on the stable platform for the inertial guidance system for the Polaris missile at General Electric and on the Talos missile guidance system at Bendix Guided Missiles.

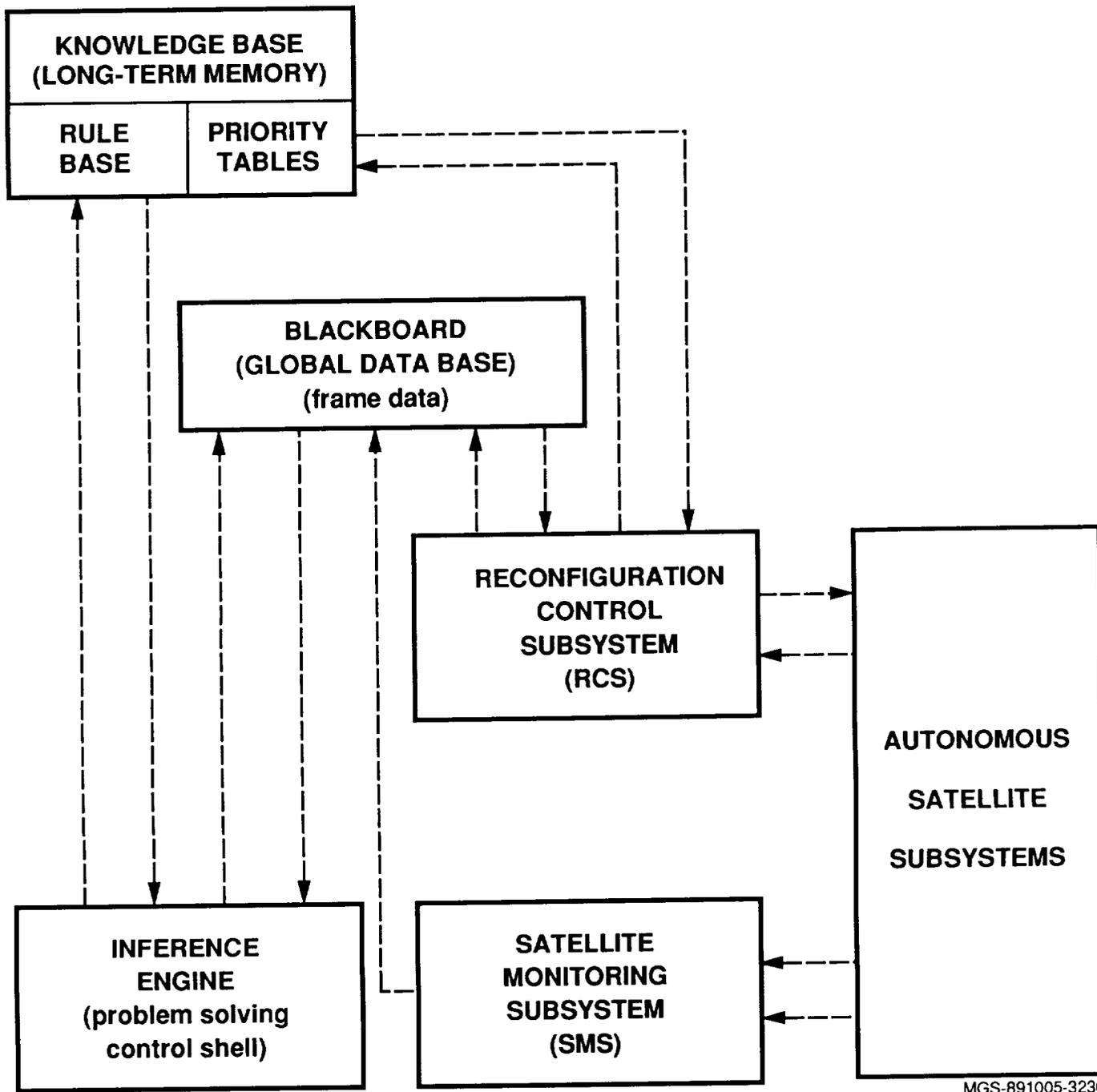
Mr. Fiala's primary interest is in the area of AI. He has coded prototype expert systems in the languages of LISP, Prolog, OPS 83, M1, and Copernicus. He has written and presented, at various national conferences, several papers in the field of AI.

REFERENCES

- [1] DeLaquil, D., T. Reid, J. Hoey, and J. Irwin. Research On Advanced Engineering Software For In-Space Assembly and the Manned Mars Spacecraft. Final Report to the Office of Exploration, NASA, Ames Research Center, Moffett Field, California, the Rockwell International Corporation, Space Transportation Systems Division, Expert Systems Application Group, Downey, California (June 30, 1989).
- [2] Fiala, Harvey E. Aerospace Applications of Artificial Intelligence—A Survey. Aerospace Applications of Artificial Intelligence Conference (AAAIC 88), Dayton, Ohio (Oct. 25-27, 1988).
- [3] Fiala, Harvey E. Artificial Intelligence Applications in Space and SDI—A Survey. The 4th Conference on AI for Space Applications, NASA MSFC, Huntsville, Alabama (Nov. 15-16, 1988).
- [4] Fiala, Harvey E. Artificial Intelligence Applications to Real-World Problems—A Survey. SES III, Rockwell International Third Annual Software Engineering Symposium, Richardson, Texas (Oct. 2-4, 1989).
- [5] Goel, P.S., S. Murugesan, N.K. Malik, and K.S. Chandra. Autonomous Safe Mode Operations of the Attitude Control System of Indian (IRS and INSAT-II) Satellites to Prevent Catastrophe, Present and Future Capabilities. Sponsored by the International Federation of Automatic Control (IFAC), organized by JPL on behalf of the American Automatic Control Council (AACC), JPL, Pasadena, California (Sept. 13-15, 1988).
- [6] Hirshfield, S.H., and T.B. Slack. ERS: An Expert System Shell Designed and Implemented in Ada. AIDA-88, Fourth Annual Conference on Artificial Intelligence and Ada, Herndon, Virginia (Oct. 14-15, 1987).

- [7] Irwin, J.A. Symbolic Processing in Ada. SES III, Rockwell International Third Annual Software Engineering Symposium, Richardson, Texas (Oct. 2-4, 1989).
- [8] Jaworski, A., PhD., D. LaVallee, and D. Zock. A LISP-Ada Connection for Expert System Development. AIDA-87, Third Annual Conference on Artificial Intelligence and Ada, Herndon, Virginia (Oct. 14-15, 1987).
- [9] Nye, H.R. Onboard Autonomy Considerations for the GIOTTO Mission to Halley's Comet, Spacecraft Autonomy: Present and Future Capabilities. Sponsored by the IFAC, organized by JPL on behalf of the AACC, JPL, Pasadena, California (Sept. 13-15, 1988).
- [10] Post, J.V. Ada Capabilities. Rockwell International, Space Transportation System Division, distributed at the Seventh National Conference on Ada Technology, Atlantic City, New Jersey (Mar. 13-16, 1989).
- [11] Post, J.V. International Federation of Automatic Control Workshop in Spacecraft Autonomy, Present and Future Capabilities. Trip Report, Jet Propulsion Laboratory (Sept. 13-15, 1988).
- [12] Robson, A. and W. Van Leeuwen. Autonomous Operation and Reconfiguration Features of ISO and Their Impact on the Ground Segment. Sponsored by the IFAC, organized by JPL on behalf of the AACC, JPL, Pasadena, California (Sept. 13-15, 1988).
- [13] Stockman, S.P. ABLE: An Ada-Based Blackboard System, AID3-88. Fourth Annual Conference on Artificial Intelligence and Ada, Herndon, Virginia (Oct. 14-15, 1987).
- [14] Wood, K., L.G. Hanely, B. Lawson, C. Randall, R. Turner, and Wang Chi-Wei. Shuttle Failure Detection, (using LATEST, a Rule-Based Expert System in Ada). Aerospace American (July 1989).

APPENDIX



MGS-891005-3230

Figure 1. Blackboard System for Autonomous SDI Satellite

Table 1. Partial Rule Base for Autonomous SDI Satellite Subsystem Reconfiguration

RULE_SPARE_HEATER: If one HEATER fails,
and at least one spare HEATER does exist;
then switch out the failed HEATER,
and switch in the spare HEATER.

RULE_NO_SPARE_HEATER: If one HEATER fails,
and a spare HEATER does not exist,
and at least one HEATER is still operating,
and a prioritized loading table does exist;
then reduce the load on the operating HEATER(s)
according to the prioritized loading
table for the HEATER.

RULE_SOLAR_PANEL: If one SOLAR_PANEL fails,
and at least one spare SOLAR_PANEL does exist;
then switch out the failed SOLAR_PANEL,
and switch in the spare SOLAR_PANEL.

RULE_NO_SPARE_SOLAR_PANEL: If one SOLAR_PANEL fails,
and a spare SOLAR_PANEL does not exist,
and at least one SOLAR_PANEL is still operating,
and a prioritized loading table does exist;
then reduce the load on the operating SOLAR_PANEL(s)
according to the prioritized loading table for
the SOLAR_PANEL.

RULE_SPARE_BATTERY: If one BATTERY fails,
and at least one spare BATTERY does exist;
then switch out the failed BATTERY,
and switch in the spare BATTERY.

RULE_NO_SPARE_BATTERY: If one BATTERY fails,
and a spare BATTERY does not exist,
and at least one BATTERY is still operating,
and a prioritized loading table does exist;
then reduce the load on the operating BATTERY(s)
according to the prioritized loading
table for the BATTERY.

RULE_LOST_CONTACT: If LOST_CONTACT_FLAG is set
then initiate the SURVIVAL mode.

RULE_CONTACT_REESTABLISHED: If
CONTACT_REESTABLISHED_FLAG is set
then resume the NORMAL mode.

Table 2. Electrical Power Loading Reconfiguration Priority Table. When the Available Power Drops by 10 Percent, the Loads Are Reconfigured According to the Table

Load Type and No.	Power Supply Output Level (%)									
	100	90	80	70	60	50	40	30	20	10
A1	10	10	10	10	10	10	10	10	10	10
B1	20	15a*	10a	10a	10a	10a	5a	5a	5a	0
B2	20	15a	10a	10a	10a	10a	5a	5a	5a	0
B3	10	10	10	5a	5a	5a	0	0	0	0
B4	10	10	10	5a	5a	5a	0	0	0	0
D1	30	30	30	30	20	10	10	0	0	0
Total	100	90a	80a	70a	60a	50a	40a	30a	20a	10

Note*: a = average power achieved by lowering the duty cycle. For example, 15a means 15 W average. The duty cycle is lowered.

Table 3. An Example of a Subsystem Frame

Subsystem name:	Laser communication receiver
Constraints:	Must not be on simultaneously with laser transmitter
Power required:	5 W
Load type:	B
Number of spares:	One
Status:	Operational
Previously bypassed:	No

Table 4. Generic Satellite Subsystems and Load Types

Satellite Subsystem	Typical Load Type
Autonomy subsystem	A
Electrical power system	A
Executive	A
IMU	A
ACS	B
Axial thrusters	B
Data processor	B
Divert thrusters	B
GN&C	B
High-power experiments	B
Low-power experiments	B
Receiver	B
Self-test	B
Solar collector	B
Star tracker	B
Tape recorder	B
Clock (timing reference)	C
Volatile memory	C
Heater	D
Transmitter	D

ADA AS AN IMPLEMENTATION LANGUAGE
FOR KNOWLEDGE BASED SYSTEMSDaniel Rochowiak
Research ScientistJohnson Research Center
University of Alabama in Huntsville
Huntsville, AL 35899

ABSTRACT

Debates about the selection of programming languages often produce cultural collisions that are not easily resolved. This is especially true in the case of Ada and knowledge based programming. The construction of programming tools provides a desirable alternative for resolving the conflict.

INTRODUCTION

If one wants to generate a debate at a party for persons connected with computer programming, just ask "What is the best programming language?" The result is often an outpouring of praise, curses, hyperbole, and technical detail that will either quicken the pulse or induce tranquil repose. Programming languages are at times treated as matters of religious fervor, and at other times treated as mere notational convention. All of this would be fine were it not for the demands for "good" software and the increasing size, complexity and seriousness of software programming projects. To be sure software is more than the code for a program. Software, in the sense includes all of the information that is: (1) structured with logical and functional properties, (2) created and maintained in various representations during its life-cycle, and (3) tailored for machine processing. This information in large projects is often used, developed, and maintained by many different persons who may not overlap in their roles as users, developers, and maintainers. In order to develop good software, one must explicitly determine user needs and constraints, design the software in light of these and in light of the needs and constraints of the implementers and maintainers, implement and test the source code, and provide supporting documentation. These dimensions and constraints on producing software can be looked at as aspects of different moments in the software production process.

The programming languages LISP and Ada can each legitimately claim a special competence. In the case of LISP, it is symbolic processing, and in Ada, uniformity and maintainability. In making a decision about a programming language, the programming language and its environment cannot be meaningfully separated. Whether one examines LISP or Ada, it is clear that the advocates of these languages are not considering the languages in isolation. The combination of programming environment and programming language is intimately connected with the programming paradigm that can be used in the construction of the program. A programming paradigm may be thought of as the style or manner in which a program is created. Within one paradigm there may be many particular templates, but there is a sense in which each of these reduces back to some primitive template. Alternatively, one may view the paradigm as a primitive object from which the specific template inherits structures and properties. Under either sort of interpretation, it should be clear that a programming paradigm acts as a vehicle through which a programmer designs and builds specific programs.

Certainly another way to generate debate is to ask, "What is the best representation of knowledge?" or "What is the best way to manipulate knowledge?" The list of answers will grow rapidly: logic, rules, frames, scripts, objects, trees, nets, inferences, associations, statistical inferencing, case based reasoning, analogy, and so on. All of these styles and techniques have valued uses. All have their strengths and weaknesses. Unless a person was very lucky, no consensus would be achieved at the party.

Behind both the questions about programming languages and the questions about knowledge is a common social structure. Programming and the construction of knowledge based systems occur in cultures. These cultures are the repository for tradition, tacit rules of procedure, and tacit rules of appraisal. A person's training is the way in which they are enculturated. Someone who is trained on a certain hardware, in a certain language, and in a certain style will carry the culture generated through that training onto his or her new works. As persons with their cultures collide differences of opinion, and difficulties in adjusting to the demands of another are sure to be produced. This collision of cultures is a central element of the issues surrounding the debates about knowledge based programming and Ada.

PARADIGMS AND CULTURES

Typically a culture has a core paradigm or set of paradigms that capture the core of the culture. The paradigms act as cognitive templates that are filled in when either trying to solve a problem or develop an object.

In programming there is an interaction between what may be considered a programming paradigm and a programming language. Stroustrup (8) sets out the relation between a programming paradigm and a programming language rather neatly.

A language is said to support a style of programming if it provides facilities that make it convenient (reasonably easy, safe, and efficient) to use that style.... Support for a paradigm comes not only in the obvious form of language facilities that allow direct use of the paradigm, but also in subtle forms of compile-time and/or run-time checks against unintended deviations from the paradigm... Extra-linguistic facilities such as standard libraries and programming environments can also provide significant support for a paradigm.

The problem of selecting a paradigm is both art and science. It is art insofar as it requires a subtle understanding of the programming craft, and is science insofar as a set of decision rules can be established for the paradigm. The four typical paradigms are: procedural, data hiding, abstract data type, and the object-oriented Paradigm.

In examining Ada and LISP the idea of the programming paradigm can be usefully extended to the paradigmatic way in which programming languages and their environments are used. The question is whether one language offers tools that make it best suited to a particular task. Although there is a formal sense in which all sufficiently rich languages are equivalent, this equivalence is only logical or formal. Although any of the paradigms can be accomplished in either LISP or Ada, this does not mean that it is either easy or reasonable to use any of the mixtures of paradigm and language that are possible. Since LISP has been the chief language of artificial intelligence research, it is reasonable to investigate whether Ada can support the constructs of LISP. In this way the issue concerns whether Ada can implement the LISP paradigm.

Schwartz and Melliar-Smith (7) analyzed the Ada specification to determine its potential as an AI research language. Their conclusion is that Ada, as defined in the Preliminary Standard,

would not be suitable as a “mainstream research language.” They proposed, however, that with some extensions it is plausible that a substantial portion of AI “algorithms” could be translated into Ada. This translation would not be easy, since it would be more of a “reimplementation” of the program, but the “complex heuristic algorithms that provide the artificial intelligence” could be retained.

Schwartz and Melliar-Smith’s claim of Ada’s unsuitability is fundamentally based on the determination to enforce a particular programming paradigm. One goal that was set forth in both the Ironman and Steelman Requirements, is to create “an environment encouraging good programming practices.” Ada imposes a style of programming that is the result of many years of research on programming methodology. Ada is intended to impose a very disciplined style of programming that assists those who are developing large, complex projects that require teams of programmers. Furthermore, Ada is said to be ‘readable and understandable rather than writeable’ so as to minimize the cost of program maintenance. Thus, Ada’s mandated programming style is beneficial for the targeted Ada community - a production community, especially a community that produces real-time embedded systems. In general, AI work does not occur in a production community, but a research and development community. This difference in orientation is a factor in making Ada unsuitable as a general AI research language. The constraints of production prevent the AI programmer from using the most natural method of expression for whatever system is being developed. The LISP programmer places greater value on code that is more easily writeable than readable. However, two things should be remembered. First, the readability of any code is a function of the enculturation of the reader. Second, the readability of the code is a function of the tools available with which to read it. This latter point is important when one considers LISP on a LISP machine. Within that environment the code may become very readable through the tools that are available for reading it.

Schwartz and Melliar-Smith contend that the utility of Ada for AI programs is confined to the reimplementation. This operation would be carried out by software teams by following the algorithms of an original program, but not necessarily its detailed code. Extensions to Ada are needed, however, if such reimplementation is to be carried out while preserving Ada’s structure and modularity.

A typical AI task for a knowledge based system in LISP is to generate solutions to problems that have a very large number of alternatives. To attempt to solve such a problem by exhaustive search or “best fit” is not feasible even with a supercomputer. A heuristic based guess is used to prune branches from the decision tree so that the problem becomes tractable. In some “classic” systems, a breadth-first or depth-first search is used to consider candidate solutions. When it becomes apparent that an incorrect decision has been made, then the search resumes at the junction where that decision was made. Use of heuristics allows for systems to “learn” from their mistakes and refine their search techniques as more is “learned” about the problem domain. Several features of AI programs stand out. First, extensive use is made of the list structure and the processing of lists. Second, procedures are often used as values that can be stored in a data structure. This allows for the construction of a generic framework for the parameterized transformation of a given type of structure. An example of this would be the construction of a system to perform an arbitrary function on a tree or graph structure. If the procedures are values of a procedural data type, then the procedures could be passed as parameters to perform the desired manipulation of data. Third, LISP provides for a similar representation of both data and programs that allows for the creation of functional abstractions “on the fly.” These abstractions, expressed by Lambda calculus list expressions, can be passed as parameters to other abstractions. Fourth, as each function or expression is defined, it becomes part of the system. Thus, the application program can examine its run-time environment, a fact which makes the program inseparable from its environment. Finally, the ability to use procedures as storable objects is essential to many AI programs. One use for this ability is as a method to express knowledge about a particular domain. Frequently, several different knowledge representations will be used

in one system. The particular representation used would depend on the availability of information.

PACKAGES FOR ADA

Much of the success of LISP as an AI language can be attributed to fact that it is extensible. It is possible, for instance, to construct rather easily interpreters for other high-level languages using LISP. This ability is facilitated by the manner in which LISP programs are represented: as lists. Ada, too is extensible. (2) However, Ada is more limited in its extension capabilities, with packages, generic procedures and tasks being all of the extension methods. Whether or not this extensibility is to limited for the needs of reimplementing AI programs remains to be seen. Ada provides data abstraction facilities that allow one to create extensions to the language by the defining of new data types and the operators that can be used to manipulate them. Through the use of a package containing a data abstraction, a programmer can write code as if the facilities provided by the package were provided by Ada. Thus the addition of packages may provide a way in which the typical features of an AI program written in LISP can be reimplemented in Ada. Such a package may, for example, supply the tools needed to handle lists, procedures, and garbage collection.

List processing is an important feature of AI research languages. Whereas Ada does provide the features needed to implement list processing, its garbage collection facilities leave much to be desired. No special considerations have been made for list processing, and consequently, the efficiency of such will likely be minimal. To implement lists in Ada, one could create a data structure as follows. Each list cell would be a record that has two list pointers: CAR and CDR. A list pointer would then be a record that has only a variant part. The discriminant of this variant part would have two possible values: ATOM and LIST. This would indicate whether the list pointer component is a list reference or an atom reference. There would need to be a LIST_REFERENCE and ATOM_REFERENCE access types for the dynamically allocated list cells and atom cells.

Although procedural variables cannot be readily added to Ada, it is conceivable that the ability to pass procedures as parameters could be added. The effect of the instruction part of a procedural value can be simulated through the use of a generic procedure. This method would avoid using a CASE statement as would be necessary if the indexing scheme were used. Generic procedures used in this fashion would carry the name of the "passed" procedure but would not have the closure or environment.

As most AI programs "run," they pursue a number of possible alternative paths of action. This attempt to find the best possible path usually succeeds in allocating a great deal of memory. Since the memory objects have a lifetime that is dependent on the duration of the utility of the data, and not the flow of control of the program, these objects must be allocated in a global heap. By using a heap, storage can remain at least as long as it is referenced anywhere else in the system. Consider a typical embedded system application. Here, the data that must have space in the heap is minimal. Thus, reclamation of heap space is not important, and in some cases, heap space is not reclaimed at all. This is yet another design philosophy contradiction, between Ada and AI languages. AI languages are designed with the philosophy that "no amount of initial heap allocation will be sufficient for the continued operation of many AI programs." It is not a question of if all of the heap space will become allocated, rather is is a question of when it will happen. Obviously, some strategy must be used to reclaim this storage space. The language specification for Ada does not preclude garbage collection capabilities, nor does it indicate these will be included. There is a mechanism, FOR-USE, which indicates the maximum number of objects of an access type that may be generated. Since the compiler knows the maximum size in advance, the necessary space can be allocated. This provides a sort of heap-type allocation with

automatic reclamation for objects that have a limited scope of use. Unfortunately, this method causes allocation/deallocation to be dependent on control flow or block entry and exit.

PATTERNS

Obviously, not everyone agrees with Schwartz and Melliar-Smith on Ada's place in AI. Larry Reeker, John Kreuter, and Kenneth Wauchope of Tulane University have done much work on pattern matching in Ada. In answer to the question of "will Artificial Intelligence be done in Ada?" they answer that "anything can be done in Ada," and attempt to show how Ada, when appropriately used, can facilitate the programming of Artificial Intelligence applications. (6)

Reeker has chosen to focus on a pattern-directed because "pattern-directed facilities provide the most effective means for creating complex programs for non-numerical applications." Further support for pattern matching can be found in the work of Warren, Pereira and Pereira. (9) They contend that pattern matching "is the preferable method for specifying operations on structured data, both from the user's and the implementer's point of view."

Reeker envisions the addition of AI oriented features to Ada through the use of packages. The list of features that are candidates for incorporation into Ada include:

- String definition and manipulation facilities more flexible than those built into Ada.
- List processing functions
- Pattern definition and matching functions for strings and lists
- A means of manipulating lists returned by the pattern matching functions

Ada's concurrency paradigms lead to a number of possible methods for pattern matching. One such method would be to use tasks as "coroutines" to match patterns. There are areas in AI which have made use of "quasi-parallel" processes previously. True parallel tasks executing on a true multiprocessor system would surely improve on those systems.

In his section of their paper, Kenneth Wauchope presents an Ada language implementation of a pattern-directed list processing facility. A set of SNOBOL-4 like primitives are used to construct lists that are equivalent to arbitrarily complex LISP-like data structures. Wauchope advocates the addition of packages to make AI feasible in Ada. In this paper he describes the operation of a package which provides basic list creation and manipulation functions similar to those in LISP. Wauchope then presents several applications of these new features, including: parsing a context free grammar and symbolic differentiation.

Kreuter presents several algorithms for pattern matching in Ada. The first of these is the recursive descent parsing method which is a common way to implement the backtracking strategy. Backtracking is based on the intuitive approach of trying every possibility for each pattern element. This generates every possible parse of the string but is rather costly in terms of time.

One particularly interesting possibility arises with the use of Ada for coding such algorithms. Since Ada allows concurrent tasks, the backtracking aspects of the algorithm could be achieved through the use of tasks that behave as coroutines. A task would start by examining each bead in the first set of alternatives. A new task is forked for each successful match. This new task will then examine the remainder of the string and the remaining sets of alternatives. After all alternatives have been examined, the task will pass back the matching substrings, or null in the case of no match, and terminate. Each successive parent task will then add its substring to the beginning of each tree on the list which has been passed to it. Then, this list is passed back, and so forth, until the master task is reached.

Combinatorially implosive algorithms (CIA's) are a class of parallel algorithms that employ two or more algorithms running concurrently such that they will solve a problem more quickly than one would by itself. Brintsenhoff, Christensen, Mangan, and Greco demonstrate a CIA coded in Ada in their paper, "The Use of Ada Concurrent Processing Features in an Implementation of Parallel Tree Searching Algorithms." (3) This study is interesting because the authors had access to a multiprocessor with run-time support for concurrent tasking. Their findings show the speed advantages of parallel algorithms written in Ada. Although the results were highly data dependent, the running of two algorithms concurrently proved to be more efficient than just one and thus proved the utility of CIA's in Ada. If such CIA's could be developed for pattern matching, it is reasonable to expect that pattern driven AI applications would prove to be very efficient in Ada.

OPTIONS

The two previous sections have indicated two ways in which the confrontation of Ada and AI might proceed. In the first way the differences of the two cultures are acknowledged and an effort is made through the addition of appropriate packages to provide the tools for a reimplementing of an AI program. The second option acknowledges the fact that in a sufficiently complete language it is possible to implement the idea of a program directly. Each approach has certain advantages and disadvantages. In the first approach the program does not have to be completely rethought and redesigned. This is a disadvantage of the second option since the ideas for the program have to be implemented from scratch. In the second approach there are potential advantages to be gained by using the strengths of Ada. This is the disadvantage of the first option. The addition of the packages may in effect provide for a LISP interpreter that circumvents the natural strengths of Ada.

One way in which a decision between these options might be facilitated is by using the resources of software engineering. Fairley (4), for example, defines software engineering as the "technological discipline concerned with the systematic production and maintenance of software products that are developed and modified on time and within cost estimates," and claims that software engineering is a "new technological discipline distinct from, but based on the foundations of, computer science, management science, economics, communication skills, and the engineering approach to problem solving." Boehm (1) identifies seven basic principles in software engineering. These are:

1. Manage using a phased life-cycle plan,
2. Perform continuous validation,
3. Maintain disciplined product control,
4. Use modern programming practices,
5. Maintain clear accountability for results,
6. Use better and fewer people,
7. Maintain a commitment to improve the process.

Of these principles one requires special attention in this context, injunction to use modern programming practices.

Programming paradigms are at the root of modern programming practices. As Boehm (1) notes, "The use of modern programming practices (MPP), including top-down structured programming (TDSP) and other practices such as information hiding, helps to get a good deal more visibility into the software development process, contributes greatly to getting errors out early, produces understandable and maintainable code, and makes many other software jobs easier, like integration and testing." At issue, of course, is what counts as a modern programming practice. Interpreting this principle is complicated by the facts that modern programming practices are not fixed, that such practices are the outgrowths of programming paradigms, and

that the paradigms are responses to the practical needs of computer software developers and the intellectual demands of computer scientists.

Thus, Boehm's principle that modern programming practices ought to be used is a bit odd. What it might really mean, however, is not that any modern programming practices should be used, but that the modern programming practices for imperative, conventional languages that are used for large software projects and can be handled within the current discipline of software engineering should be used. In this sense the modern programming practices are those geared to the community and culture of production. Neither LISP nor object-oriented programming can satisfy those demands. However, Ada comes near to being the ideal language from the point of view of software engineering with conventional languages. This points out the difficulty in generating a set of principles to guide software engineering. The analogy of software engineering to the rest of the engineering field (10) begins to break as one attends to the nonphysical character of software. For example, when building a bridge or a pipeline, the standard elements of the construction remain static. Bridges will have beams and pipelines will have pipes. The materials and techniques may change, but the basic elements remain. Unconfined by such physical characteristics, the elements of software construction can change. Subroutines, subprograms, libraries, modules, packages, units, functions, objects and many other elements are available to the software programmer, and new as yet unthought of constructs might be added. All of this adds to the complexity of choosing and using modern programming practices, and points to the important role of the software manager even within the software engineering discipline.

The decision as to which of the two options should be pursued any not therefore be decidable on the grounds of software engineering alone. If the other principles that Boehm isolates are essentially management principles then it is fair to assume that they can be satisfied with any language and any programming paradigm. In this sense they are transcultural. However, the injunction to use modern programming practices is what the collision of cultures is about. What is a modern programming practice? Each culture will defend itself as being the exemplar of modern programming practice. Given the existence of the colliding cultures, it does not appear that the principles of software engineering will be able to generate a clear choice.

Another way in which the choice might be made is to focus on a technological solution. In particular the development of software tools that allow for program development in a neutral environment, but can generate code in a target language. The use of automated tools to manage the software coding process, including the generation of source code in a target language, raises another interesting issue, however. If the tools are good tools and if the code they generate is good code, then what is the programmer doing? In a primary sense he or she is running the tool; in a secondary sense he or she is programming in some language. There is a sense in which if the tools are very well done, the "programmer" need not even know the language in which he or she is programming, and, indeed, need not even know in what language the code is being generated. As Howden (5) has noted:

The manufacture of software is perhaps one of the most logically complicated tasks. The intellectual depth of software systems development coupled with the lack of physical restrictions imposed by properties of the product make software manufacturing intriguing in its possibilities for highly automated, sophisticated manufacturing environments. Research has begun, on environments containing their own concept models of general programming knowledge... It has been speculated that in the future software engineers will be able to describe application programs to a system capable of automatically generating specifications and code.

An intriguing possibility! The programmer is no longer a crafter of code, but an expert user of a tool. The connection between the programming language and the programmer is, in a sense, severed. The programmer with such tools may, therefore, function at a higher, more natural level of abstraction without needing to attend to the syntactic complexities of the language in which the application is finally coded. This is not, however, surprising. It represents simply another moment in the evolution toward higher level languages. Rather than a traditional higher level language being used with a compiler to generate the low level instructions to the processor, a new generation of tools may operate at an even higher level and a translator may then convert the tool's specifications into a higher level language which in turn may be compiled.

If this technological path is found desirable, then it suggests that the first option, the addition of packages and reimplementations of code, is on the right track. Further it suggests that the second options benefits might be incorporated into the tool. If for example the target hardware is a multiprocessor system, then a tool should be able to guide the tool user in creating code appropriate to the hardware. The creation of such tools is not, however, thought of as revolutionary. Rather the emergence of such is another evolutionary step in generating higher level software facilities for programming more complicated hardware.

SPECULATION

It is clear, for example, that processors have improved greatly over the past two decades. Increased speed, increased word size, augmented capabilities, decreased power consumption, and decreased cost are readily apparent. All of these factors combine to allow those who design and build languages and environments to implement more easily and effectively ideas and constructs which with less capable processors would remain dream and desire. One need only to recall what it was like to run LISP on a PDP-11 under RSTS and look at a Symbolics or Texas Instruments LISP machine to recognize the difference. Similarly, C in its own UNIX environment has come to be recognized as a powerful system, and has led to the evolution and development of the computer workstation. The future holds even more promise. Even as physical limitations begin to affect the development of better processors, new architectures begin to evolve. Multiprocessor machines, parallel processor machines, and other objects of wonder and splendor open new vistas to the language crafter. Although languages like LISP and C will probably move into these new environments, their form and function will probably be much different. Equally probable is that new languages will emerge. In any case, one point is clear. Languages are not static. Language development responds to the state of the processor art. As long as processor development continues, it is reasonable to expect programming languages to develop.

It should also be clear that programming paradigms change over time. The changes of paradigm reflect both the intellectual development of computer programming and the ability of the language crafters to build support for a paradigm into a language. BASIC was a wonderful language. It was criticized for not supporting a structured programming paradigm. New BASIC arose in response to that criticism. Classic LISP did not support object-oriented programming. LISP with FLAVORS is a virtually seamless environment in which such programming is supported and encouraged. C did not support the object-oriented paradigm; C++ is a response as is Objective C. If it were not for the government's involvement with Ada, one might well think that OO-Ada (Object-Oriented Ada) might soon appear. There is, of course, no reason to think that the story ends with the object-oriented paradigm. New paradigms, perhaps tailored to particular classes of problems, may well arise. As they do, old languages may evolve to support them, and new languages may arise to enforce them in much the way that PASCAL and MODULA-2 enforce structured programming, SMALLTALK enforces object-oriented programming, and Ada enforces some software engineering practices.

Perhaps the most dramatic changes of language will occur with the improvement and development of programming environments and tools. It is often the environment that captures

the programmer. The facilities of the LISP and C environments allow the programmer to concentrate on the task at hand, and quickly and efficiently produce the needed code. This is especially true of a LISP environment on a LISP machine. The programmer can build his own tools and tailor the environment to his or her needs and preferences. More importantly, the environment and machine function in harmony to allow the programmer to build new languages in which problems can be solved. By allowing a measure of abstraction, generality and efficiency can be gained. All of these things taken together point out that the developmental environment is an important factor in selecting a languages.

As programming tools and aids evolve, the direct contact with the programming language may begin to disappear. Such tools may allow the programmer to either break the programming task down into parts that are sufficiently small and standard that existing libraries of routines can be employed, or may allow the programmer to build the program specifications in such a way that a translator will be able to translate the specification into the target language. Both approaches currently have their problems. In the former the programmer is left at some point to grapple with the language itself, and in the latter the programmer might find the translated code for the target language indecipherable. Although these are serious problems, they may not be insurmountable. If they can be overcome, the contact of the programmer with the programming language will be stretched thinner and thinner.

The continued improvement of programming tools and environments, may lead the manager to base his or her decision on which programming language to use on the presence or absence of certain features in the tools and environments more than on the characteristics of the languages. The decision, of course, is still affected by external factors. Ada will be used on the Space Station. However, much might be learned by examining the environments and tools for other languages such as LISP and C in an effort to build better tools for Ada. After all, given that Ada is a sufficiently universal language, it can be made to look like other languages.

CONCLUSION

It is difficult if not impossible to directly solve the cultural collisions that are bound to occur in the interaction of programming languages, and paradigms. Those cultural collisions will not be resolved by attempting to enforce a uniform programming language and culture. An alternative, however, is to build tools that remove the programmer from direct contact with the programming language. This removal can allow the tool user to overcome the cultural problems, while still allowing the production of code in a desired language. If and when such tools become available, the questions with which this essay started will be displaced with the question, "What can your tool do?"

ACKNOWLEDGEMENTS

This research has been partially funded under NAS8-36955 (Marshall Space Flight Center) D.O. 34 "Applications of Artificial Intelligence to Space Station."

REFERENCES

1. BOEHM, B.W. "Seven Basic Principles of Software Engineering," *The Journal of Systems and Software* 3, (1983), pp. 3-24.
2. BOOCH, G. *Software Components with Ada: Structures, Tools, and Subsystems*. The Benjamin/Cummings Publishing Company, Inc., Menlo Park, California, 1987.
3. Brintsenhoff, Alton; Christensen, Greco, Joe; Steve; Mangan, John. "The Use of Ada Concurrent Processing Features in an Implementation of Parallel Tree Searching Algorithms. *Proceedings of the Third Annual Conference on Artificial Intelligence and Ada*. George Mason University, October, 1987.
4. FAIRLEY, R.E. *Software Engineering Concepts*. McGraw-Hill Book Company, New York, 1985.
5. HOWDEN, W.E. "Contemporary Software Development Environments," *Communications of the ACM*, 25, 5 (1982), pp. 318-329.
6. REEKER, LARRY H.; KREUTER, JOHN; WAUCHOPE, KENNETH. "Artificial Intelligence: Pattern-Directed Processing." Final Report AFHRL-TR-85-12 Air Force Human Resources Laboratory, Lowry Air Force Base, Colorado. May 1985.
7. SCHWARTZ, RICHARD L AND MELLIAR-SMITH, P.M. "On the Suitability of Ada for Artificial Intelligence Applications." Final Report for Defence Advanced Research Projects Agency Contract DAAG29-79-C-0216. July 1980.
8. STROUSTRUP, B. "What is 'Object-Oriented Programming'?" *ECOOP '87: European Conference on Object-Oriented Programming, Paris, France, June 15-17, 1987, Proceedings*. [Bézivin, J., P. Cointe, J.-M. Hullot, and H. Lieberman (Eds.)]. *Lecture Notes in Computer Science* (276), Goos, G. and J. Hartmanis (Eds.), Springer-Verlag, Berlin, 1987.
9. WARREN, D. H. D.; PEREIRA, L. M.; PEREIRA, F. "PROLOG - the language and its implementation compared with LISP." *Proceedings of the ACM Symposium on Artificial Intelligence and Programming Languages*, Rochester, New York, pp 109-115. 1977.
10. ZELKOWITZ, M.V., A.C. SHAW, and J.D. GANNON. *Principles of Software Engineering and Design*. Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

Index

Adler, Richard M.	231	Gilstrap, Lewey	147
Babin, Brian	281	Gonzales, Avelino J.	393
Bellman, Kirstie L.	497	Graves, Sara J.	525
Bhatt, Rahul	429	Greene, Richard J.	317
Biegl, Csaba A.	203	Grisham, Tollie	551
Biglari, Haik	373	Hackey, Keith	507
Blokland, W.	165	Hadden, George D.	175
Bock, Conrad	557	Han, Chia Yung	103
Bonasso, R. Peter	257	Harrington, Jim	175
Bond, W. E.	507	Hinman, Elaine	551
Boose, John H.	271	Ho, H.	457
Bradshaw, Jeffrey M.	271	Hoey, John	351
Butcher, A.	545	Holmes, Willard M.	421
Carlisle, W. Homer	567	Howard, E. Davis, III	139
Carmody, Cora	185	Huang, Shie-rei	429
Casadaban, Cyprian E.	517	Hull, Larry G.	147
Case, C.	95	Iverson, David L.	341
Chang, Kai-Hsiung	567	Jackson, Robert	11
Chen, Alexander Y.	331	Janik, D. F.	155
Chen, I.	535	Johannes, James D.	113, 129
Chi, Sung-Do	25	Johnson, Jim	373
Coker, Cindy	551	Johnston, Mark	11
Cook, George E.	203	Karr, C. L.	43
Cottman, Bruce H.	231	Kaukler, William F.	59
Cox, Preston A.	291	Keleher, William	567
Craig, F. G.	95, 211	Kiss, Peter A.	315
Cross, James H.	567	Kladke, Robin R.	393
Cutts, D. E.	95, 211	Kurth, R. E.	457
Das, A.	545	Landry, Steve P.	121
Davis, William S.	1	Lehman, Larry	373
Delcambre, Lois M. L.	121	Lennington, Kent	185
Dewberry, Brandon	193	Lewis, Steven M.	497
Dugan, Tim	185	Loganatharaj, Rasiah	281, 403
Feldman, Roy	557	Luckhardt-Redfield, Carol A.	467
Fennel, T. R.	95, 113, 211	Meredith, D. L.	43
Fernandez, Kenneth R.	203	McKenzie, Frederic D.	393
Fiala, Harvey E.	577	Miller, Glenn	11
Filman, Robert E.	557	Modesitt, Kenneth L.	301, 487
Forbes, John H.	291	Mohapatra, Saroj Kumar	361
Freeman, L. M.	43	Moseley, Warren	477
Gholdston, E. W.	155		

Myler, Harley R.	393	Yeager, Dorian P.	85
Narayan, Srinii	69	Zeilinggold, Daphna	351
Nelson, Bob	185	Ziegler, Bernard P.	25
Newell, J. F.	457		
Newton, K. A.	155		
Padalkar, S.	165		
Palmer, J. R.	95		
Patterson-Hine, F. A.	341		
Pulaski, Kirt	305		
Purves, B.	211		
Reddy, Y. V.	545		
Riedesel, Joel	241		
Rochowiak, Daniel	477, 589		
Rogers, John	19, 361		
Rubin, Stuart H.	383		
Sabharwal, C. L.	507		
St. Clair, D. C.	507		
Sary, Charisse	147		
Schaefer, Phil	33		
Selig, William John	129		
Shackelford, Keith	567		
Sharma, D. D.	69, 429		
Shema, David B.	271		
Shiva, Sajjin G.	439		
Singh, H.	545		
Sponsler, Jeff	11		
Springfield, James F.	203		
Sridharan, N. S.	429		
Sztipanovits, J.	165		
Tanner, Steve	525		
Tillotson, Brian	325		
Toms, David	175		
Towhidnejad, Massood	393		
Varghese, Joseph	403		
Vick, Shon	11		
Wakefield, G. Steve	53		
Walls, Bryan	241		
Wan, Liqun	103		
Wee, William G.	103		
Williams, Douglas	439		
Woods, Donald	221		
Workman, Gary L.	59, 551		



Report Documentation Page

1. Report No. NASA CP-3073	2. Government Accession No.	3. Recipient's Catalog No.	
4. Title and Subtitle Fifth Conference on Artificial Intelligence for Space Applications		5. Report Date May 1990	6. Performing Organization Code
		8. Performing Organization Report No.	
7. Author(s) S. L. O'Dell, Compiler		10. Work Unit No. M-627	
		11. Contract or Grant No.	
9. Performing Organization Name and Address George C. Marshall Space Flight Center Marshall Space Flight Center, Alabama 35812		13. Type of Report and Period Covered Conference Publication	
		14. Sponsoring Agency Code	
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D. C. 20546		15. Supplementary Notes Co-sponsored by the University of Alabama in Huntsville, the Huntsville Chapter of the IEEE, and the Alabama-Mississippi Section of the AIAA.	
16. Abstract Proceedings of a conference held in Huntsville, Alabama, on May 22-23, 1990. The Fifth Conference on Artificial Intelligence for Space Applications brings together diverse technical and scientific work in order to help those who employ AI methods in space applications to identify common goals and to address issues of general interest in the AI community. Topics include the following: automation for Space Station; intelligent control, testing, and fault diagnosis; robotics and vision; planning and scheduling; simulation, modeling, and tutoring; development tools and automatic programming; knowledge representation and acquisition; and knowledge-base/data-base integration.			
17. Key Words (Suggested by Author(s)) Artificial Intelligence Space Station Automation Knowledge-Base Systems		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 616	22. Price A99

NASA FORM 1626 OCT 86

For sale by the National Technical Information Service, Springfield, VA 22161-2171

