NASA Conference Publication 10085

2nd CLIPS Conference Proceedings Volume 1

(NASA-CP-10085-Vol-1) SECOND CLIPS CONFERENCE PROCEEDINGS, VOLUME 1 (NASA) 232 p CSCL 09B

N92-16568 --THRU-- N92-16589 Unclas G3/61 0057680

Proceedings of a conference sponsored by the CLIPS Users Group and hosted by Lyndon B. Johnson Space Center Houston, Texas September 23-25, 1991

NASA

NASA Conference Publication 10085

2nd CLIPS Conference Proceedings Volume 1

Proceedings of a conference sponsored by the CLIPS Users Group and hosted by Lyndon B. Johnson Space Center Houston, Texas September 23-25, 1991

NASA

NASA Conference Publication 10085

2nd CLIPS Conference Proceedings Volume 1

Joseph Giarratano, Editor
University of Houston - Clear Lake
Houston, Texas

Christopher Culbert, Editor
NASA Lyndon B. Johnson Space Center
Houston, Texas

Proceedings of a conference sponsored by
the CLIPS Users Group and hosted by
Lyndon B. Johnson Space Center
Houston, Texas
September 23-25, 1991

National Aeronautics
and Space Administration
**Scientific and Technical
Information Branch
1991**

FOREWORD

I am very pleased to have helped bring about the compilation and presentation of the papers in these proceedings for the 2nd CLIPS conference. The papers provide a breadth of topics from teaching with CLIPS to extensions of the CLIPS code. The many applications of CLIPS attest to its worth as a focus for the conference and to the appropriateness of the CLIPS Users Group itself.

As president of the CLIPS Users Group, I was particularly happy to be able to see many of the people with whom I have had contact over the past year. It was also rewarding to observe the richness of the interchange between conference attenders. The efforts of the presenters, helpers, and organizers seemed to be genuinely appreciated.

In addition to the professional interaction associated with the presentations and demonstrations, the banquet and receptions provided an opportunity for social interaction as well. It was good to see that many spouses attended these activities. Without question, the excellent presentation by astronaut Dr. Linda Godwin provided an ideal setting for an evening of good will and enjoyment. All of us who attended the conference can be proud for helping to provide an enhancing experience for everyone.

The quality of the papers presented can be readily observed in these proceedings. However, we would also like for the reader to be aware of the significant benefits that come from attending the conference and actively participating in the forum of discussion that is focused through CLIPS.

The reader is also encouraged to consider joining the CLIPS Users Group. The new CLIPS Board and the new slate of officers are enthusiastically planning more benefits to membership and the next CLIPS Conference.

To Bob Savely, Chris Culbert, Gary Riley, Brian Donnell, Joe Giarratano, Carl Armstrong, Marty Buff, Linda Cook, Linda Martin, Matt Berry, and Pat Mortensen -- and to all of the members of the CLIPS family of users, I send as my last official act as president of the CLIPS Users Group,

Best Wishes!

Len Myers

PRECEDING PAGE BLANK NOT FILMED

1 October 1991
San Luis Obispo, California

Fellow CLIPS users,

It is my distinct pleasure to provide some remarks to accompany the Proceedings of the Second Annual CLIPS Conference. The conference was both enjoyable and productive, and a quick look through these papers will confirm the many successes being had using the CLIPS language.

A successful conference requires the hard work of many people. I would like to personally thank everyone who helped out with this conference and especially thank Dr. Joe Giarratano, Gary Riley, Brian Donnell and Carla Armstrong for all their hard work. Last, but not least, I would like to recognize and thank my predecessor, Dr. Len Myers, whose tireless efforts on behalf of the Users Group and the Conference have not gone unnoticed. In recognition of his unwavering support the CLIPS Users Group Board of Directors has established the "Len Myers Best Student Paper Award" to be awarded at each subsequent CLIPS Conference. Thank you Len.

Finally, I would like to encourage you to join the CLIPS Users Group. Besides sponsorship of the Third Annual CLIPS Conference we have several other activities planned including the continuation of our newsletter, establishment of an online CLIPS applications library and continued liaison with the CLIPS developers. With your support we can make our plans a reality.

Looking forward to seeing you at the next CLIPS conference.

Cheers!

Doyle

Rodney Doyle Raines III
rraines@polyslo.calpoly.edu
President
CLIPS Users Group. Inc.

PRECEDING PAGE BLANK NOT FILMED

NASA Johnson Space Center
SOFTWARE TECHNOLOGY BRANCH

COSMIC
382 E. Broad St.
Athens, GA 30602
(404) 542-3265

CLIPS

A Tool for the Development and Delivery of Expert Systems

CLIPS is a productive development and delivery expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. CLIPS was created by the Software Technology Branch of the Information Systems Directorate at NASA/Johnson Space Center (JSC) with support from the USAF. CLIPS is the first language to provide a verification/validation utility for the development of expert systems. CLIPS enabled the use of expert system technology in the JSC Mission Control Center and is being used by over 3,000 users throughout the public and private community including: all NASA sites and branches of the military, numerous federal bureaus, government contractors, 160 universities, and many companies. The key features of CLIPS are:

- **Knowledge Representation:** CLIPS 5.0 provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural. Rule-based programming allows knowledge to represented as heuristics, or "rules of thumb", which specify a set of actions to be performed for a given situation. Object-oriented programming allows complex systems to be modeled as modular components (which can be easily reused to model other systems or to create new components). The procedural programming capabilities provided by CLIPS are similar to capabilities found in languages such as C, Pascal, Ada, and LISP.
- **Portability:** CLIPS is written in C for portability and speed and has been installed on many different computers without code changes. Computers on which CLIPS has been tested include IBM PC compatibles, Macintosh, VAX 11/780, Sun 3/260, and HP9000-500. CLIPS can be ported to any system which has an ANSI compliant C compiler. CLIPS comes with *all* source code, approximately 40,000 lines of C, which can be modified or tailored to meet a user's specific needs.
- **Integration/Extensibility:** CLIPS can be embedded within procedural code, called as a subroutine, and integrated with languages such as C, FORTRAN and ADA. CLIPS can be easily extended by a user through the use of several well-defined protocols.
- **Interactive Development:** The standard version of CLIPS provides an interactive, text oriented development environment, including debugging aids, on-line help, and an integrated editor. An interface providing features such as pulldown menus, an integrated editor, and multiple windows has been developed for the Macintosh (and similar interfaces will soon be available for MS-DOS, Windows 3.0, and X-Windows environments).
- **Verification/Validation:** A unique utility called CRSV aids in verification and validation of rules by providing cross referencing of patterns, style checking, and semantic error checking.
- **Fully Documented:** CLIPS comes with extensive documentation including a full Reference Manual and a User's Guide. An Architecture Manual provides a guide to the source code and internal operations of CLIPS. A 650-page college textbook, *Expert Systems Principles and Programming,* using CLIPS is available from PWS Kent publishers.
- **ADA Version:** A version of CLIPS developed entirely in Ada and fully syntax compatible with the C version of CLIPS is currently available for VAX and UNIX workstations.
- **Availability:** CLIPS version 5.0 is currently available. CLIPS is free to NASA, USAF, and their contractors for use on NASA and USAF projects by calling the Software Technology Branch Help Desk between the hours of 9:00 AM to 4:00 PM (CST) Monday through Friday at (713) 280-2233. Government contractors should have their contract monitor call the Software Technology Branch Help desk to obtain CLIPS. Others may purchase CLIPS (including all documentation) from COSMIC at a nominal fee for unlimited copies with no royalties. An electronic bulletin board containing information regarding CLIPS can be reached 24 hours a day at (713) 280-3896 or (713) 280-3892. Communications information is 300, 1200, or 2400 baud, no parity, 8 data bits, and 1 stop bit.

2/7/91

PRECEDING PAGE BLANK NOT FILMED

NASA Johnson Space Center
SOFTWARE TECHNOLOGY BRANCH

COSMIC
382 E. Broad St.
Athens, GA 30602
(404) 542-3265

CLIPS

Version 5.0 Announcement

The Software Technology Branch of the Information Technology Division at NASA/Johnson Space Center announces the upcoming release of version 5.0 of CLIPS. CLIPS is a powerful development and delivery expert system tool which provides a complete environment for the construction of rule-based expert systems. CLIPS is free to NASA, USAF, and their contractors for use on NASA and USAF projects by calling the CLIPS Help Desk between the hours of 9:00 AM to 4:00 PM (CST) Monday through Friday at (713) 280-2233. Government contractors should have their contract monitor call the CLIPS Help desk to obtain CLIPS. Others may purchase CLIPS from COSMIC. The key new features of CLIPS 5.0 are:

- **Object Oriented Programming:** The primary addition to version 5.0 of CLIPS is the CLIPS Object-Oriented Language (COOL). COOL supports many of the features found in commercial object-oriented languages such as Common Lisp Object System (CLOS) and SmallTalk. The features of COOL include: classes with multiple inheritance, single and multi-valued slots (with slot daemons), instances with encapsulation, and message-passing (with before, after, primary, and around message-handlers). COOL is not tightly integrated with the rule system (i.e. you cannot pattern match against instances on the LHS of a rule, and rules are not instances of the rule class). However, an extensive query system is provided for finding and/or performing actions on sets of instances which meet arbitrary user-defined restrictions. Coordination between rules and objects can be achieved easily by explicit programmer control. (A future release of CLIPS will support direct pattern-matching on objects).
- **Deffunctions:** Procedural code (called deffunctions) can be defined directly in CLIPS. Deffunctions may be used to add new capabilities to CLIPS without having to write new code in C and recompile CLIPS.
- **Generic-Functions:** Procedural code (called generic-functions) can be defined directly in CLIPS. Generic-functions can be used to overload functions in a manner similar to, but much more powerful than, languages such as Ada and C++. A user may also use generic-functions to add new capabilities to CLIPS without having to write new code in C and recompile CLIPS.
- **Global Variables:** Variables that are global in scope may be defined in a manner similar to procedural languages. These variables can then be accessed or set from within rules, message-handlers, generic-function methods, etc.
- **Integer Data Type Support:** An integer data type is supported which is represented internally as a C long integer. Floating point values are now stored internally as C double precision numbers for greater accuracy.
- **Conflict Resolution Strategies and Salience Extensions:** Seven conflict resolution strategies are provided including depth, breadth, lex, mea, simplicity, complexity, and random. Salience values can be expressions and contain global variables. Salience values can also be dynamically evaluated each time a new activation is added to the agenda or every cycle of execution.
- **Deftemplate Field Checking:** Type, value, and range checking for deftemplate field values are now supported both statically (when rules are loaded) and dynamically (when new facts are asserted) .
- **Incremental Reset:** Newly added rules are automatically initialized when defined, and any new activations are placed on the agenda. Rules may also be "refreshed" which adds previously executed activations which are still valid for a rule to the agenda.
- **Truth Maintenance:** Facts can be made logically dependent upon the existence or non-existence of other facts.

2/7/91

CONTENTS

Volume 1

2ND CLIPS CONFERENCE

AGENDA

Gilruth Center. Johnson Space Center. Houston, TX.

September 23 - 25, 1991

sponsored by the CLIPS Users Group with support by NASA/Johnson Space Center

* *

welcome

2ND CLIPS CONFERENCE

September 23 - 25, 1991

Gilruth Center. Johnson Space Center. Houston, TX.

sponsored by the CLIPS Users Group with support by NASA/Johnson Space Center

Co-directors:	Joseph Giarratano	**Administrative Chair:**	Carla Armstrong
	Leonard Myers	**Registration Chair:**	Marlon Buff

* *

CLIPS Users Group

The CLIPS Users Group is a nonprofit corporation organized to promote the interchange of information about CLIPS, the C Language Integrated Production System developed by the Software Technology Branch of the Information Technology Division at NASA/Johnson Space Center.

The CLIPS Users Group Conference is open to all parties who pay the appropriate registration fees. Although the presentations are associated with the use of CLIPS, all persons interested in knowledge based systems will find topics of general usefulness in the field.

CLIPS Users Group Officers 1990-1991:

President	Leonard Myers	California Polytechnic State University (CAL POLY)
Vice-President	Matthew Barry	Rockwell International Corporation
Vice-President	Ray Foster	United States Army
Treasurer	Marlon Buff	United States Army
Secretary	Linda Cook	Independent Consultant
Secretary	Linda Martin	Rockwell International Corporation

- -

*** * * We gratefully acknowledge the assistance of June Muecke and the Clear Lake Area Convention and Visitors Bureau for providing the "goodie bags" and registration services. * * ***

- -

AGENDA

MONDAY, SEPTEMBER 23

07:30 - 16:00 **Registration** — Main Entrance Gilruth Center

08:00 - 08:15 **WELCOME ADDRESS** — **Ballroom**

Host: Joseph Giarratano
Speaker: Aaron Cohen, Director, Johnson Space Center

08:15 - 08:45 **CLIPS - Visions** — **Ballroom**

Host: Joseph Giarratano
Speakers: Robert Savely, Software Technology Branch, NASA/JSC.
Chris Culbert, Software Technology Branch, NASA/JSC.

The status and direction of CLIPS will be discussed.

PRECEDING PAGE BLANK NOT FILMED

08:45 - 09:45 **SESSION 1** **Ballroom**

Continuity *Host:* Leonard Myers

The presenters for this session were asked to followup on the papers they presented at the first CLIPS conference.

08:45 - 9:15 **Rule Groupings: An Approach towards Verification of Expert Systems.** Mala Mehrotra, Vigyan Inc.

Knowledge-based expert systems are playing an increasingly important role in NASA space and aircraft systems. However, many of NASA's software applications are life- or mission-critical and knowledge-based systems do not lend themselves to the traditional verification and validation techniques for highly reliable software. Rule-based systems lack the control abstractions found in procedural languages. Hence, it is difficult to verify or maintain such systems. Our goal is to automatically structure a rule-based system into a set of rule-groups having a well-defined interface to other rule-groups.

9:15 - 9:45 **Enhanced Use of CLIPS at the Los Alamos National Laboratory.** K. H. Duerre, W. J. Parkinson, J. J. Osowski, Los Alamos National Laboratory.

Early efforts in producing Expert Systems for engineering applications used a limited subset of CLIPS features. In this paper we discuss the implementation details of previous Expert Systems and of the current Expert System, which is used for training operators in the control of Isotope Separation System.

09:45 - 10:00 **Refreshment Break**

10:00 - 11:30 **SESSION 2 CONCURRENT GROUPS, A AND B**

Group 2A: **Ballroom**

Host: Yuh-jeng Lee

10:00 - 10:30 **Using a CLIPS Expert System to Automatically Manage TCP/IP Networks and Their Components.** Ben Faul, TRW.

This paper describes an expert system than can directly manage networks components on a TCP/IP network. Previous expert systems for managing networks have focused on managing network faults after they occur. However this proactive expert system can monitor and control network components in near real time. The ability to manage directly network elements from CLIPS is accomplished by the integration of the Simple Network Management Protocol (SNMP) and an Abstract Syntax Notation (ASN) parser into the CLIPS artificial intelligence language.

10:30 - 11:00 **NMESys: An Expert System for Network Fault Detection.** Peter Nelson and Janet Warpinski, University of Illinois at Chicago.

The problem of network management is becoming an increasingly difficult and challenging task. It is very common today to find heterogeneous networks consisting of many different types of computers, operating systems, and protocols. The complexity of implementing a network with this many components is difficult enough, while the maintenance of such a network is an even larger problem. This paper presents a prototype network management expert system, NMESys (pronounced nemesis), implemented in CLIPS. NMESys concentrates on solving some of the critical problems encountered in managing a large network. The major goal of NMESys is to provide a network operator with an expert system tool to quickly and accurately detect hard failures, potential failures, and to minimize or eliminate user down time in a large network.

11:00 - 11:30 **A Mission Executor for an Autonomous Underwater Vehicle.**

Yuh-jeng Lee and Paul Wilkinson, Naval Postgraduate School.

The Naval Postgraduate School has been conducting research into the design and testing of an Autonomous Underwater Vehicle (AUV). One facet of this research is to incrementally design a software architecture and implement it in an advanced testbed, the AUV II. As part of the high level architecture, a Mission Executor is being constructed using CLIPS version 5.0 The Mission Executor is an expert system designed to oversee progress from the AUV launch point to a goal area, and back to the origin. it is expected that the Executor will make informed decisions about the mission, taking into account the navigational path, the vehicle subsystems health and the sea environment, as well as the specific mission profile which is downloaded from an offboard mission planner. Heuristics for maneuvering, avoidance of uncharted obstacles, waypoint navigation, and reaction to emergencies (essentially the expert knowledge of a submarine captain) are required. Many of the vehicle subsystems are modeled as objects using the CLIPS Object Oriented Language (COOL) embedded in CLIPS 5.0 Additionally, truth maintenance is applied to the knowledge base to keep configurations updated.

Group 2B: Room 217

Host: Michael Steib

10:00 - 10:30 **The Automated Army ROTC Questionaire (ARQ).**

David Young, U. S. Army.

The Reserve Officer Training Corps Cadet Command (ROTCCC) takes applications for its officer training program from college students and Army enlisted personnel worldwide. ROTCCC asked the Artificial Intelligence Center at Fort Monroe for an inexpensive and reliable way of automating their application process. After reviewing the process, the Center determined that an expert system with good end-user interface capabilities could be used to solve a large part of the problem. The system captures the knowledge contained within the regulations, enables the quick distribution and implementation of eligibility criteria changes, and distributes the expertise of the admissions personnel at Cadet Command to the education centers and colleges worldwide. The expert system uses a modified version of CLIPS that was streamlined to make the most efficient use of its capabilities. A user interface with windowing capabilities provides the applicant with simple and effective way to input his/her personal data.

10:30 - 11:00 **Decision Blocks: A Tool for Automating Decision Making in CLIPS.** Christoph Eick, University of Houston and Nikhil Mehta, GE Government Services.

The human capability of making complex decision is one of the most fascinating facets of human intelligence, especially if vague, judgmental, default or uncertain knowledge is involved. Unfortunately, most existing rule-base forward-chaining languages are not very suitable to simulate this aspect of human intelligence, because of the lack of support for approximate reasoning techniques needed for this task, , and due to the lack of specific constructs to facilitate the coding of frequently reoccurring activities in decision making processes. The paper advocates to extend CLIPS by a new component called decision block to provide better support for the design and implementation of rule-based decision support systems. A language called BIRBAL, which is defined on the top of CLIPS, for the specification of decision blocks is introduced. Empirical experiments involving the comparison of the length of CLIPS-program with the corresponding BIRBAL-program for three different applications are surveyed. The results of these experiments suggest that for decision making intensive applications a CLIPS-program tends to be about three times longer than the corresponding BIRBAL-program.

11:00 - 11:30 **Automated Predictive Diagnosis (APD): A Three Tiered Shell for Building Expert Systems for Automated Predictions and Decision Making.**

Michael Steib, Vitro Corporation.

The APD software features include: On-line help, Three-level architecture, (Logic environment, Setup/Application environment, Data environment), Explanation capability, and File handling. The kinds of experimentation and record keeping that leads to effective expert systems is facilitated by: a) a library of inferencing modules (in the logic environment), b) An explanation capability which reveals logic strategies to users, c) Automated file naming conventions, d) An information retrieval system, e) On-line help. These aid with effective use of knowledge, debugging and experimentation. Since the APD software anticipates the logical rules becoming complicated, it is imbedded in a production system language (CLIPS) to insure the full power of the production system paradigm of CLIPS and availability of the procedural language C. This paper discusses the development of the APD software and three example applications: toy, experimental, and operational prototype for submarine maintenance predictions.

11:35 - 12:25 **SPECIAL DEMONSTRATION*** **Ballroom**
Host: Leonard Myers

ICADS: A Cooperative Decision Making Model With CLIPS Experts.
Yens Pohl and Leonard Myers, California Polytechnic State University

This paper describes a cooperative decision making model comprising six concurrently executing domain experts coordinated by a blackboard control expert. The focus application field is architectural design, and the domain experts represent consultants in the areas of daylighting, noise control, structural support, cost estimating, space planning, and climate responsiveness. Both the domain experts and the blackboard have been implemented as production systems, utilizing an enhanced version of the basic CLIPS package. Acting in unison as an Expert Design Advisor, the domain and control experts react to the evolving design solution progressively developed by the user in a 2-D CAD drawing environment. A Geometry Interpreter maps each drawing action taken by the user to real world objects, such as spaces, walls, windows, and doors. These objects, endowed with geometric and non-geometric attributes, are stored as frames in a semantic network. Object descriptions are derived partly from the geometry of the drawing environment and partly from knowledge bases containing prototypical, generalized information about the building type and site conditions under consideration.

*** special thanks to IBM Corporation and Buster Malenek for the loan of an RS/6000 system and extra installation of software for the demonstration.**

12:30 - 13:25 **LUNCH** **Ballroom**

Note: Room 206 is the Demonstration Room. Feel free to use the equipment to demonstrate, test and share CLIPS software. Please observe common courtesy protocol and all legal requirements. If you have any problems with the equipment and can not find the room monitor person, report your difficulty to the registration desk.

Also Note: The announcement of CLIPS 5.1 availability will be made in the CLIPS Technical Session, following Session 3.

13:30 - 15:00 SESSION 3 CONCURRENT GROUPS, A AND B

Group 3A: **Ballroom**

Host: J. R. Stagner

13:30 - 14:00 **A CLIPS/X-Window Interface.**
Kym Pohl, CAL POLY.

This paper describes the design and implementation of an interface between the CLIPS expert system development environment and the graphic user interface development tools of the X-Window system. The underlying basis of the CLIPS/X-Window interface is a client-server model in which multiple clients can attach to a single server that interprets, executes and returns operation results, in response to client action requests. Implemented in an AIX (Unix) operating system environment, the interface has been successfully applied in the development of graphics interfaces for production rule cooperating agents in a knowledge-based CAD system. Initial findings suggest that the client-server model is particularly well suited to a distributed parallel processing operational mode in a networked workstation environment.

14:00 - 14:30 **Application of Machine Learning and Expert Systems to Statistical Process Control (SPC) Chart Interpretation.**
Mark Shewhart, Air Force Logistics Command.

Statistical Process Control (SPC) Charts are one of several tools used in Quality Control. Other tools include flow charts, histograms, cause-and-effect diagrams, check sheets, Pareto diagrams, graphs, and scatter diagrams. A control chart is simply a graph which indicates process variation over time. The purpose of drawing a control chart is to detect any changes in the process, signalled by abnormal points or patterns on the graph. The Artificial Intelligence Support Center (AISC) of the Acquisition Logistics Division (ALD/JTI) has developed a hybrid machine-learning/expert-system prototype which automates the process of constructing and interpreting control charts.

14:30 - 15:00 **Application of Software Technology to Automatic Test Data Analysis.** J. R. Stagner, Jet Propulsion Laboratory.

The verification process for a major software subsystem was partially automated as part of a feasibility demonstration. The methods employed are generally useful and applicable to other types of subsystems. The effort resulted in substantial savings in test engineer analysis time and offers a method for inclusion of automatic verification as part of regression testing.

Group 3B: **Room 217**

Host: Kwok-bun Yue

13:30 - 14:00 **Acquisition, Representation and Rule Generation for Procedural Knowledge.** Chris Ortiz, Johnson Space Center, Sachin Mithal, Tim Saito, Computer Sciences Corporation, R. Loftin, University of Houston.

This paper describes current research into the design and continuing development of a system for the acquisition of procedural knowledge, its representation in useful forms, and proposed methods for automated CLIPS rule generation. TARGET (Task Analysis and Rule Generation Tool) is intended to permit experts, individually or collectively, to visually describe and refine procedural tasks. The system is designed to represent the acquired knowledge in the form of graphical objects with the capability for generating production rules in CLIPS. The generated rules can then be integrated into applications such as NASA's ICAT (Intelligent Computer Aided Training) architecture. The paper concludes by describing proposed methods for use in translating the graphical and intermediate knowledge representations into CLIPS rules. Systems such as TARGET have the potential to profoundly reduce the time, difficulties, and costs of developing knowledge-based systems for the performance of procedural tasks.

14:00 - 14:30 **Projects In An Expert System Class.**
George Whitson, The University of Texas at Tyler.

Many universities now teach courses in expert systems. In these courses students study the architecture of an expert system, knowledge acquisition techniques, methods of implementing expert systems and verification and validation techniques. A major component of any such course is a class project consisting of the design and implementation of an expert system. This paper discusses a number of techniques that we have used at The University of Texas at Tyler to develop meaningful projects that could be completed in a semester course.

14:30 - 15:00 **Using CLIPS as the Cornerstone of a Graduate Expert Systems Course.**
Kwok-bun Yue, University of Houston - Clear Lake.

This short article describes the effective use of CLIPS as the cornerstone in a graduate expert systems course. The course included about 8 to 9 hours of in-depth lecturing in CLIPS, as well as a broad coverage of major topics and techniques in expert systems. As part of the requirement of the course, students solved two small yet non-trivial problems in CLIPS before going on to develop a toy expert system in CLIPS in an incremental manner as the term project. Furthermore, students were required to evaluate CLIPS programs by the classmates. An anonymous questionnaire at the end of the semester revealed that the students responded very favorably about the course, especially their experience with CLIPS.

15:00 - 16:30 CLIPS USERS GROUP MEETING **Ballroom**
Host: Leonard Myers

Secretary's Report
Treasurer's Report
By-Laws Report
Election of Board Members

NOTE: Five members are to be elected to the Board of Directors. This "CLIPS Board" will then be responsible for the selection of President, Vice-President, Secretary and Treasurer for the CLIPS Users Group. The officers need not be members of the Board, and the offices of Secretary and Treasurer may be held by one person.

It is anticipated that the new Board members will meet during the conference to discuss such matters as the selection of officers, the production of a newsletter, additional services of the Users Group, and the Third Conference.

The receptions on Monday and Tuesday will enable members to meet the Board members and provide suggestions for the oncoming year of CLIPS activities.

On Wednesday the Board will have an opportunity to address the members in the first meeting of the new CLIPS Board.

16:30 - 17:30 RECEPTION (Cash Bar) **Room 217**

17:30 - 19:00 BANQUET **Ballroom**
Host: Joseph Giarratano
Speaker: Astronaut Linda Godwin

We are delighted to have Linda provide us with a change of pace from our CLIPS emphasis and give us a bit of that unique experience that only those few who have been in space can relate. Linda's presentation will follow the dinner.

TUESDAY, SEPTEMBER 24

08:30 - 16:00 Late Registration — Main Entrance Gilruth Center

08:45 - 10:15 SESSION 4 **CONCURRENT GROUPS, A AND B**

Group 4A: **Ballroom**

Host: Stephen Scott

08:45 - 09:15 **CRN5EXP - Expert System for Statistical Quality Control.** Mariana Hentea, Interactive Business Systems.

The purpose of the Expert System CRN5EXP is to help the user to check the quality of the coils at two very important mills: Hot Rolling and Cold Rolling in a steel plant. The system interprets the statistical quality control charts, diagnoses and predicts the quality of the steel. Measurements of process control variables are recorded in database (ADABAS) and sample statistics such as the mean and the range are computed and plotted on a control chart. The chart is analyzed through patterns using CLIPS and forward chaining technique to reach a conclusion about the causes of defects and to take management measures for the improvement of the quality control techniques. The Expert System combines the certainty factors associated with the process control variables to predict the quality of the steel. The paper presents the approach to extract data from database, the reason to combine certainty factors, the architecture and the use of the Expert System. however, the interpretation of control charts patterns requires the human expert's knowledge and lends to Expert Systems rules. The conclusions reached with this system help the management and the quality engineers to eliminate the special causes of the process control variable variations and to correct about 85% of the problems from these mills.

09:15 - 09:45 **Distributed Semantic Networks and CLIPS.** James Snyder and Tony Rodriguez, CAL POLY.

Semantic networks of frames are commonly used as a method of organizing and reasoning in many types of problems. In most of these applications the semantic network exists as a single entity in a single process environment. Advances in workstation hardware provide support for more sophisticated applications involving multiple processes, interacting in a distributed environment. In these applications the semantic network may well be distributed over several concurrently executing tasks. This paper describes the design and implementation of a frame-based, distributed semantic network in which frames are accessed both through CLIPS expert systems and procedural C++ language programs. The application area is a knowledge-based, cooperative decision making model utilizing both rule-base and procedural experts.

09:45 - 10:15 **Object-Oriented Knowledge Representation for Expert Systems.** Stephen Scott, Hughes Information Technology.

Object-oriented techniques have generated considerable interest in the AI community in recent years. This paper discusses an approach for representing expert system knowledge using classes, objects, and message passing. The implementation is in version 4.3 of NASA's CLIPS, an expert system tool that does not provide direct support for object-oriented design. The method uses programmer-imposed conventions and keywords to structure facts, and rules to provide object-oriented capabilities.

Group 4B: **Room 217**

Host: Mark Miller

08:45 - 09:15 **LinkFinder: An Expert System That Constructs Phylogenic Trees.** James Inglehart and Peter Nelson, The University of Illinois at Chicago.

An expert system has been developed using CLIPS that automates the process of constructing DNA sequence-based phylogenies - trees or lineages that indicate evolutionary relationships. LinkFinder takes as input homologous DNA sequences from distinct individual organisms. It measures variations between the sequences, selects appropriate proportionality constants, and estimates (if possible) the time that has passed since each pair of organisms diverged from a common ancestor. it then designs and outputs a phylogenic map summarizing these results. LinkFinder can find genetic relationships between different species, and between individuals of the same species, including humans. It was designed to take advantage of the vast amount of sequence data being produced by the celebrated Genome Project, and should be of great value to evolution theorists who wish to utilize this dat, but who have no formal training in molecular genetics.

09:15 - 09:45 **Generating Target System Specifications From a Domain Model Using CLIPS.** Vijayan Sugumaran, Hassan Gomaa and Larry Kerschberg, George Mason University.

The quest for reuse in software engineering is still being pursued and researchers are actively investigating the domain modeling approach to software construction. There are several domain modeling efforts reported in the literature and they all agree that the components that are generated from domain modeling are more conducive to reuse. Once a domain model is created, several target systems can be generated by tailoring the domain model or by evolving the domain model an then tailoring it according to the specified requirements. This paper presents the Evolutionary Domain Life Cycle (EDLC) paradigm in which a domain model is created using multiple view, namely, aggregation hierarchy, generalization/specialization hierarchies, object communication diagrams and state transition diagrams. The architecture of the Knowledge Based Requirements Elicitation Tool (KBRET) which is used to generate target system specifications is also presented. The preliminary version of KBRET is implemented in CLIPS.

09:45 - 10:15 **The Management and Security Expert (MASE).** Mark Miller, Stanley Barr, Coranth Gryphon, Jeff Keegan, Catherine Kniker and Patrick Krolak, University of Massachusetts at Lowell.

Today's computing environments are increasingly complex: they often consist of large networks of computers that include multiple vendors and operating systems. The various systems and the communications between them must be kept running at their peak performance levels to meet the demands of the user community. They must also be kept safe from malevolent intruders and damaging viruses. The Management and Security Expert (MASE) can help. MASE is a distributed expert system that monitors the operating systems and applications of a network. It is capable of gleaning the information provided by the different operating systems in order to optimize hardware and software performance; recognize potential hardware and/or software failure, and either repair the problem before it becomes an emergency, or notify the systems manager of the problem; and monitor applications and known security holes for indications of an intruder or virus. MASE can eradicate much of the guess work of system management.

10:15 - 10:30 Refreshment Break

10:30 - 12:30 SESSION 5 CONCURRENT GROUPS, A AND B

Group 5A: **Ballroom**

Host: Peter Homeier

10:30 - 11:00 **Adding Run History to CLIPS.** Sharon Tuttle and Christoph Eick, University of Houston.

To debug a CLIPS program, certain 'historical' information about a run is needed. It would be convenient for system builders to be able to ask questions requesting such information. For example, system builders might want to ask why a particular rule did not fire at a certain time, especially if they think that it should have fired then, or they might want to know at what periods during a run a particular fact was in working memory. It would be less tedious to have such questions directly answered, instead of having to rerun the program one step at a time or having to examine a long trace file. This paper advocates extending the Rete network used in implementing CLIPS by a temporal dimension, allowing it to store 'historical' information about a run of a CLIPS program. We call this extended network a *historical Rete network*. To each fact an instantiation are appended *time-tags*, which encode the period(s) of time that the fact or instantiation was in effect. In addition, each Rete network memory node is partitioned into two sets: a *current* partition, containing the instantiations currently in effect, and a *past* partition, containing the instantiations which are not in effect now, but which were earlier in the current run. These partitions allow the basic Rete network operation to be surprisingly unchanged by the addition of time-tags and the resulting effect that no-longer-true instantiations now do not leave the Rete network.

11:00 - 11:30 **CLIPS Application User Interface for the PC.** Jim Jenkins, Rebecca Holbrook, Mark Shewhart, Joey Crouse and Stuart Yarost, Air Force Logistics Command.

The majority of applications that utilize expert system development programs for their knowledge representation and inferencing capability require some form of interface with the end user. This interface is more than likely an interaction through the computer screen. When building an application the user interface can prove to be the most difficult ant time consuming aspect to program. Commercial products currently exist which address this issue. To keep pace CLIPS will need to find a solution for their lack of an easy-to-use Application User Interface (AUI). This paper represents a survey of the DoD CLIPS' user community and provides the backbone of a possible solution.

11:30 - 12:00 **Expert Networks in CLIPS.** Susan Hrushka, A. Dalke, J. Ferguson and R. Lacher, Floriday State University.

Rule-based expert systems may be structurally and functionally mapped onto a special class of neural networks called *expert networks*. This mapping lends itself to adaptation of connectionist learning strategies for the expert networks. Following a process introduced by Kuncicky, Hruska, and Lacher, a parsing algorithm to translate CLIPS rules into a network of interconnected assertion and operation nodes has been developed. The translation of CLIPS rules to an expert network and back again is illustrated. Measures of uncertainty similar to those used in MYCIN-like systems are introduced into the CLIPS system and techniques for combining and firing nodes in the network based on rule-fireing with these certainty factors in the expert system are presented. Several learning algorithms are under study which automate the process of attaching certainty factors to rules.

12:00 - 12:30 **ECLIPS: An Extended CLIPS for Backward Chaining and Goal-Directed Reasoning.** Peter Homeier and Thach Le, The Aerospace Corporation.

Realistic production systems require an integrated combination of forward and backward reasoning to reflect appropriately the processes of natural human expert reasoning. a control mechanism that consists solely of forward reasoning is not an effective way to promptly focus the system's attention as calculation proceeds. Very often expert system programmers will attempt to compensate for this lack by using data to enforce the desired goal-directed control structure. This approach is inherently flawed in that it is attempting to use data to fulfil the role of control This paper will describe our implementation of backward chaining in CLIPS, and show how this has shortened and simplified various CLIPS programs.

Group 5B: **Room 217**

Host: Richard Adler

10:30 - 11:00 **Extensions to the Parallel Real-time Artificial Intelligence System (PRAIS) for Fault-tolerant Heterogeneous Cycle-stealing Reasoning.** David Goldstein, University of Texas, Arlington.

Extensions to an architecture for real-time, distributed (parallel) knowledge-based systems called the Parallel Real-time Artificial Intelligence System (PRAIS) are discussed. PRAIS strives for transparently parallelizing production (rule-based) systems, even under teal-time constraints. PRAIS accomplished these goals (presented at the first annual CLIPS conference) by incorporating a dynamic task scheduler, operating system extensions for the fact handling, and message-passing among multiple copies of CLIPS executing on a virtual blackboard. This distributed knowledge-based system tool uses the portability of CLIPS and common message-passing protocols to operate over a heterogeneous network of processors. Results using the original PRAIS architecture over a network of Sun 3's, Sun 4's and VAX;s are presented. Mechanisms using the producer-consumer model to extend the architecture for fault-tolerance and distributed truth maintenance initiation are also discussed. Also, recently designed approaches and extension, including improvements to RETE and an entirely new pattern matching algorithm to meet hard-real-time deadlines are discussed.

11:00 - 11:30 **PCLIPS - Parallel CLIPS.** Coranth Gryphon and Mark Miller, University of Massachusetts at Lowell.

PCLIPS (Parallel CLIPS) is a set of extensions to the CLIPS expert system language. PCLIPS is intended to provide an environment for the development of more complex, extensive expert systems. Multiple CLIPS expert systems are now capable of running simultaneously on separate processors, or separate machines, thus dramatically increasing the scope of solvable tasks within the expert systems. PCLIPS allows for an expert system to add to its fact-base information generated by other expert systems, thus allowing systems to assist each other in solving a complex problem. This allows individual expert systems to be more compact and efficient, and thus run faster or on smaller machines.

11:30 - 12:00 **Separating Domain and Control Knowledge using Agenda.** Paul Haley, The Haley Enterprise.

Controlling the behavior of CLIPS applications usually involves the use of salience and/or the introduction of control conditions into the conditions of most rules. Using more than a few levels of salience facilitates poor rule-based programming technique and inhibits incremental refinement and maintenance of a knowledge base by implicitly encoding procedural control throughout the knowledge base. To avoid the problem of overusing salience, developers of CLIPS (and OPS5) programs necessarily introduce conditions representing procedural state into their rules. When the rule base is embedded within an application, the controlling program, in addition to the rules themselves, assert and retract facts which instantiate the control conditions which are distributed throughout the rule base. Although effective, this technique is tedious, reduces sharing, increases match volatility, involves database overhead, and - most importantly - couples control and domain knowledge such that certain operators which may usefully be applied in several (but not all)(control contexts, must be implemented redundantly albeit with distinct control conditions. We describe an extension to CLIPS which allows a rule to be declared as belonging to one of several agenda. Using this capability, we show how the need for rule salience and control conditions can be eliminated from rules using a standard procedural language, such as C, to control the execution of rule activations among agenda. Examples are given that show how eliminating salience and control conditions from rules makes knowledge bases easier to comprehend. We also show how eliminating control information from rules improves efficiency and reduces space requirements. We conclude that, using these capabilities, any flow-chartable control flow can be imposed on the knowledge base without rule saliences or control conditions.

12:00 - 12:30 **Integrating CLIPS Applications into Heterogeneous Distributed Systems.** Richard Adler, Symbiotics, Inc.

SOCIAL is an advanced, object-oriented development tool for integrating intelligent and conventional applications across heterogeneous hardware and software platforms. SOCIAL defines a family of "wrapper" objects called *Agents*, which incorporate predefined capabilities for distributed communication and control. Developers embed applications within Agents and establish interactions between distributed Agents via non-intrusive message-based interfaces. This paper describes a predefined SOCIAL Agent that is specialized for integrating CLIPS-based applications. The Agent's high-level Application Programming Interface supports *bidirectional* flow of data, knowledge, and commands to other Agents, enabling CLIPS applications to initiate interactions autonomously, and respond to requests and results from heterogeneous, remote systems. The design and operation of CLIPS Agents is illustrated with two distributed applications that integrate CLIPS-based expert systems with other intelligent systems for isolating and managing problems in the Space Shuttle Launch Processing System at NASA Kennedy Space Center.

12:30 - 13:25 LUNCH Ballroom

13:30 - 14:30 CLIPS SPECIAL SESSION Ballroom

Hosts: Brian Donnell and Gary Riley

Brian Donnell and Gary Riley will make presentations and answer technical questions about CLIPS.

The availability of CLIPS 5.1 will be announced.

14:30 - 14:45 <u>Refreshment Break</u>

14:45 - 16:15 <u>SESSION</u> 6 **CONCURRENT GROUPS, A AND B**

<u>Group 6A:</u> **Room 204**

Host: Ken Porter

14:45 - 15:15 Data-Driven Backward Chaining.

Paul Haley, The Haley Enterprise, Inc.

CLIPS cannot effectively perform sound and complete logical inference in most real-world contexts. The problem facing CLIPS is its lack of goal generation. Without automatic goal generation and maintenance, Forward chaining can only deduce all instances of a relationship. Backward chaining, which requires goal generation, allows deduction of only that subset of what is logically true which is also relevant to ongoing problem solving. Goal generation can be mimicked in simple cases using forward chaining. However, such mimicry requires manual coding of additional rules which can assert an inadequate goal representation for every condition in every rule that can have corresponding facts derived by backward chaining. in general, for N rules with an average of M conditions per rule the number of goal generation rules required is on the order of N*M. This is clearly intractable from a program maintenance perspective. We describe the support in Eclipse for backward chaining which automatically asserts as it checks rule conditions. Important characteristics of this extension are that it does not asset goals which cannot match any rule conditions, that two equivalent goals are never asserted, and that goals persist as long as, but no longer than, they remain relevant.

15:15 - 15:45 Automated Information Profiling with CLIPS.

Rodney Raines, U. S. Coast Guard.

Rapid analysis and dissemination of information is critical to the success of many computer centers. With advances in information technology, greater volumes of information pass through computer networks and into storage. Much of this information is extraneous, consuming valuable storage space and wasting CPU time. Without proper management, this volume of information can reduce the performance of the computer center and the end user.

15:45 - 16:15 Proposal for a CLIPS Software Library.

Ken Porter, Harris Corporation.

This paper is a proposal to create a software library for CLIPS, the C Language Integrated Production System expert system shell developed by NASA. Many innovative ideas for extending CLIPS were presented at the First CLIPS Users Conference, including useful user and database interfaces. CLIPS developers would benefit from a software library of re-usable code. The CLIPS Users Group should establish a software library -- this paper proposes a course of action to make that happen. Open discussion to revise this library concept is essential, as only a group effort is likely to succeed. At the end of the paper is a response form intended to solicit opinions and support from the CLIPS community.

<u>Group 6B:</u> **Gym**

Host: Chin-Liang Chang

14:45 - 15:15 Improving NAVFAC's Total Quality Management of Construction Drawings with CLIPS.

Albert Antelman, Naval Civil Engineering Laboratory.

This paper describes a diagnostic expert system to improve the quality of Naval Facilities Engineering Command (NAVFAC) construction drawings and specification. CLIPS and CAD layering standards are used in an expert system to check and coordinate construction drawings and specifications to eliminate errors and omissions.

15:15 - 15:45 **Validation of an Expert System Intended for Research in Distributed Artificial Intelligence.**
C. Grossner, J. Lyons and T. Radhakrishnan, Concordia University.

The expert system discussed in this paper is designed to function as a testbed for research on cooperating expert systems. Cooperating expert systems are members of an organization which dictates the manner in which the expert systems will interact when solving a problem. The Blackbox Expert described in this paper has been constructed using CLIPS, C++, and X widowing environment. CLIPS is embedded in a C++ program which provides objects that are used to maintain the state of the Blackbox puzzle. These objects are accessed by CLIPS rules through user-defined function calls. The performance of the Blackbox Expert is validated by experimentation. A group of people are asked to solve a set of test cases for the Blackbox puzzle. A metric has been devised which evaluates the "correctness" of a solution proposed for a test case of Blackbox. Using this metric and the solutions proposed by the humans, each person receives a rating for their ability to solve the Blackbox puzzle. The Blackbox Expert solves the same set of test cases and is assigned a rating for its ability. Then the rating obtained by the Blackbox Expert is compared with the ratings of the people, this establishing the skill level of our expert system.

15:45 - 16:15 **Testing Validation Tools on CLIPS-Based Expert Systems.**
Chin-Liang Chang, R. Stachowitz and J. Combs Lockheed Software Technology Center.

The Expert systems Validation Associate (EVA) is a validation system which was developed at the Lockheed Software Technology Center and Artificial Intelligence Center between 1986 and 1990. EVA is an integrated set of generic tools to validate any knowledge-based system written in any expert system shell such as CLIPS, ART, OPS5, KEE and others. Many validation tools have been built in the EVA system. In this paper, we describe the testing results of applying the EVA validation tools to the Manned Maneuvering Unit (MMU) Fault Diagnosis, Isolation, and Reconfiguration (FDIR) expert system, written in C Language Integrated Production System (CLIPS), obtained from the NASA Johnson Space Center.

16:30 - 17:30 BOARD RECEPTION (Cash Bar) Nassau Bay Hilton
It's Happy Hour at the Marina Via - with free hor d'ouvres!

WEDNESDAY, SEPTEMBER 24

08:45 - 10:15 SESSION 7 CONCURRENT GROUPS, A AND B

Group 7A: **Ballroom**
Host: Terry Feagin

08:45 - 09:15 **Integrating the IMKA Technology with CLIPS.**
David Scarola, Carnegie Group, Inc.

This presentation will share our experiences in evaluating the technical alternatives for integrating the IMKA frame-based knowledge representation system with CLIPS. The initiative for Managing Knowledge Assets (IMKA), consisting of Carnegie Group, Inc., Digital Equipment Corporation, Ford Motor Company, Texas Instruments Incorporated and U S WEST, In., was formed to foster the cooperative funding and development of a software technology that will meet each company's and their clients' needs for capturing and managing complex, corporate-wide knowledge. The IMKA Technology is a frame-based knowledge representation system for developing knowledge-based applications. Integrating the IMKA technology with CLIPS allows application knowledge to be encoded naturally using both frames and rules, and allows the knowledge stored in frames to be reasoned about using rules. Integrating a frame-based system with a RETE-based rule system is a challenging task because the approach to accessing data is very different in each system. We found three integration models that can be used to address the different data access methods of IMKA and CLIPS. This presentation provides an overview of the features of the IMKA technology, describes the challenges of integrating the IMKA technology with CLIPS, and discusses the three integration models and the circumstances under which each is appropriate.

09:15 - 09:45 **A CLIPS-Based Tool for Aircraft Pilot-Vehicle Interface Design.** Tom Fowler, CAL POLY, and Steven Rogers, Anacapa Sciences.

The Pilot-Vehicle Interface of modern aircraft is the cognitive, sensory, and psychomotor link between the pilot, the avionics modules, and all other systems on board the aircraft. To assist Pilot-Vehicle Interface designers a CLIPS-Based tool has been developed that allows design information to be stored in a table that can be modified by rules representing design knowledge Developed for the Apple Macintosh, the tool, allows users without any CLIPS programming experience to form simple rules using a point and click interface.

09:45 - 10:15 **On the Generation of Graphical Objects and Images From Within CLIPS Using XView.** Terry Feagin,University of Houston - Clear Lake.

A variety of features that support the generation and manipulation of graphical objects and images in CLIPS are described. These features provide the CLIPS programmer with the ability to develop an enhanced user interface. Windows and objects within the windows (such as buttons, sliders, text, etc.) can be created, hidden, redisplayed, given new values, properties, or positions, and manipulated in various other ways. Menus can be generated for any window or object and displayed when desired. Interaction with the user is primarily via mouse movements and mouse button selections made over windows, item, or menus. User input is recorded in the form of fact assertions. Limited animation of objects is also supported.

Group 7B: **Room 217**

Host: George Whitson

08:45 - 09:15 **Passive Acquisition of CLIPS Rules.** Vincent Kovarik, Software Productivity Solutions, Inc.

The automated acquisition of knowledge by machine has not lived up to expectations and knowledge engineering remains a human intensive task. Part of the reason for the lack of success is the difference in cognitive focus of the expert. The expert must shift his or her focus from the subject domain to that of the representation environment. In doing so, this cognitive shift introduces opportunity for errors and omissions. This paper presents work which observes the expert interact with a simulation of the domain. The system logs changes in the simulation objects and the expert's actions in response to those changes. This is followed by the application of inductive reasoning to move the domain-specific rules observed to general domain rules.

09:15 - 09:45 **YUCSA: A CLIPS Expert Database System to Monitor Academic Performance.** Anestis Toptsis, Frankie Ho, Milton Leindekar, Debra Low Foon and Mike Carbonaro, YORK University.

This paper present YUCSA (York University CLIPS Student Administrator), an expert database system implemented in CLIPS, for the monitoring of the academic performance of undergraduate students at York University. The expert system component of the system has been already implemented for two major Departments and it is under testing and enhancement for more Department. Also, more elaborate user interfaces are under development. We describe the design and implementation of the system and also discuss the problems encountered, as well as our immediate future plans. The system has excellent maintainability and it is very efficient (assessment of one student takes less than one minute.)

09:45 - 10:15 **A CLIPS Based Personal Computer Hardware Diagnostic System.** George Whitson, The University of Texas at Tyler.

Often the person designated to repair personal computers has little or no knowledge of how to repair a computer. This paper describes a simple expert system to aid these inexperienced repair people. The first component of the system leads the repair person through a number of simple system checks such as making sure all cables are tight ant that the dip switches are set correctly. The second component of the system assists the repair person in evaluating the error codes generated by the computer. In the final component the system applies a large knowledge base to attempt to identify the component (components) of the personal computer that is (are) malfunctioning. We have implemented and tested our design with a full system to diagnose problems for an IBM compatible system based on the 8088 chip. In our tests, the inexperienced repair people found the system very useful in diagnosing their hardware problems.

10:15 - 10:30 **Refreshment Break**

10:30 - 12:00 **SESSION 8 CONCURRENT GROUPS, A AND B**

Group 8A: **Ballroom**

Host: Craig Boyle

10:30 - 11:00 **PVEX - An Expert System for Producibility/Value Engineering.** Chun Lam and Warren Moseley, The University of Alabama in Huntsville.

This paper describes PVEX, an expert system that solves the problem of selection of the material and process in missile manufacturing. The producibility and value problem has been deeply investigated in he past years, and was written in dBase III and PROLOG before. This project represents a new approach to it in the the solution is achieved by introducing hypothetical reasoning, heuristic criteria integrated with a simple hypertext system and shell programming. PVEX combines KMS with Unix scripts which graphically depicts decision trees. The decision trees convey high level qualitative problem solving knowledge to users, and a stand-alone help facility and technical documentation is available through KMS. The system the authors developed is considerably less development in development costs than any other comparable expert system.

11:00 - 11:30 **Rule Groupings in Expert Systems Using Nearest Neighbor Decision Rules, and Convex Hulls.** Stergios Anastasiadis, McGill University.

Expert System shells lack in many areas of software engineering. Large rule-based systems are not semantically comprehensible, difficult to debug, and impossible to modify or validate. Partitioning a set of CLIPS rules, into groups of rules which reflect the underlying semantic subdomains of the problem, will address adequately the concerns stated above. In this paper we introduce techniques to structure a CLIPS rule-base into groups of rules that inherently have common semantic information. The concepts involved are imported for the fields of A.I., Pattern Recognition, and Statistical inference. Techniques focus on the areas of feature selection, classification, and a criteria of how "good" the classification technique is, based on Bayesian Decision Theory. We discuss a variety of distance metrics for measuring the "closeness" of CLIPS rules and describe various Nearest Neighbor classification algorithms based on the above metrics.

11:30 - 12:00 **Debugging Expert Systems using a Dynamically Created Hypertext Network.** Craig Boyle and John Schuette, Texas A&M University.

The labor-intensive nature of expert system writing and debugging has motivated this study. Our hypothesis is that a hypertext based debugging tool is easier and faster than one traditional tool, the graphical execution trace. HESDE (Hypertext Expert System Debugging Environment) uses Hypertext nodes and links to represent the objects and their relationships created during the execution of a rule based expert system. HESDE operates transparently on top of the CLIPS rule-base system environment and is used during the knowledge-base debugging process. During the execution process HESDE builds an execution trace. Use of facts, rules and their values are automatically stored in a Hypertext network for each execution cycle. After the execution process the knowledge engineer may access the Hypertext network and browse the network created. The network may be viewed in terms of rules, facts and values. An experiment was conducted to compare HESDE with a graphical debugging environment. Subjects were given representative tasks. For speed and accuracy, in eight of the eleven tasks given to subjects HESDE was significantly better.

Group 8B: **Room 217**

Host: Scott Huse

10:30 - 11:00 **Implementing a Frame-Based Representation with CLIPS/COOL.** Leonard Myers and James Snyder, CAL POLY

The purpose of this paper is to describe and evaluate an implementation of frames in COOL. The test case is a frame-based semantic network previously implemented in CLIPS Version 4.3 as part of the Intelligent Computer-Aided Design System (ICADS) and reported in the first CLIPS conference.

11:00 - 11:30 **Application of a Rule-Based knowledge System Using CLIPS for the Taxonomy of Selected *Opuntia* Species.** Bart Heymans, Joel Onema and Joseph Kuti,Texas A & I University

A rule-based knowledge system was developed in CLIPS (C-Language Integrated Production System) for identifying *Opuntia* species in the family Cactaceae, which contains approximately 1,500 different species. This botanist expert tool system is capable of identifying selected *Opuntia* plants from the family level down to the species level when given some basic characteristics of the plants. Many *Opuntia* species are cultivate as ornamental plants and some are significant as food crops. *Opuntia* plants are becoming of increasing importance because of their nutrition and human health potential especially in the treatment of diabetes mellitus. The expert tool system described in this paper can be extremely useful in an unequivocal identification of many useful *Opuntia* species.

11:30 - 12:00 **The Nutrition Advisor Expert System.** Scott Shyne and Scott Huse, Rome Laboratory.

The Nutrition Advisor Expert System (NAES) is an expert system written in the C Language Integrated Production System (CLOPS)/ NAES provides expert knowledge and guidance into the complex world of nutrition management by capturing the knowledge of an expert and placing it at the user's fingertips. Specifically, NAES enables the user to: (1) obtain precise nutrition information for food items, (2) perform nutritional analysis of meal(s), flagging deficiencies based upon the United States Recommended Daily Allowances, (3) predict possible ailments based upon observed nutritional deficiency trends, (4) obtain a top-ten listing of food items for a given nutrient, and (5) conveniently upgrade the database. An explanation facility for the ailment prediction feature is also provided to document the reasoning process.

12:05 - 13:05 LUNCH **Ballroom**

13:10 - 14:10 BOARD MEETING (open to all) **Room 217**

The primary purpose of this meeting to is provide the new Board members with an opportunity to present their initial ideas for the CLIPS Users Group and to formally hear from the CLIPS Users Group members.

- -

INFORMATION FORM

If you need to change information for the mailing of the proceedings from that on your original registration, please fill out the following and return it to the registration table.

Second Annual CLIPS Users Group Conference

September 23-25, 1991 Gilruth Center, Johnson Space Center, Houston, TX

(please print)

Name ______________________ Title ____________

Address ______________________ Phone ____________

City ______________________ State ___ Zip ________

SESSION 1

N92-16569

Rule Groupings: An Approach towards Verification of Expert Systems

Mala Mehrotra
Vigyan Inc.
30, Research Drive
Hampton, Va 23666.
mm@air12.larc.nasa.gov

Knowledge-based expert systems are playing an increasingly important role in NASA space and aircraft systems. However, many of NASA's software applications are life- or mission-critical and knowledge-based systems do not lend themselves to the traditional verification and validation (V&V) techniques for highly reliable software. Rule-based systems lack the control abstractions found in procedural languages. Hence, it is difficult to verify or maintain such systems. Our goal is to automatically structure a rule-based system into a set of rule-groups having a well-defined interface to other rule-groups. Once a rule base is decomposed into such "firewalled" units, studying the interactions between rules would become more tractable. Verification-aid tools can then be developed to test the behavior of each such rule-group. Furthermore, the interactions between rule-groups can be studied in a manner similar to integration testing. Such efforts will go a long way towards increasing our confidence in the expert-system software.

There are two main reasons why expert systems defy verification and validation efforts. First, rapid prototyping and iterative development form key features of any expert system development activity. This has led to the development of ad-hoc techniques for expert-system design without any software engineering guidelines. Second, due to the data-driven nature of expert systems, as the number of rules of an expert system increase, the number of possible interactions between the rules increases exponentially. The complexity of each pattern in a rule compounds the problem of testing even further. This makes exhaustive, or even systematic, testing of large knowledge bases infeasible. As a result, large expert systems tend to be incomprehensible, difficult to debug or modify, and almost impossible to verify or validate.

This situation is not wholly unlike that which faced traditional software development before the introduction of structured software engineering.Conventional software yields more easily to verification efforts because control is explicitly represented as procedures which can be structured to encapsulate run-time abstractions. Modules can be designed in con-

MISSING
PRECEDING PAGE ~~BLANK~~ NOT FILMED

ventional software, each consisting of a manageable unit with a well-defined interface. Furthermore, procedures can be grouped into packages or objects which share an internal data structure. These units can then be subjected to unit/integration testing techniques. In expert systems, rules play a role analogous to paths through procedures. However, each rule in an expert system is data-driven, since the presence or absence of data controls the flow of execution. Hence, V&V techniques for expert systems have to view interactions between all pairs of rules. For large expert systems, this is quite difficult and can be prohibitively expensive.

Our research efforts address the feasibility of automating the identification of rule groups, in order to decompose the rule base into a number of meaningful units. Each such group can then be viewed as a procedure. Identification of the intra-group and inter-group items for a group of rules would be analogous to defining local variables and parameters for procedures in conventional software. A verification-aid tool could then test the behavior of each such unit under all possible values of inputs [1].

The grouping of rule bases can play an important role in verification and validation of flight-critical systems at NASA. In such systems one needs to identify critical regions, assert various criticality levels [2] for them, and test such regions both analytically and exhaustively. If one is able to isolate the group of rules that deal with the critical features of the problem domain, certain safety properties of the system can be verified. Knowledge of the function of a group of rules would allow us to choose the distribution of inputs in such a way that typical situations where functioning of the system is critical could be studied in isolation [7]. Moreover, if support existed for specifying what rules should not get fired under certain circumstances, backward flow analysis techniques [3] could be used to locate critical paths. An additional advantage of modularization would be the identification of modules and data items that are necessary in a degraded (fail-soft) processing mode. Validating such modules is clearly critical to confidence in the reliability of the software.

Validation of conventional software systems relies on systematic testing, since exhaustive testing of such systems is generally infeasible. Such systematic testing is based on abstractions such as procedures, functions and other control structures. Along with such explicit abstractions, other testing techniques, such as data flow and path testing, take into account implicit dependencies between different parts of the program. Rule-based systems lack the control abstractions found in procedural languages. Although the non-procedural nature of rule-based systems is sometimes cited as an advantage, the subsequent lack of control abstractions makes it difficult to verify and maintain these systems.

Towards this goal, we would like to structure a rule base into a set of groups consisting of related rules. We have taken a pattern-matching approach for this grouping of rules. In this approach, the commonality of items in the rules determines the distance between them. Our rule grouping process consists of three stages. First, the distance between each pair of rules is computed and stored in a distance matrix. In the second stage, the computed distances are modified so that all distances satisfy the triangle inequality. That is, we replace the distance between two rules by a shorter distance, if there exists an intermediate rule through which a shorter path can be created. This procedure thus extracts transitive dependencies between rules. Finally, we apply a clustering algorithm to form our groups.

In [4, 5, 6], we have studied three types of distance metrics and two approaches to clustering. The first one is an automatic clustering algorithm based on graph-partitioning approaches. The second requires the user to designate certain rules as "primary rules" or "seed rules" around which the clustering algorithm will form groups. These primary rules typically reflect key concepts from the domain; thus, the resulting clusters correspond closely to the user's conceptual model of the problem domain. Two independent evaluation criteria were developed to measure the effectiveness of the grouping strategies.

Based on our results so far, we believe a comprehensive set of distance metrics can be designed which would be effective in grouping all types of rule bases. By applying our grouping tool to more complicated and larger rule bases, we hope to obtain more insight into parameters that play a critical role in grouping. A significant outcome from our study will be the formulation of software engineering guidelines for the design of rules, which would promote the grouping process without sacrificing programming flexibility. In fact these guidelines may actually aid in programming by providing guidance and discipline, similar to the aid that structured programming gives in maintaining intellectual control of traditional software. This is also analogous to work in language design which is driven by the ease of proof rules for verification. A good style of rule-based programming will make the underlying relationships between key concepts more transparent and easier to understand and verify.

In future, we intend to study the interplay of distance metrics, clustering criteria, objective functions to be optimized and software engineering guidelines on grouping. We would also like to formulate rigorous evaluation criteria to judge the quality of groups formed. A handle on the definition of what constitutes a "good grouping" will feed back into the criteria to be used for formation of good groups. A set of software engineering requirements could then be laid out that are necessary for producing useful partitions in a rule base from the point of view of testing them. The "quality" of grouping is very much dependent on the motivation behind grouping. For maintainability and ease of comprehension of the rule base one would want the groupings to "match" human expectations. However, for testing and verification, this aspect is less important. In fact, one could argue that "unnatural" groupings would force the software engineer to see the rule base in a new light which could give insight into its behavior - *particularly* its unexpected behavior.

Our research plans are designed to give us insight into the factors that lead to meaningful rule groupings after which we intend to develop automatic verification aid tools for unit/integration testing of well-partitioned rule bases. Each partition or group obtained through our grouping process can be viewed as a procedure. Our software tool would be extended to allow identification of the common intra-group and inter-group items for a group of rules, which would be analogous to local variables and parameters for procedures in conventional software. Such groups can be formed into software modules with "firewalls," having well-defined inputs and outputs. Formation of such modules can be seen as the first step towards managing their complexity for verification and validation. The interface of the modules can then help in the automatic generation of test suites required for verification and validation. Validating such modules is clearly critical to confidence in the reliability of the knowledge-based system software.

References

[1] C. Culbert and R. T. Savely. Expert system verification and validation. In *Proceedings, Validation and Testing Knowledge-Based Systems Workshop*, August 1988.

[2] S. C. Johnson. Validation of highly reliable, real-time knowledge-based systems. In *SOAR 88 Workshop on Automation and Robotics*, July 1988.

[3] N. G. Leveson. Safety-critical software development. In T. Anderson, editor, *Safe & Secure Computing Systems*, chapter 9, pages 155–162. Blackwell Scientific Publications, 1989.

[4] M. Mehrotra. Rule groupings: A software engineering approach towards verification of expert systems. Technical Report NASA CR-4372, NASA Langley Research Center, Hampton, VA., May 1991.

[5] M. Mehrotra and S. C. Johnson. Importance of rule groupings in verification of expert systems. In *Notes for the AAAI-90 Workshop on Verification, Validation and Testing of Knowledge-Based Systems*, July 1990.

[6] M. Mehrotra and S. C. Johnson. Rule groupings in expert systems. In *Proceedings, First CLIPS Users Group Conference*, Aug 1990.

[7] D. L. Parnas, J. van Schouwen, and S. Po Kwan. Evaluation of safety-critical software. *Communications of the ACM*, 33(6):636–648, June 1990.

N92-16570

ENHANCED USE OF CLIPS AT THE LOS ALAMOS NATIONAL LABORATORY

by

K. H. Duerre
W. J. Parkinson
Group MEE-9
and
J. J. Osowski
Group MEE-4
Los Alamos National Laboratory
Los Alamos,NM 87545

ABSTRACT

Early efforts in producing Expert Systems for engineering applications used a limited subset of CLIPS features. In this paper we discuss the implementation details of previous Expert Systems and of the current Expert System, which is used for training operators in the control of the Isotope Separation System.

INTRODUCTION

Three CLIPS-based Expert Systems[1] were developed to assist in solving engineering problems at the Los Alamos National Laboratory (LANL). These systems were justified by The Laboratory's need to save corporate knowledge that might be lost by personnel attrition and by the requirement to transfer the technology to industry. Two of the systems were advisors designed to aid in the selection of the "best" equation of state models for process design[2], and to screen enhanced oil recovery methods[3]. These advisors were not used for control nor did they include explanation facilities. The third Expert System[4] to assist in the production of silicon carbide whiskers is similar in function to that of our current system. As can be seen in Fig. 1, whisker growth is a complex process. Although whisker growth is difficult, if not impossible, to model mathematically, "excellent" whiskers can be grown by a human expert.

The LANL Materials Science and Technology Group MST-3 is responsible for research and development of systems and equipment for the handling and processing tritium, the heavy radioactive isotope of hydrogen. The group runs the Tritium Systems Test Assembly (TSTA) in which equipment and processes are tested for use with a fusion reactor fuel cycle.

The Isotope Separation System (ISS) shown schematically in Fig. 2, is one of the systems housed in the TSTA. It utilizes cryogenic distillation columns to separate hydrogen isotopes (protium, deuterium and tritium) into pure component streams. Controllers, which are conventional and of 10 year-old technology, require frequent manual adjustments of setpoints to accommodate the non-steady state nature of the ISS. A large amount of data must be monitored by operating personnel. System interactions are complex, making optimum operation difficult to attain.

The operation and control of the system is an art that is understood by only a few technical personnel who have long experience with the system. Even these experienced personnel have problems assimilating and reconciling variables in a timely manner. It was decided that an Expert System should be developed to collect and document the knowledge of the few experts in the ISS operation. This system could then be used to assist the technicians who normally operate the system and to train new personnel.

The success of earlier Expert Systems was largely instrumental in considering the use of an expert advisor for this system. The Whisker Production Advisor is currently being transferred to industry.

DESIGN CONSIDERATIONS

Whisker Production Advisor

We felt that any usable expert system should be executable with an inexpensive shell and an easily available computer. The PC version of CLIPS was chosen for this task. The search trees associated with production run setup and run control had five levels, so they required a good search algorithm in order to obtain a reasonable response. CLIPS's RETTE algorithm was adequate for the task.

ISS Advisor

Since the resident experts were approaching retirement age, the knowledge base constructed should be easy to maintain and should be as complete as possible. The operators had considerable experience in seeing the data appear on graphic devices and, although an easy interface was not top priority, we soon learned that it made acceptance of this tool much easier. Available hardware was limited to the IBM AT class of personal computers. The size of program developed should fit in available memory (<640 kbytes). Finally, the program should be easy to maintain.

THE PROGRAMS

The Whisker Advisor

Figure 3 shows the organization of the whisker advisor. The two parts, run setup and control system, were developed separately, and each was embedded in the CLIPS shell. The operator interacts with the system through the computer keyboard and screen. Figure 4 illustrates a partial sample dialog for a hypothetical run.

The ISS Advisor

The structure and information flow of the ISS advisor is shown in Fig. 5. The main program is written in ANSI C. It provides the user interface, whether the interface is textual or graphics, obtains the initial conditions, and asserts the facts obtained in the main program, to the embedded CLIPS rules. The main program retrieves the advice generated by CLIPS and displays it to the operator.

The program begins by giving the user a brief description of the program's purpose. The CLIPS environment is then initialized and rules from the file ISS.CLP are loaded into the system. The program now has the "brain" necessary to make decisions based on data to be input later. The user then enters the main loop of the program.

The main loop initially gives the user a list of possible options pertaining to the type of problem to be solved. Depending on the users answer, the program searches for data pertaining to the specified problem. The user is also permitted to input fresh data via Option 6. The program loads in data and the user is permitted to make adjustments. The user is also permitted to input "no data available" for any combination of data cells. Figure 6, shows the data for a default scenario, the cursor is positioned at the data cell to be changed and new data can replace that shown. Compare this with the elicitation scheme shown in Fig. 4.

Once data is available, it is asserted to the rule base (brain) and the appropriate action is determined based on the given information. Figure 7 illustrates the types of rules created from the asserted information (could be specified or measured). If blanks are input into the data, the value "99999" is asserted to the rule base. This number is used to numerically mark (becomes a flag) to indicate "no data available." Any rule finding "99999" as data **will not fire.** As rules fire, a queue is filled containing text or display information that is sent back to the main C driver program from the rulebase, placed on a stack, and sorted. The text statements or display information is then shown as suggestions to the user. This completes a pass through the main loop of the program; however, the user is permitted to cycle through the loop as many times as requested with new and/or reused data.

Three versions of the ISS advisor exist: (1) ISS.EXE (a color text version); (2) ISSL.EXE (a black and white version for lap-top personal computers); and (3) ISSIS.EXE (a graphics version which has proven to be the most popular of the three versions).

Figure 8 is a sample screen produced by the advisor. The amount of information presented is much greater and more compact than simple textual dialog.

CONCLUSIONS

A friendlier interface was obtained by embedding CLIPS inside a main C program than for our earlier versions of Expert Systems. Preliminary versions of the ISS advisor have shown that the memory limitation has been avoided by the careful use of files for the verbose advice indexed by the "fired" rules and also by placing the initial conditions for various scenarios in data files. The use of templates has helped in the documentation of the rules and has generally improved the readability of the code.

It would have been helpful if rounded floating point data were available to be passed back in string form to the main program. CLIPS version 5.0 was received too late for use in this project.

Although testing the advisor against the actual operation of the Isotope Separation System has been limited, the use of graphics has greatly enhanced the acceptance of the advisor.

REFERENCES

1. W. J. Parkinson, G. F. Luger, and R. E. Bretz,"Three CLIPS-Based Expert Systems for Solving Engineering Problems," Proceedings of The First Annual CLIPS User's Conference, Houston, Texas, Vol. 1, pp. 3-17, August 13-15, 1990.

2. W. J. Parkinson, G. F. Luger, and R. E. Bretz,"Using PC-Based Shells to Write an Expert Assistant for Use with the ASPEN Computer Code," Paper presented at the AIChE Annual Meeting, Session on Applications of Artificial Intelligence in Chemical Engineering, April 2-6, 1989, Houston, Texas.

3. W. J. Parkinson, G. F. Luger, R. E. Bretz, and, J. J. Osowski,"An Expert System for Screening Enhanced Oil Recovery Methods," Paper presented at The 1990 Summer National Meeting of The American Institute of Chemical Engineers, San Diego, California, August 19-22, 1990.

4. W. J. Parkinson, P. D. Shalek, E. J. Peterson, and G. F. Luger, "Designing an Expert System for the Production of Silicon Carbide Whiskers," Paper presented at the TMS Annual Meeting, Symposium--Expert System Applications in Materials Processing & Manufacturing, February 19-22, 1990, Anaheim, California.

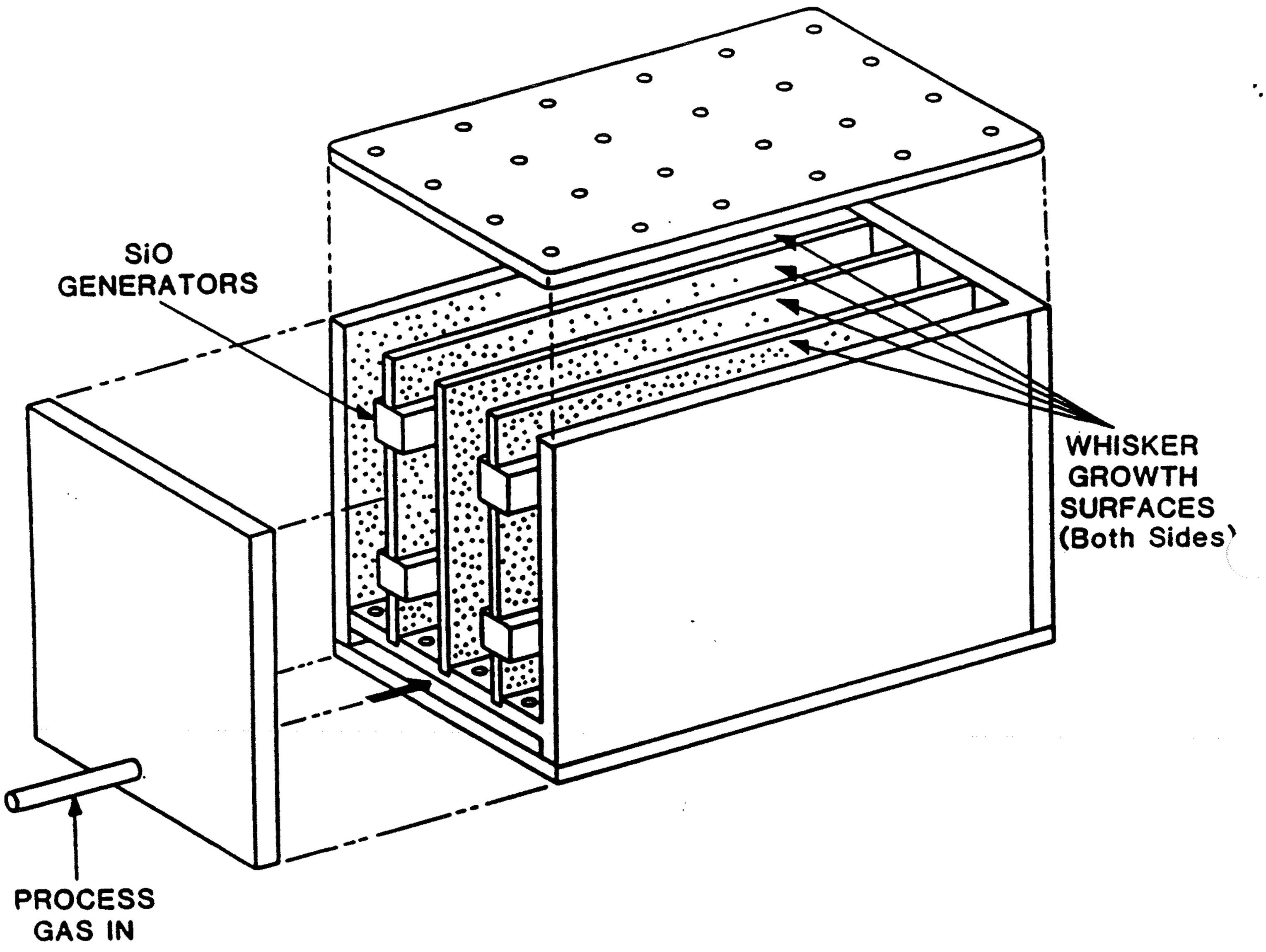


Figure 1. Los Alamos Silicon Carbide
Whisker Production Reactor.

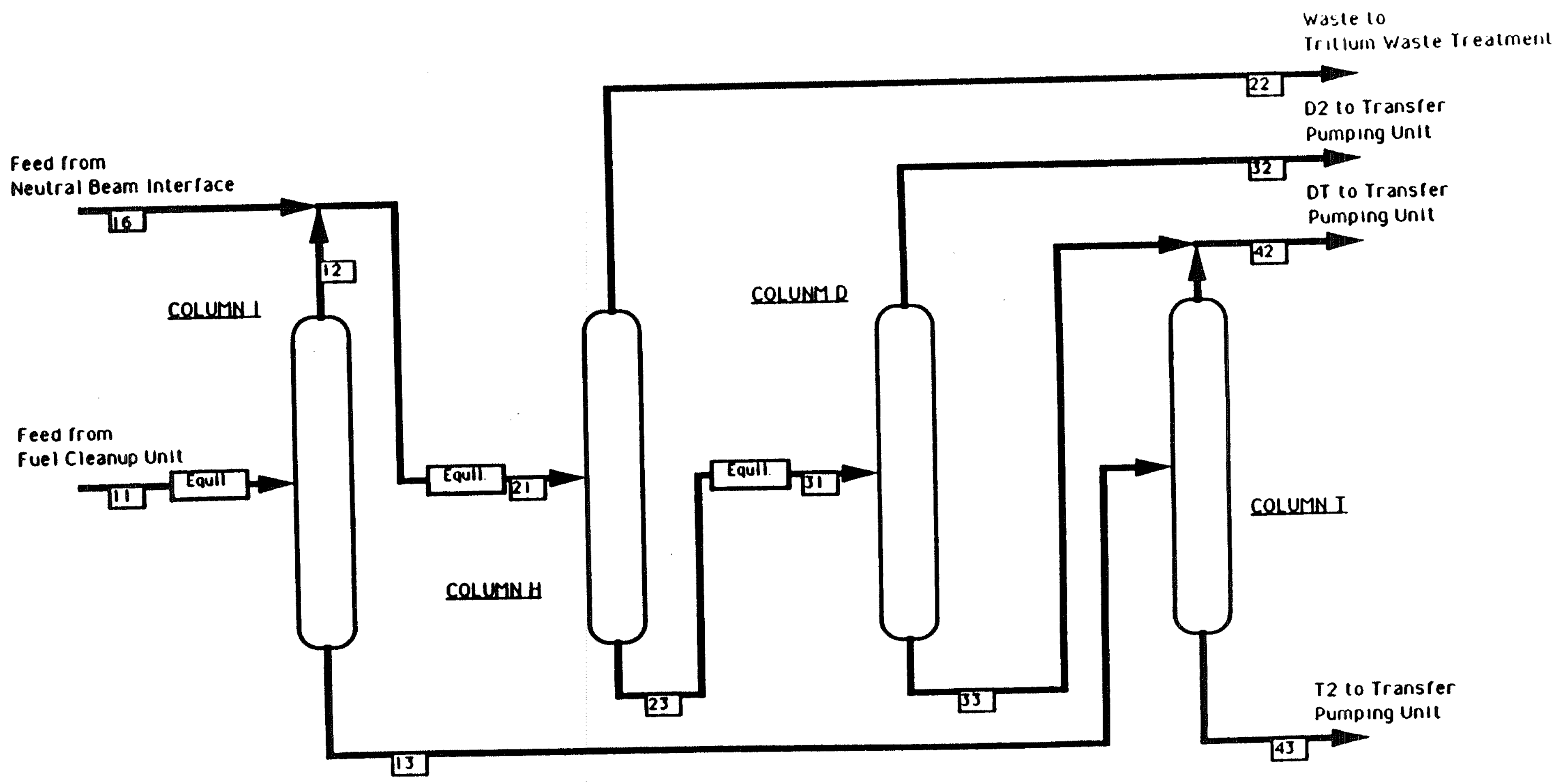


Figure 2. Isotope Separation System Schematic.

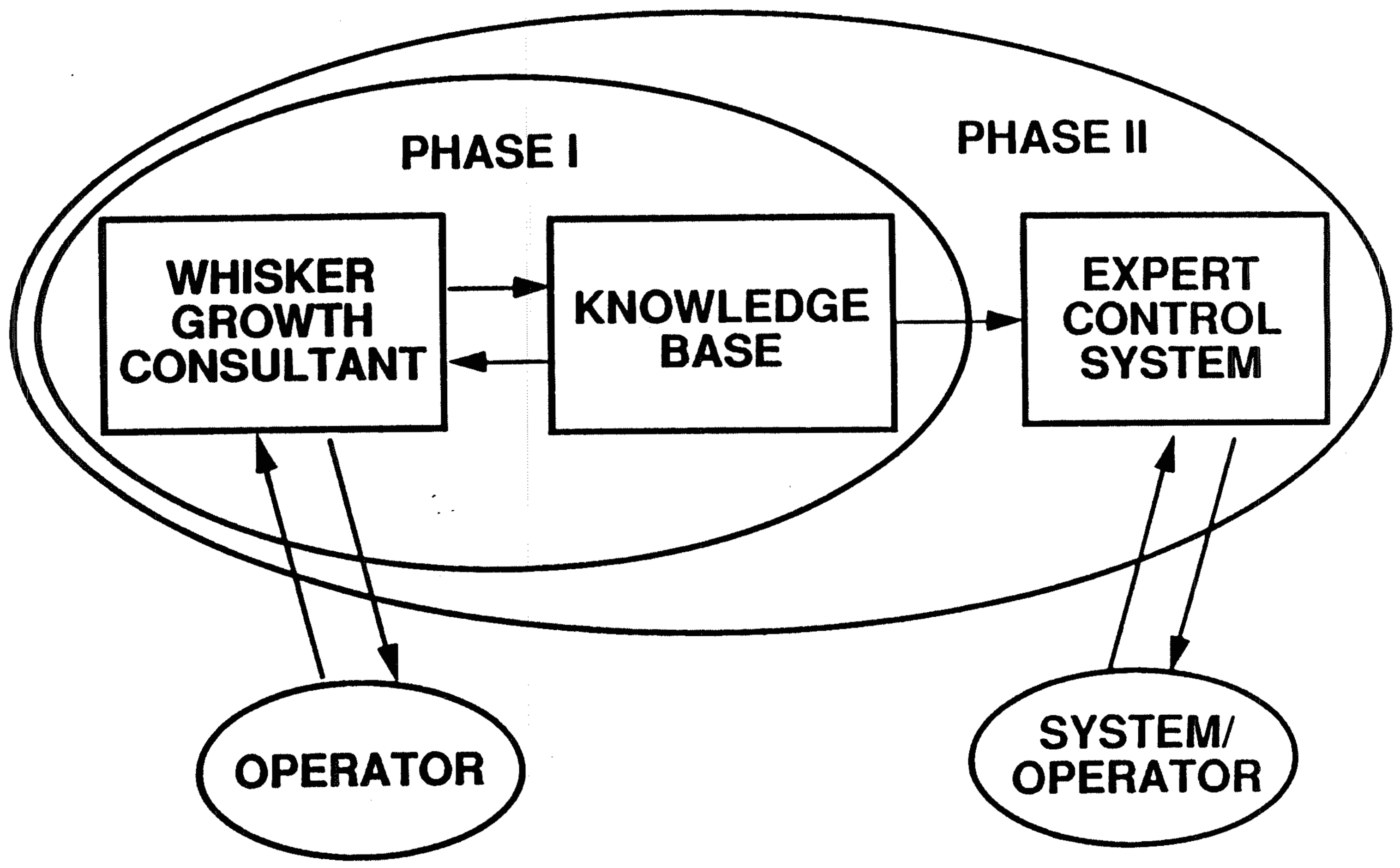


Figure 3. Whiskers Expert System.

```
What is the desired average whisker length ?
(in inches 0. to 3.5)

3.0

We recommend reactor type B,  which will you use ?
(A or B)

B

What is the desired average whisker diameter  ?
(in microns,  0 to 12)

10

We recommend the manganese based catalyst,  which
one will you choose  ?  (manganese or iron)

manganese

.

.
.
.
We recommend that you vary the CO concentration according to
time-concentration profile A.  What profile will you use  ?

(A, B, C, or D)

A
```

Figure 4. Sample Dialog for The Whiskers Advisor.

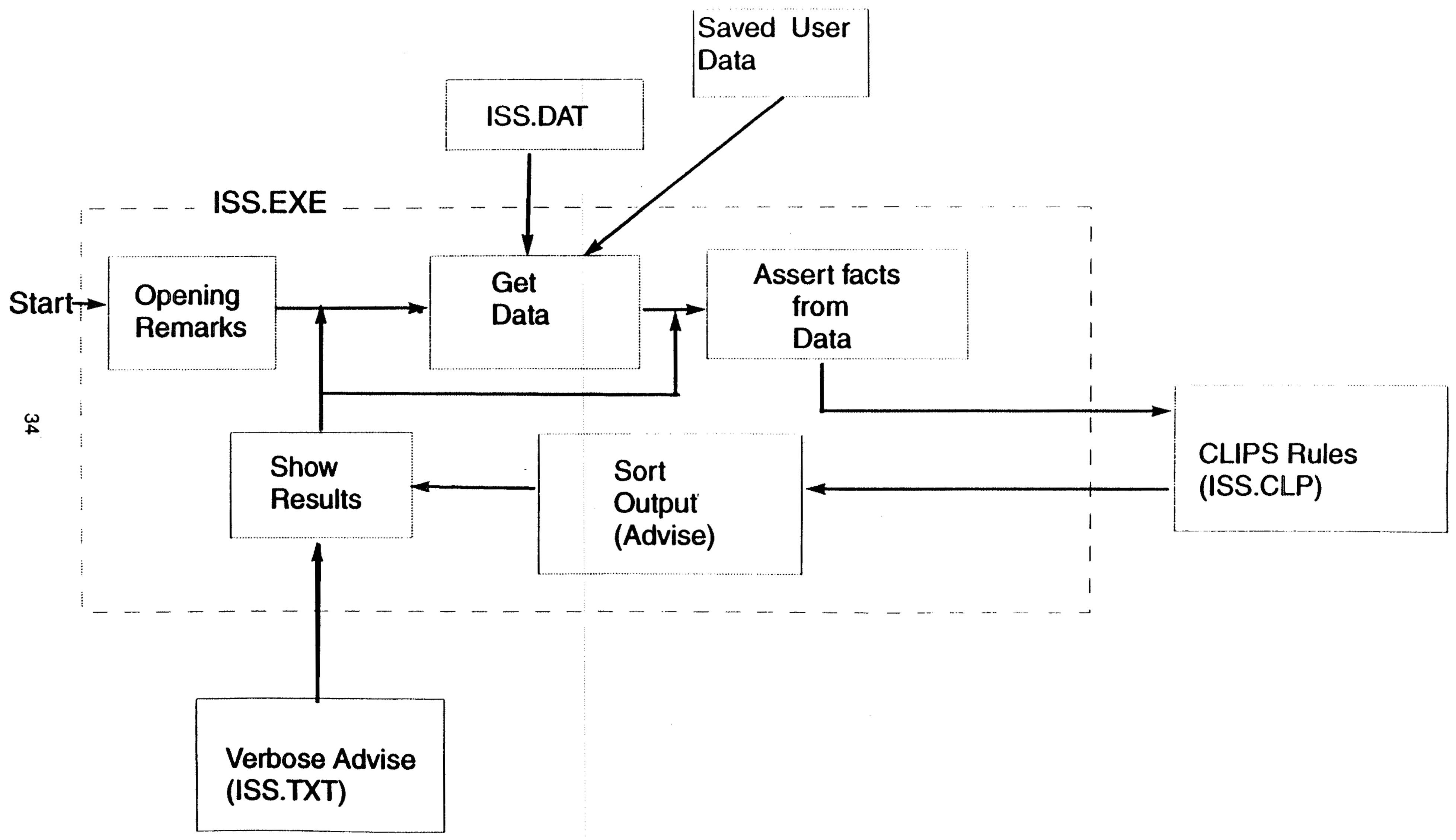
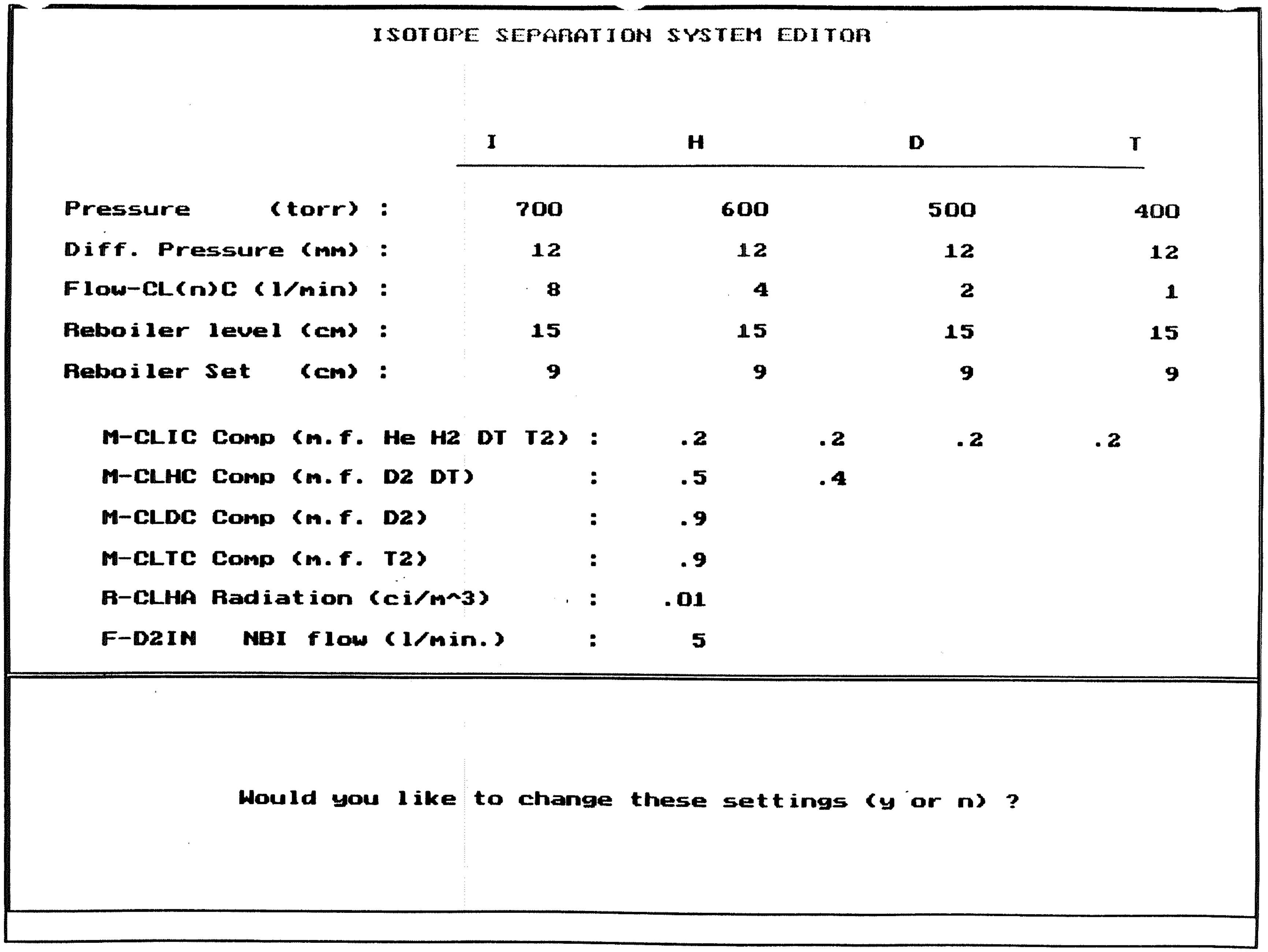


Figure 5. ISS Advisor Structure.



```
                    ISOTOPE SEPARATION SYSTEM EDITOR


                                 I          H          D          T
                            ____________________________________________

Pressure       (torr) :         700        600        500        400
Diff. Pressure (mm) :           12         12         12         12
Flow-CL(n)C (l/min) :            8          4          2          1
Reboiler level (cm) :           15         15         15         15
Reboiler Set    (cm) :           9          9          9          9

  M-CLIC Comp (m.f. He H2 DT T2) :    .2        .2        .2        .2
  M-CLHC Comp (m.f. D2 DT)       :    .5        .4
  M-CLDC Comp (m.f. D2)          :    .9
  M-CLTC Comp (m.f. T2)          :    .9
  R-CLHA Radiation (ci/m^3)      :   .01
  F-D2IN   NBI flow (l/min.)     :     5


     Would you like to change these settings (y or n) ?
```

Figure 6. ISS Input Editor Display Screen.

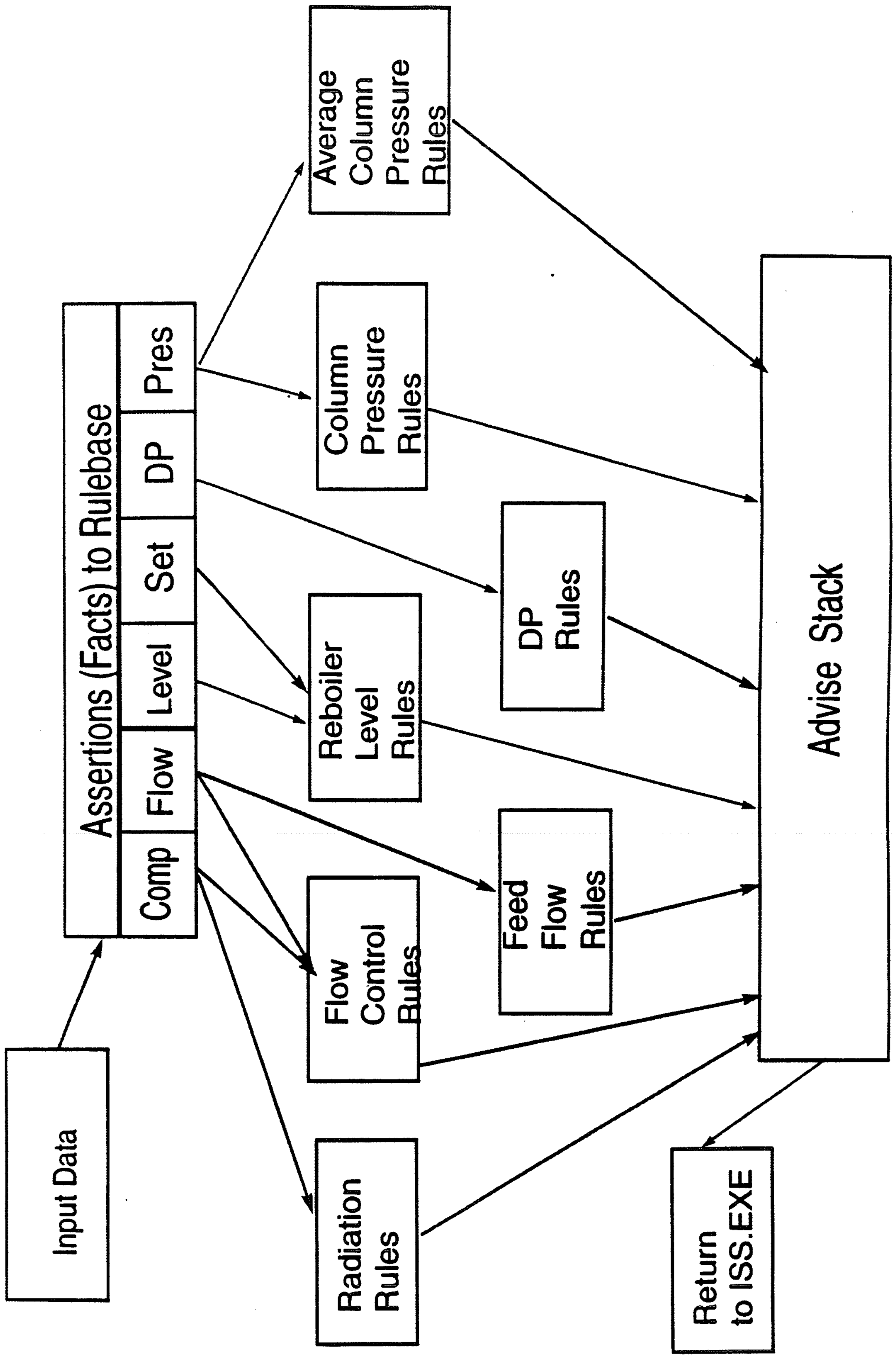
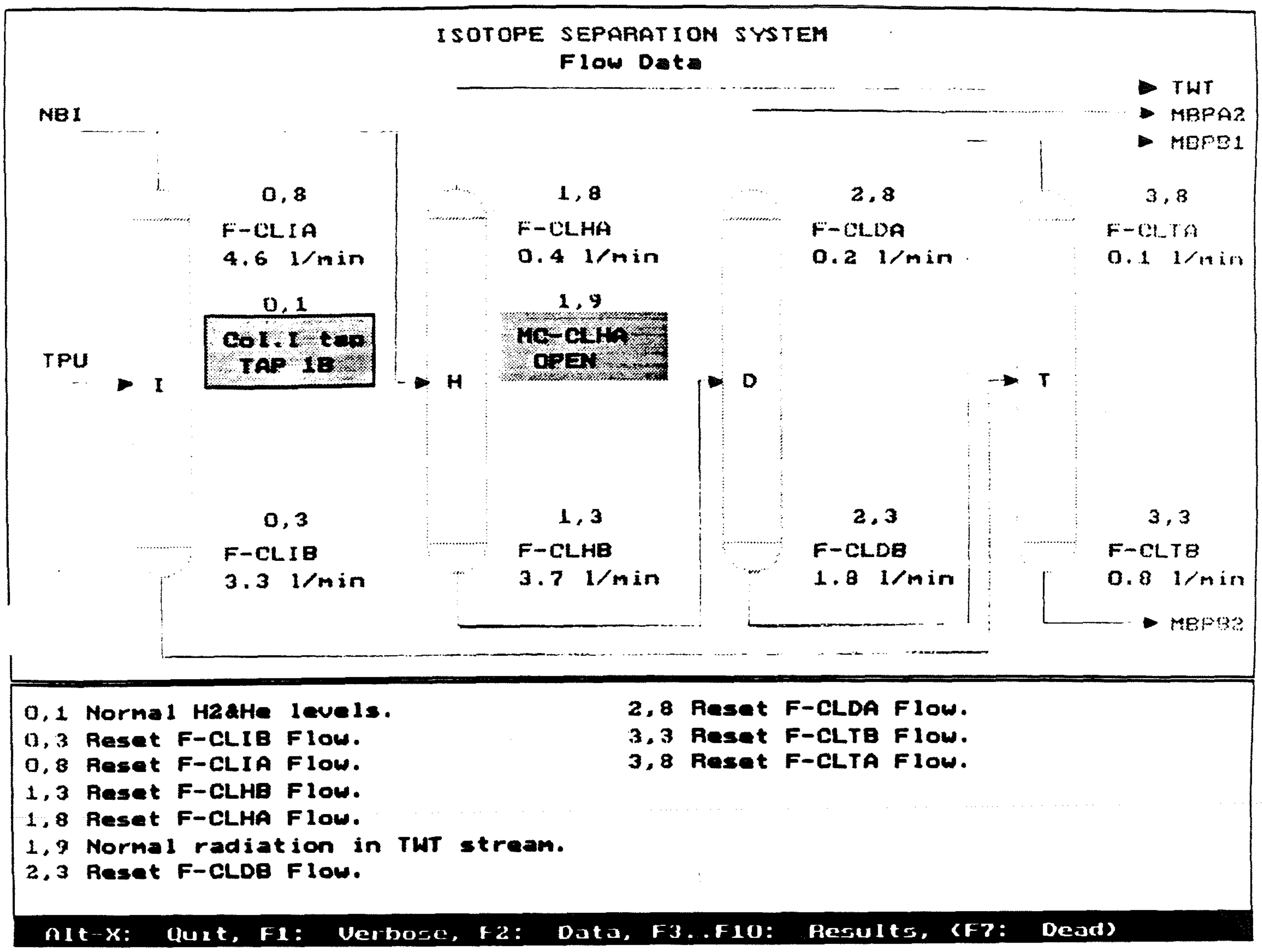


Figure 7. ISS Rulebase.



```
0,1 Normal H2&He levels.          2,8 Reset F-CLDA Flow.
0,3 Reset F-CLIB Flow.            3,3 Reset F-CLTB Flow.
0,8 Reset F-CLIA Flow.            3,8 Reset F-CLTA Flow.
1,3 Reset F-CLHB Flow.
1,8 Reset F-CLHA Flow.
1,9 Normal radiation in TWT stream.
2,3 Reset F-CLDB Flow.
```

Figure 8. ISS Sample Output Graphics Display.

ORIGINAL PAGE IS
OF POOR QUALITY

SESSION 2 A

N92-16571

USING A CLIPS EXPERT SYSTEM TO AUTOMATICALLY MANAGE TCP/IP NETWORKS AND THEIR COMPONENTS

Ben M. Faul

TRW Systems Engineering & Development Division
Carson, California

Abstract. This paper describes an expert system that can directly manage networks components on a TCP/IP network. Previous expert systems for managing networks have focused on managing network faults after they occur. However this proactive expert system can monitor and control network components in near real time. The ability to manage directly network elements from CLIPS is accomplished by the integration of the Simple Network Management Protocol (SNMP) and an Abstract Syntax Notation (ASN) parser into the CLIPS artificial intelligence language.

INTRODUCTION

Networking is one of the fastest growing segments of the computer market. Networks can be as simple as several PCs on a LAN to a corporate-wide area network composed of hundreds of machines to a global network comprised of hundreds of local area networks and hundreds of thousands of machines.

The emergence of network-based applications and even operating systems demands the network components operate as efficiently and effectively as possible.

Managing a network can be an arduous task, as there are numerous components that comprise the network, originationg from many vendors. In addition, the components are usually dispersed over a large geographic area. But, even if the network components were co-located, most of the network devices don't even have an operator's console.

All of these factors add up to a nightmare when things go wrong in the network. Traditional system operation concepts deal with problems as they arise. In a network, just locating a downed component can be a major task. All the while, applications and users are idle while technicians scour the campus looking for the problem. Several CLIPS applications have been described previously that aid in the isolation and diagnosis of problems [Leigh A.] by using a question and answer session with a human.

The next logical step in managing a network, is to look for, and solve, problems under expert system control. This is a natural application for an expert system like CLIPS. However, to utilize CLIPS as a solution, the expert system shell must incorporate several new features it does not currently have.

MISSING
PRECEDING PAGE ~~BLANK~~ NOT FILMED

NEW CLIPS FEATURES TO FACILITATE NETWORK MANAGEMENT

The US Government and commercial vendors recognized the need for developing a vendor-independent mechanism for managing network components, largely because of the network management chaos that erupted <u>after</u> networking became so prevalent.

Typical automated network management systems rely on a specific vendor's diagnostic hardware. One has even been written in CLIPS. [Hansen & Flores]. However, vendor-dependent network management solutions have only limited application in a network comprised of elements from different vendors.

To answer this need for vendor-independent network management the US Government's Network Working Group developed the Simple Network Management Protocol or, SNMP [Case, Fedor, Schoffstall, & Davin].

By integrating SNMP into the CLIPS language, expert systems can then be built that can take direct control of network elements; thus obviating the need for most human interaction.

SNMP Architectural Model

Implied in the SNMP architecture is a collection of network management stations and network elements. The network management station executes the applications that monitor and control network elements. Network elements are devices on the network such as hosts, routers, gateways, terminal concentrators, PCs, etc. that communicate on the network. The SNMP is used to communicate the management information to the network elements. The CLIPS program will be the manager for the network. The various hardware components on the network will be the network elements that CLIPS will manage.

The first goal of the SNMP integrated into CLIPS is to explicitly minimize the number and complexity of functions used by the manager program. This will result in the reduction of new language constructs in CLIPS; hence maintaining the portability and integrity of the language.

Another goal of the SNMP/CLIPS integration is to provide a paradigm for monitor and control that can accommodate unanticipated aspects of network management. As time and network products progress, the CLIPS manager will be extendible at the expert system shell level. This will tend to eliminate further extensions to the language itself.

The third and most important goal is that the resultant system will be independent of the architecture and mechanisms of the particular network elements. Achievement of this goal allows the CLIPS network manager to control network elements from any vendor.

Representation of Management Information

The information communicated using SNMP is represented using the ASN.1 language [ISO Standard 8824]. Use of the ASN.1 language internal to SNMP is key to its machine independence and eventual conformance with GOSIP mandates.

The information communicated using ASN.1 is called the management information base (MIB). There is a standard MIB, that all the conforming network products recognize.

An example ASN.1 variable in the MIB is represented in Table 1.

```
    sysUpTime OBJECT-TYPE
        SYNTAX TimeTicks
        ACCESS read-only
        STATUS mandatory
        ::= { system 2 }
```

Table 1. An ASN.1 Definition

The example in Table 1 will be referenced in the following paragraphs to illustrate CLIPS network management via the SNMP.

Protocol Operations

The SNMP functions integrated into CLIPS operate as inspections or modifications of variables that correspond to entries in the MIB. The manager specifies the MIB variable to view or alter, and the managee (also called "agent") does the appropriate get variable or set variable action. Notice in the above example that the variable is read-only. Thus the manager may view, but may not modify this variable.

Using the MIB, the variables become accessible in a machine-independent form. The CLIPS manager does not care what the network element's internal representation is of the variable or how it is derived and maintained.

Primarily, the CLIPS manager works by polling the network element agents for the appropriate information.

There are no imperative commands in the protocol. The manager merely sets a MIB variable to some value. The network element agent then decides what to do with the value. The example given in Request for Comments (RFC) 1157 [Case, Fedor, Schoffstall, & Davin] is that of a "reboot command". Rather than explicitly implementing a REBOOT command, this action might be invoked by simply setting a parameter indicating the number of seconds until the system reboots.

Identification of Object Instances

The variables (or names) of all object types in the MIB are defined explicitly in the Internet-standard MIB [Rose M.], known as the MIB-II of RFC 1158. The entries in this standardized MIB make it possible for CLIPS to manage TCP/IP network elements in a vendor-independent fashion. Referencing RFC 1158, the CLIPS developer can access any of the defined variables on any network element that supports the SNMP.

Each instance of any object type defined in the MIB is identified with a variable name. The MIB is organized in a hierarchical fashion, thus making it easy to "walk the MIB" to obtain aggregate information. An example of walking a portion of the MIB is to obtain all the information under the variable name "system".

In SNMP the objects are identified with fully qualified variable names in "x.y" format, where "x" is the name of a non-aggregate object defined in the MIB and "y" is the object identifier that is specific to the desired instance. This naming strategy admits exploitation of contiguous lexicographic retrieval of related variables, which makes MIB walking possible.

For example, the fully qualified ASN.1 name that represents how long a network element has been up and running, "upTime", is:

```
iso.org.dod.internet.mgmt.mib.system.sysUpTime.0
 1   3   6     1       2   1     1       2     0
```

The numbers underneath the definition show the integer representation that defines the variable. The CLIPS programmer references variables by the text ASN.1 name, the ASN.1 parser converts the name into the array of integers that correspond to the name for actual transmission over the SNMP. Notice from Table 1 the last statement was the assignment ::= {system 2}. Working backwards one can see that "mib ::= { mgmt 1 }", "mgmt ::= {internet 2}", and so on.

It is possible to obtain the system up time from a remote network element by asking for this variable. Alternatively, all system variables could be retrieved from the network element by requesting "iso.org.dod.internet.mgmt.mib.system".

As a short cut, all the variables are presumed to be preceded with "iso.org.dod.internet.mgmt.mib". Thus a request for system up time merely becomes "system.sysUpTime.0".

There are about one hundred variables defined in the MIB. For purposes of example in this paper, and to eliminate confusion, only a portion of the internet MIB and variables are presented. The MIB variables used in the code fragments presented in this paper are defined in Table 2.

```
system      OBJECT IDENTIFIER ::= { mib 1  }
ip          OBJECT IDENTIFIER ::= { mib 4  }

sysDescr OBJECT-TYPE            -- Describes the system
      SYNTAX      octet-string
      ACCESS      read-only
      STATUS      mandatory
      ::= { system 1 }

sysContact OBJECT-TYPE          -- Who to contact if problem
      SYNTAX      octet-string
      ACCESS      read-write
      STATUS      mandatory
      ::= { system 4 }

sysLocation OBJECT-TYPE         -- Where is this hardware
      SYNTAX      octet-string
      ACCESS      read-write
      STATUS      mandatory
      ::= { system 6 }

ipInDiscards OBJECT-TYPE        -- How many packets discarded
      SYNTAX      counter         -- because of internal errors
      ACCESS      read-only       -- in the system.
      STATUS      mandatory
      := { ip 8 }
```

Table 2. Example MIB Definition

The variables defined implement self-explanatory functions, with the possible exception of ip.ipInDiscards.0. This variable is a counter in the network element that tallies network packets that were destroyed because the network element couldn't process them. Usually this means that there wasn't enough buffer space to process the message. Obviously, if the number of discards goes up, the more problematic the operation of the network will become. It is this variable, ip.ipInDiscards, that will be examined in the further examples.

INTEGRATION OF SNMP WITH CLIPS

The SNMP protocol and an ASN.1 parser have to be built in order to access the MIB in the remote network entities. Fortunately, neither the SNMP protocol nor the ASN.1 parser are particularly hard to come by in the "C" language. There are two public sources of SNMP, one is from Carnegie Mellon University, and the other is from the Massachusetts Institute of Technology. Both are distributed without charge, if you follow their liberal licensing agreement.

The Carnegie Mellon University SNMP was chosen because of the author's familiarity with other CMU efforts and was thus comfortable with their code.

The SNMP code from CMU implements an ASCII database of MIB variables for SNMP, an ASN.1 parser, the SNMP protocol over TCP/IP, and a set of applications programs that allow one to access the MIB variables on the network elements. The SNMP system is designed to run under the UNIX operating system. For the management station, the CLIPS system was built on an Everex 80386 system running the SCO Open Desktop (UNIX System V.32) operating system.

The first step in the process was to build the application programs and use them to access the MIB variables of network elements.

There were two germane applications: "snmpget" and "snmpwalk". The snmpget allows one to get a variable from the network element. The syntax is "snmpget <host-name> <access-control> <asn.1 name>. The access control parameter is the "password" that allows one to access the variables. For read-only purposes "public" will do. Table 3 shows what the snmpget command will return if issued against a network element named "gandalf".

```
$snmpget gandalf public system.sysDescr.0
IBM RS6000, version 1.1 AIX operating system
$
```

Table 3. The SNMPGET Command

However, there was no "snmpset" provided so one was coded using the CMU snmplib.a application library. Then it was possible to both view and modify the variables (if allowed). In Table 3, the system contact name on the remote element will be viewed, then changed.

```
$snmpget gandalf public system.sysContact.0
Edward Williams, 213-555-1212
$
$snmpset gandalf private system.sysContact.0 "Ben Faul 213-555-1212"
$
```

Table 4. View Then Set a Variable

From now on, any one requesting the system contact name of element gandalf will get the new contact name "Ben Faul".

Once familiar with the application programs and the SNMP library, integration with the CLIPS expert system was an easy process. The three application programs were then modified to be incorporated into CLIPS. The UNIX syntax was preserved in the language to allow the application programs' documentation to be used with CLIPS.

Using standard CLIPS implementation methodologies the language was extended to include the following new constructs: (snmp-get), (snmp-getnext), (snmpset).

Surprisingly, these are the only new constructs required in the language to facilitate SNMP access.

Get An Instance Variable

To get the value of a variable on the network element, the (snmp-get) command is used. The form of the command is:

```
(snmp-get <host> <variable-name> <access>)
```

The <host> parameter defines the symbolic name of the element being queried. The <variable-name> is the ASN.1 name of the variable. The <access> parameter is used for authenticating privilege to view or modify the variable. Usually for this command the <access> variable is set to "public".

The snmp-get returns a multi-field variable, The first field is the return code. The second field is the type (integer or string), which is used to field the returned variable in the third field (if integer) or fourth field (if string). If an error occurs a -1 is returned in the type field, with an error string contained in the fourth field.

Get The Next Instance Variable

For getting contiguous variables in the MIB, the snmp-get-next command is provided. The form of the command is:

```
(snmp-get-next <host> <variable-name> <access>).
```

The <host> parameter defines the symbolic name of the element being queried. The <variable-name> is the ASN.1 name of the variable fragment. The <access> parameter is used for authenticating privilege to view or modify the variable. Usually for this command the <access> variable is set to "public".

The snmp-get-next returns a multi-field variable, The first field is the return code. The second field is the type (integer or string), which is used to field the returned variable in the third field (if integer) or fourth field (if string). The fifth field is a string that identifies the fully qualified name being returned. If an error occurs a -1 is returned in the type field, with an error string contained in the fourth field.

Each successive call to the snmp-get-next returns the next variable that was contiguous with the previous variable, until the MIB variables in that fragment is exhausted. In this manner one may examine the MIB by "walking down the branches". Indeed, the entire MIB may be returned by successive calls using the ASN.1 variable: "iso.org.dod.internet.mgmt.mib"

Set An Instance Variable

To set a variable in a network element, the (snmp-set) command is used, assuming that one has authority to do so. The form of the command is:

```
(snmp-set <host> <variable-name> <access> <new-value>).
```

The <host> parameter defines the symbolic name of the element being queried. The

<variable-name> is the ASN.1 name of the variable. The <access> parameter is used for authenticating privilege to view or modify the variable. Usually for this command the <access> variable is set to "private".

The snmp-set returns a multi-field variable, The first field is the return code. If the code is 1 then the replacement was successful. Otherwise, 0 indicates an error occurred with the third field containing an applicable error string.

The architecture of the SNMP/CLIPS integration, and how it is applied to manage a sample network is described in Figure 1.

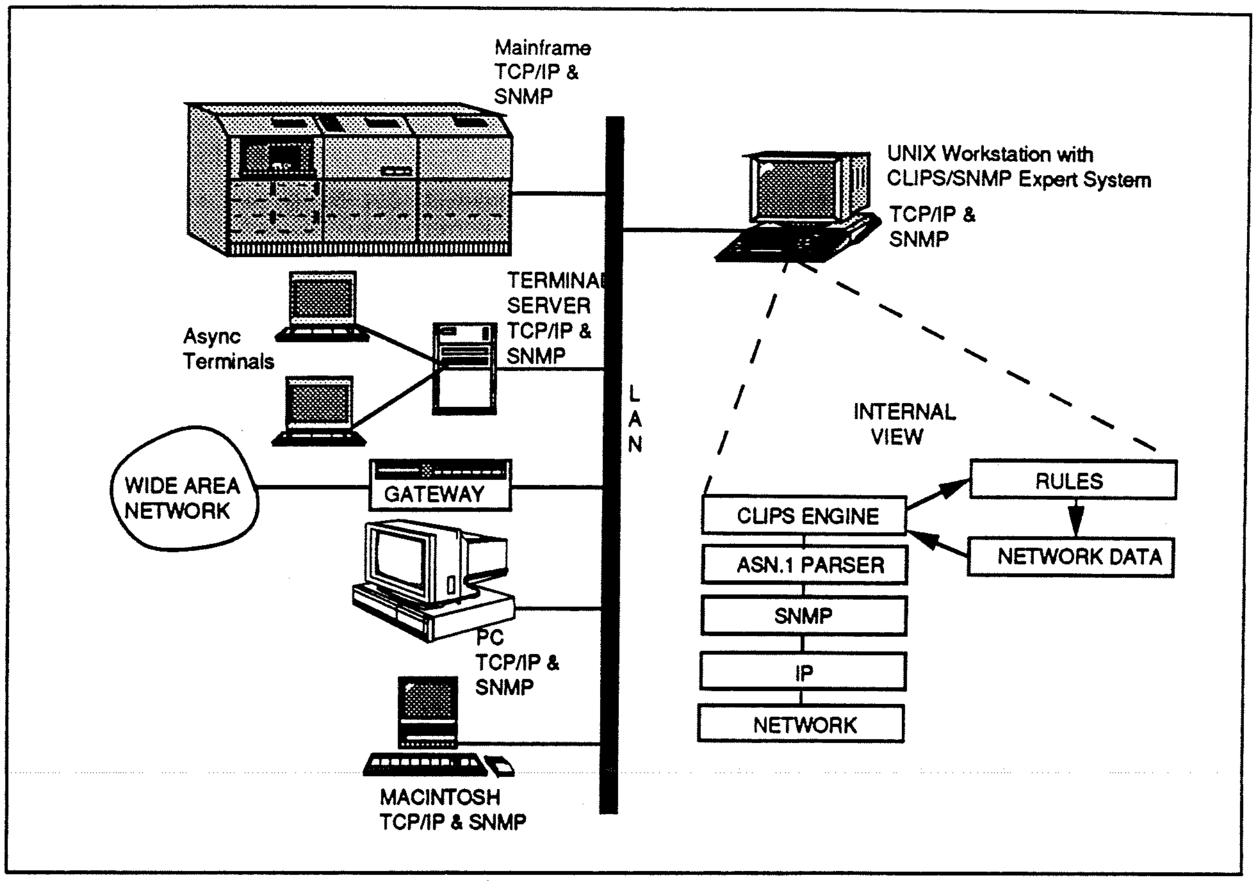


Figure 1, SNMP/CLIPS Integrated Architecture

The network elements, such as the gateway, terminal concentrator, and host interface are all controlled, automatically from the host running the CLIPS expert system. The expert system gathers SNMP data from the network elements and stores it in the data file. The data file is used as feedback to the expert system to change the network element parameters based on trend analysis and knowledge contained within the expert system.

EXAMPLE CLIPS NETWORK MANAGEMENT SYSTEM

The previous sections dealt with building the infrastructure in CLIPS to enable it to do the work of network management. The concepts of proactive network management will be discussed in this section and displayed in the code fragment in Table 5, below.

Network Initialization

The network manager expert system must obtain the element names of the various network entities to be managed on the network. These are simply stored in an ASCII database from which the expert system reads upon initialization.

A more flexible approach would be to utilize a database for the network elements, but that would complicate the design for example purposes.

Network Status Information

The network manager expert system polls the network elements for various performance parameters. The information returned is stored in an ASCII flat file for future reference.

Problem Detection

Problem detection is accomplished by applying rules against the ASCII flat file that holds network status information. Upon finding a potential problem, the expert system asserts the facts determined by the current and historical status as determined from analyzing the ASCII file.

Problem Correction

The network manager expert system's problem correction rules attempt to solve the problem by class of failure. In the case of overloaded gateways, routing tables may be adjusted to alleviate traffic in this element. In the case of an interface card reporting many transmission errors, a diagnostic printout showing the location of the unit and the type of error condition may be printed.

ILLUSTRATIVE EXAMPLE

To see how well SNMP and CLIPS work together, consider the CLIPS code fragment presented in Table 5.

```
        (bind $?discard-data
              (snmp-get ?element "public" "ip.ipInDiscards.0")
        (bind ?rc (nth 1 $?discard-data))
;
; If no network error then process, else note the error
;
        (if (ne 1 ?rc)
        then
              (printout t "Error getting data from " ?element)
              (printout t "     Error = " (nth 4 $?discard-data))
              (assert network-error ?element))
        else
;
; Determine if packet discard threshold has been exceeded
;
              (if (> (nth 3 $?discard-data) ?max-allowed)
              then
                    (bind $?contact-data
                          (snmp-get ?element "public" "system.sysContact.0"))
                    (bind $?location-data
                          (snmp-get ?element "public" "system.sysLocation.0"))
                    (printout t "Packet discards exceeded for " ?element)
                    (printout t "   Likely problem is memory exceeded")
                    (printout t "      Check unit at location "
                          (nth 4 $?location-data))
                    (printout t "      Notify "
                          (nth 4 $?contact-data))
                    (printout t "   Re-route in progress)
                    (assert reroute-to ?element)))
```

Table 5. CLIPS/SNMP Sample Code

As can be seen in Table 5, the addition of the SNMP capability to CLIPS does not greatly influence the character and flavor of the language. However, the addition of the SNMP access allows the full power of the language to be used to influence the performance of a network of SNMP compliant commercial products.

CONCLUSIONS

The application of CLIPS to SNMP has proved to be quite successful. The network manager expert system is capable of detecting faults and does a credible job of proactive management.

To enhance the proactive management capabilities, the expert system should utilize an SQL database to store the polled data. This would permit more extensive trend analysis to be done.

The SNMP library from CMU utilizes a synchronous network connection. This means that the (snmp-get) hangs until the message is returned to CLIPS or the read times out. To manage a very large number of elements efficiently within CLIPS, non-synchronous network I/O will be required.

The user interface of the network manager is a simple X Window System/MOTIF GUI that uses very simple icons to represent the different entities. A full featured X Window System interface that is truly object-oriented is a must for building a serious network management product.

REFERENCES

Case, J., Fedor, M., Schoffstall, M., Davin M. (1990) A Simple Network Management Protocol, *DARPA RFC 1157.*

Hansen, R.F., Flores, L.M., (1990) JESNET Expert Assistant, *First CLIPS Conference Proceedings, Houston, pp.140-146.*

ISO Standard 8824 (1987), "Specification of Abstract Syntax Notation One (ASN.1), *International Organization for Standardization*

Leigh, A.B. (1990) The Network Management Expert System Prototype for Sun Workstations. *First CLIPS Conference Proceedings, Houston, pp. 148-154.*

Rose, M. (ed) (1990) Management Information Base for Network Management of TCP/IP based Internets: MIB-II, *DARPA RFC 1158*

N92-16572

NMESys: AN EXPERT SYSTEM FOR NETWORK FAULT DETECTION

Peter C. Nelson and Janet Warpinski

Department of Electrical Engineering and Computer Science
University of Illinois at Chicago
Chicago, IL 60680

Abstract. The problem of network management is becoming an increasingly difficult and challenging task. It is very common today to find heterogeneous networks consisting of many different types of computers, operating systems, and protocols. The complexity of implementing a network with this many components is difficult enough, while the maintenance of such a network is an even larger problem. This paper presents a prototype network management expert system, NMESys (pronounced nemesis), implemented in CLIPS. NMESys concentrates on solving some of the critical problems encountered in managing a large network. The major goal of NMESys is to provide a network operator with an expert system tool to quickly and accurately detect hard failures, potential failures, and to minimize or eliminate user down time in a large network.

1.0 INTRODUCTION

The problem of network management is becoming an increasingly difficult and challenging task. Networks can fail at many different components, connections and levels, often potentially disrupting service to many users. Sometimes, portions of the network can detect a failure in other portions of the network. Other times, the fault may go undetected by the network. NMESys (Network Management Expert System) is a prototype network management system which monitors alarms in a network and helps a network operator determine points of failure. NMESys is able to receive and decipher information from components in the network about detected failures. NMESys also has the ability to proactively interrogate the network to find undetected failures. The primary goal of NMESys is to detect and isolate faults so they can be repaired with minimal or no user down time.

2.0 SYSTEM OVERVIEW

There are many problems to be solved when managing a large heterogeneous network. First, just determining that a failure has occurred can be a difficult problem, even if the failing component or an adjacent component has detected the failure. Often large networks post many types of messages to report conditions which may or may not warrant operator intervention. Operators may be bogged down researching dead ends, while some major component is in a state of failure. NMESys is able to filter out the messages which do not indicate hard failures. This allows operators to concentrate on the problems which pose the greatest threat to the integrity of the network.

Another problem in managing the network is determining conditions which are degrading the network. Often error messages do not necessarily indicate a hard failure, but rather some temporary error condition. Since the component recovers, these conditions often go unattended. With many of these types of messages being generated in a large network, operators do not have the time to research and resolve each one. However, this type of event can be an indicator of a more severe problem which is starting to manifest itself. If the problem goes on long enough, it may result in an extended outage of a component which could have been repaired before the hard failure occurred. NMESys tracks all conditions which have been reported to it, even if the indication was non-fatal. For degrading conditions, threshholds are utilized to determine when a non-fatal condition warrants attention. If a threshhold has been exceeded within a certain time frame, then an error message is posted to the network operator indicating that a potentially serious condition has developed. Thus the problem can be addressed before the failure even occurs.

NMESys always knows the status of each component in the network, since it receives all messages which indicate changes in state of the components. The system can always give an operator the current status of any component or set of components. This can be very helpful to an operator to know how many components are down at any one time. NMESys also has the ability to show the message history for any component so that problems can be researched. In addition, NMESys can give an operator a list of events which have occurred in the network which need action. As operators take care of these events, they are acknowledged. This allows the system to always have a current list of events warranting attention.

NMESys has the ability to interrogate components in the network about their current status. This functionality provides a proactive approach to detect conditions which, for whatever reason, have not been reported properly. Often, error conditions can go undetected by either the failing component or any component communicating with it. NMESys periodically initiates these integrity checks to determine whether its current view of the state of the network is correct. If some error is detected, then the condition is handled just as a reported failure would have been.

3.0 THE METHODOLOGY OF NMESys

NMESys is currently implemented on a DOS machine. This PC is connected to the network so that it becomes the network monitor. The interface to the network is implemented in C. The alarm processor and user command processor are implemented in CLIPS. These are used for the interpretation and tracking of alarms. This design strategy makes NMESys more flexible since only the C component would need major changes if a new type of network were being monitored. NMESys has three major components: the alarm and user interface, the alarm processor, and the user command processor. The basic architecture of NMESys and the interaction between the components is shown in Figure 1.

3.1 THE ALARM AND USER INTERFACE

The alarm and user interface handles incoming alarms, timing for integrity checking, and the user input menu. This portion is written in C for three reasons. First, this task must be able to communicate with external devices. Second, it must be able to receive new alarms from the network as well as detect that the user has initiated a command from the menu. Third, it is the component in the system which implements time. Since NMESys is a prototype system and is not connected to a network at this time, the alarm and user interface also contains a random alarm generator. When an alarm is generated, it is passed to the CLIPS alarm processor. The current time is also sent to CLIPS so that degrading conditions beyond their threshhold can be properly determined. The user menu is also presented from this process. If a command is initiated, then the appropriate command request is passed to the CLIPS user command processor. In both cases, control is returned to the alarm and user interface when processing is completed. In addition, every five minutes a command is automatically generated for CLIPS to initiate any integrity checks which are due. This command tells CLIPS the current time so that the appropriate calculations can be made.

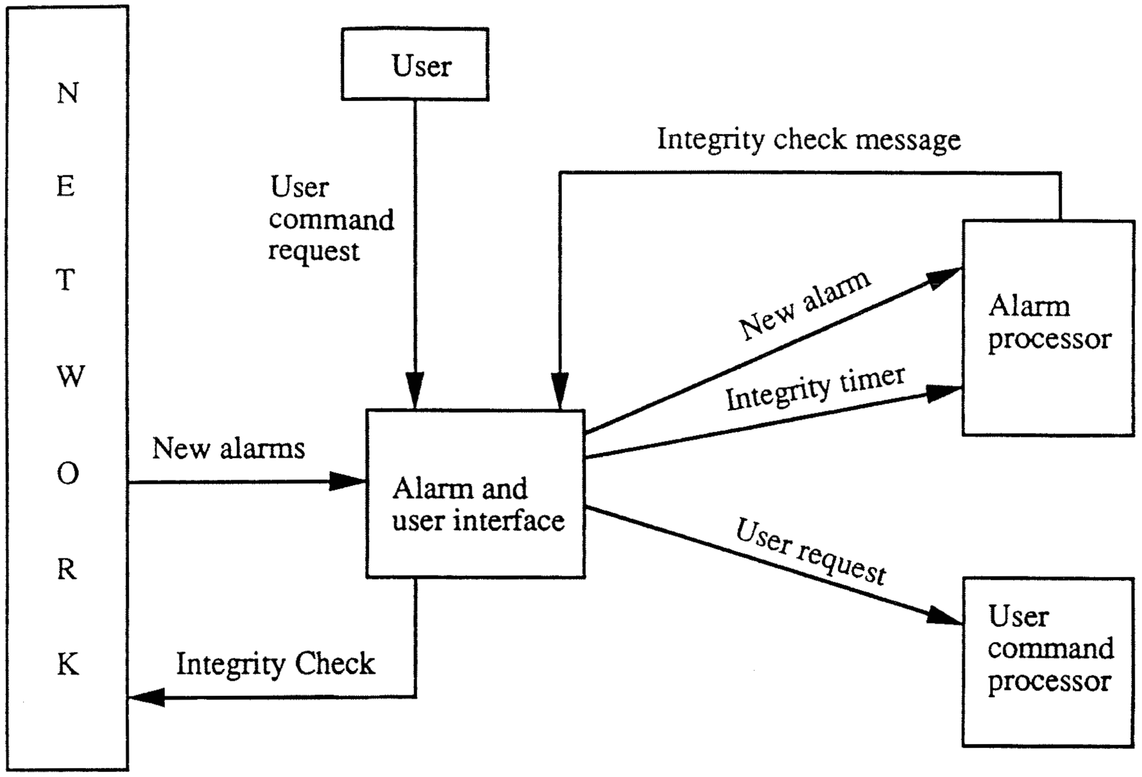


Figure 1. NMESys architecture

3.2 THE ALARM PROCESSOR

The alarm processor is written in CLIPS. It contains a list of all alarm types which can be generated by the network. Thus the new state of the component can be determined by the type of alarm which has been received. Since the alarm types and their characteristics are facts to CLIPS, implementation of a new type of network would involve only redefining the new alarms to CLIPS. The received alarm is always logged for site history purposes. If the alarm indicated a DOWN condition, an alert is generated and posted to the screen. The alert is logged so that an operator can always see the current list of alerts. If the alarm indicated a DEGRADING condition, then CLIPS checks if the threshhold for that alarm has been exceeded. If so, an alert is posted. If the alarm indicates an UP condition, any alerts which are active for this site are automatically removed from active status. Thus, if alerts have been logged for a condition, but service has been restored, the preceeding alerts disappear from the active alert list. If this were not the case, then an operator would be forced to investigate each condition, only to find that the outage has been restored. By watching the active alert list, the operators only need to look into conditions which reflect the current status of the network.

The alarm processor also handles integrity checking. Integrity checking provides an additional level of confidence in the accuracy of the status of the network. Without a method of detecting failures, the network management system is only as powerful as the failure detection process of its weakest component. When an integrity check message is received from the alarm and user interface, CLIPS determines the last time each site received an integrity check. If one is due, then the appropriate message is sent to the interface so that it can be in turn sent to the appropriate site. NMESys knows what type of equipment is at each site. It also knows the format of the proper integrity check message by the equipment type. These are the two pieces of information which must be sent to the interface. Once again, the implementation of a new piece of equipment or an entirely new network would only involve definition of the new type(s) of equipment and the appropriate integrity message.

3.3 THE USER COMMAND PROCESSOR

The user command processor is also implemented in CLIPS. The commands which are implemented in the user menu are shown in Figure 2. The command processor contains commands to show all sites which are UP, DOWN, or DEGRADED. The operator can also show all alarms for a particular site, show all active alerts, or acknowledge a particular alert. The commands to show UP, DOWN, or DEGRADED sites allow the operator to get a current status of the network at any point in time. This is also useful when a network user is reporting a problem. The operator can check the status to see if any of the received alarms could be causing the reported failure. There is also a command to view the history of alarms for a particular site. This is most useful when diagnosing a problem to try to determine the exact point of failure. Network operators typically must research failure conditions which have originated from either user complaints or from received alarms. When a network user calls in a complaint, the actual point of failure may not be as obvious as when an alarm is reported. Thus, a network operator must have the ability to view the history and status of many of the components in the network to determine the exact point of failure.

There are also two alert commands which are most useful to the network operator diagnosing problems from conditions which have been reported through the network rather than user reported problems. One of the principles of NMESys is that it can filter alarms which do not need attention from an operator. An alert is a detected condition which warrants action. Thus, an alert may be initiated by a single alarm, or by some combination of alarms which indicate a failure. The command to view all active alerts is probably the most utilized command. This allows the operator to see all events which require some action. The network operator's job is to resolve each of these events one by one. If no events are on the list, then all conditions have either been resolved or are in the process of being resolved. The command to acknowledge an alert allows the operator to tell NMESys that the alert has been recognized. This does not necessarily mean that the situation has been resolved. It might mean a repair technician has been dispatched or that the appropriate agency responsible for the equipment has been notified. By acknowledging the alert, the active alert list can be maintained as only the events which still require operator action.

```
          Menu

1) Show DOWN sites
2) Show DEGRADED sites
3) Show UP sites
4) Show all alarms for a site
5) Show active alerts
6) Acknowledge an alert

0) Exit

Enter selection:
```

Figure 2. The user menu

4.0 RESULTS

NMESys is a prototype expert system, programmed in CLIPS, used to perform the task of network management. CLIPS made the alarm processing and user command processing tasks quick and easy to write. Information parsing and error checking routines can be implemented in just a few lines of code. This made it very simple to write and test routines quickly. In the future, enhancements will be implemented and evaluated without a large investment in programming effort. This is obviously an excellent type of environment for prototyping systems. CLIPS can also interface with system level commands, in this case, DOS commands. Thus, CLIPS can be used in conjunction with any other tool or user program. If there is some task that CLIPS cannot accomplish, the task can be written in another language and interfaced to CLIPS. This was the case in NMESys where C was utilized to communicate with the network.

The use of DOS presented some problems, mainly because of the lack of multitasking. C was utilized so that NMESys could process network messages while waiting for user commands. The requirements of NMESys do not allow the use of the CLIPS read statement for user input. If the CLIPS read statement were used, then the operator could be at a read prompt while alarms were coming in from the network and NMESys would not be able to process the alarms. There would be no way to limit how long the operator remained at the prompt. All input from the operators had to be implemented in C and interfaced to CLIPS.

NMESys also brought out some application level discoveries and problems. It was initially thought that the status type commands would be most helpful to the network operator. However, it soon became apparent that the alert portion of the system was more helpful. The active alert list became the work queue for the network operator. This section of the system seems to have the most potential for expansion to further aid the network operator and produce more accurate diagnosis.

5.0 CONCLUSIONS AND FUTURE WORK

NMESys has a number of benefits to assist in the task of network management. First, the network operators's job is eased since NMESys does all the tracking of equipment states as well as events which require action. The system provides more accurate real time status than a human operator could provide. This allows the network operators to have timely information when network users call in to report problems. NMESys also maintains the work queue for the network operator by defining the active alerts. The network operators only need watch the alert list to determine what areas in the network need attention. NMESys is also flexible since a different type of network could easily be implemented by only changing the C interface and defining the facts for the types of alarms and equipment in the new network.

There are many potential enhancements which are planned for NMESys to increase its functionality. First, a chronic function should be implemented. This function would alert an operator when a site has posted too many alarms in a certain time period. For example, a site may post many alarms which clear very soon afterward. An operator might not even see the alarms if they clear quick enough since NMESys will remove active alerts when an UP condition is detected. Or, if different operators work on the problems, they may not realize that the site has actually exhibited many problems. Another condition which could occur is that the alarms are diagnosed as no trouble found situations. In this case, the operator might be able to clear the problem by executing some sequence of commands at the computer site. In both of these situations, the operators should be alerted that a particular problem is occurring over and over again. This would give the operators the indication that these are not random harmless failures, but possibly the result of a problem which is manifesting itself. Thus, similar to the threshhold concept for degrading conditions, sites which show many short spans of down time can be diagnosed as to the true cause of the failure.

Another enhancement which could be very helpful would be to expand how active alerts are purged from the list when an UP condition is detected. For example, an UP condition for some type of DOWN alarm might not always mean that all DOWN conditions have been restored. An example would be if a DOWN alarm indicated a failure on a particular connection to another computer and another DOWN alarm meant that the printer was not working. If the computer connection comes back UP, this does not mean that the printer has also been restored. NMESys should contain an alarm cross reference table to indicate which alarms are related. Thus, when an UP condition is received, only related DOWN alarms are removed from the active alert list.

Many enhancements could also be made to the functionality of the alert list. When an operator acknowledges an alert, a reason for the resolution should be requested from the operator. For example, the operator might indicate that the disk drive was replaced, or that there was no trouble found. This allows NMESys to begin to help the operator diagnose the potential cause of the failure before researching the problem. Over time, trends can be developed on the types of fixes which occurred depending on the reported alarm code. The operator could query NMESys on the past history of a certain alarm code. NMESys could indicate the percentage of each kind of resolution code for the history of alarms. This feature would be extremely helpful to the less experienced network operators. These are just a few of the areas where NMESys could further help the network operator to perform the job more accurately and efficiently. As NMESys continues to grow, more and more enhancements can be envisioned, each building toward a fully automated network monitoring and diagnosis system.

6.0 REFERENCES

[1] Callahan, P., "Expert Systems for AT&T Switched Network Maintenance", AT&T Technical Journal, Jan. 1988

[2] Giarratano, Joseph, Riley, Gary, "Expert Systems Principles and Programming", PWS-KENT Publishing Company, 1989

[3] Harmon, Paul, et al., "Expert Systems Tools & Applications", John Wiley & Sons, Inc., 1988

[4] Vesonder, G. et al., "ACE: An Expert System For Telephone Cable Maintenance", IJCAI (1983) pp. 116-121

[5] Waterman, Donald A., "A Guide to Expert Systems", Addison-Wesley Publishing Company, 1986

N92-16573

A MISSION EXECUTOR FOR AN AUTONOMOUS UNDERWATER VEHICLE

Yuh-jeng Lee and Paul Wilkinson

Computer Science Department
Naval Postgraduate School
Monterey CA 93943

Abstract. The Naval Postgraduate School has been conducting research into the design and testing of an Autonomous Underwater Vehicle (AUV). One facet of this research is to incrementally design a software architecture and implement it in an advanced testbed, the AUV II. As part of the high level architecture, a Mission Executor is being constructed using CLIPS version 5.0. The Mission Executor is an expert system designed to oversee progress from the AUV launch point to a goal area and back to the origin. It is expected that the Executor will make informed decisions about the mission, taking into account the navigational path, the vehicle subsystems health and the sea environment, as well as the specific mission profile which is downloaded from an offboard mission planner. Heuristics for maneuvering, avoidance of uncharted obstacles, waypoint navigation, and reaction to emergencies (essentially the expert knowledge of a submarine captain) are required. Many of the vehicle subsystems are modeled as objects using the CLIPS Object Oriented Language (COOL) embedded in CLIPS 5.0. Additionally, truth maintenance is applied to the knowledge base to keep configurations updated.

AUTONOMOUS UNDERWATER VEHICLE RESEARCH

The development of autonomous vehicles has been an ambition for decades. Automated weapons such as the Tomahawk missile now have a proven record of achievement in hazardous conditions. The MAZLAT/AAI Pioneer, a remotely-piloted vehicle (while not fully autonomous), similarly has a capable record in high-risk environments, as evidenced by the Gulf War. Several marine autonomous and remotely-piloted vehicles are already in use for such diverse functions as underwater cable inspection, hydrography, and mine-hunting. The practical advantage of low-risk to humans coupled with the potential ability to operate at over-the-horizon distances from the control platform make the autonomous underwater vehicle a highly desirable project. While there are several operational autonomous underwater vehicle testbeds in the United States, until recently most underwater vehicles have been tele-operated or merely data autonomous while receiving power via an umbilical cable.

Many software architectures have been proposed and are currently being tested for a fully autonomous underwater vehicle. One of the well-known is MIT's Sea Sprite Vehicle which adapted the layered control architecture proposed by Brooks (Bellingham 1990, Brooks 1986). The KB/EAVE (Knowledge-Based Experimental Autonomous Vehicle) AUV program of the University of New Hampshire's Marine Systems Engineering Lab essentially uses a subsumption architecture (as generally described by Brooks). High level and low level tasks are divided in hardware. The software uses the "focus of attention" approach to keep upper-level reasoning foremost while low-level behaviors occur (Blidberg 1990). International Submarine Engineering of Canada also uses a layered control architecture with behaviors classified as reflexive, logical, and trained. These require reasoning on several levels, with planned and learned responses, encoded in a scripting language instead of a traditional AI language (Zheng 1990).

The Naval Postgraduate School has been conducting research into the design and testing of an Autonomous Underwater Vehicle. Both high-level and low-level software have gone through several versions of development. Currently, the software is destined to reside on a GESPAC MPU30HF processor board using the OS-9 operating system on a Motorola 68030 central processing unit. From a software architecture standpoint, the AUV software can best be designed in a hierarchical structure and viewed at different levels of abstraction for different purposes, for example, mission planning, mission execution, world modeling, collision avoidance, and vehicle control. This software has to perform both numeric computing and symbolic reasoning. Most of the computations also involve real-time constraints and time-dependent representations of the states of the AUV and the environment. In addition, many tasks are knowledge intensive and require domain specific information. For example, the collision avoidance routine needs to interpret sensor input, react to uncharted obstacles, replan a new vehicle path or mission based on available choices, and so on.

The NPS AUV II software is partitioned into several main modules, including an off-line mission planner, mission executor, guidance system , autopilot system, navigation system, sonar data processor, on-board mission replanner, and vehicle system monitor (Healy et al. 1990). Each of the modules contains submodules performing more specific tasks. For example the autopilot system includes routines for digital-analog data conversion, for hydrodynamic surfaces control, and for main motor control; the guidance system includes a local path planner for creating postures form waypoints and the tracking controller for providing desired postures to the autopilot (Cloutier 1990, Lee et al. 1991). Figure 1 shows the dataflow diagram of the AUV II baseline system.

DESIGN OF SKIPPER

The high-level design of the AUV II is the result of an incremental development which began in 1988 with AUV I. Initially, vehicle control was essentially lower-level closed-loop. Evolutionary changes in subsequent software designs resulted in the need for a high-level control module to coordinate the functionalities of various subsystems. The Mission Executor, SKIPPER, attempts to do this while integrating decisions based on input from three worlds: the vehicle's internal systems, the environment, and the mission. The design of Mission Executor essentially consists of a rule base and an object base. The major equipments aboard the submarine are modeled as objects to be monitored. Further, each obstacle encountered by the submarine sonar, whether planned for or not, is modeled as an object. Decisions on courses of action to take are modeled as objects for the purpose of easy retrieval via hyperlinks (this will be more self-evident shortly). All of these are linked together in the SKIPPER's Display, a blackboard subset of all of these. The SKIPPER's Display is a composite of the most vital information and consists only in the current decisions, obstacles which are still active (those in a 180-degree arc about the bow of the submarine), and the current state of the system monitors. This intelligent database is frequently updated and queried by the rule base.

While the vehicle's different states are updated and monitored by querying objects (and firing attached daemons), the heuristics for the three worlds (internal, environmental and mission)

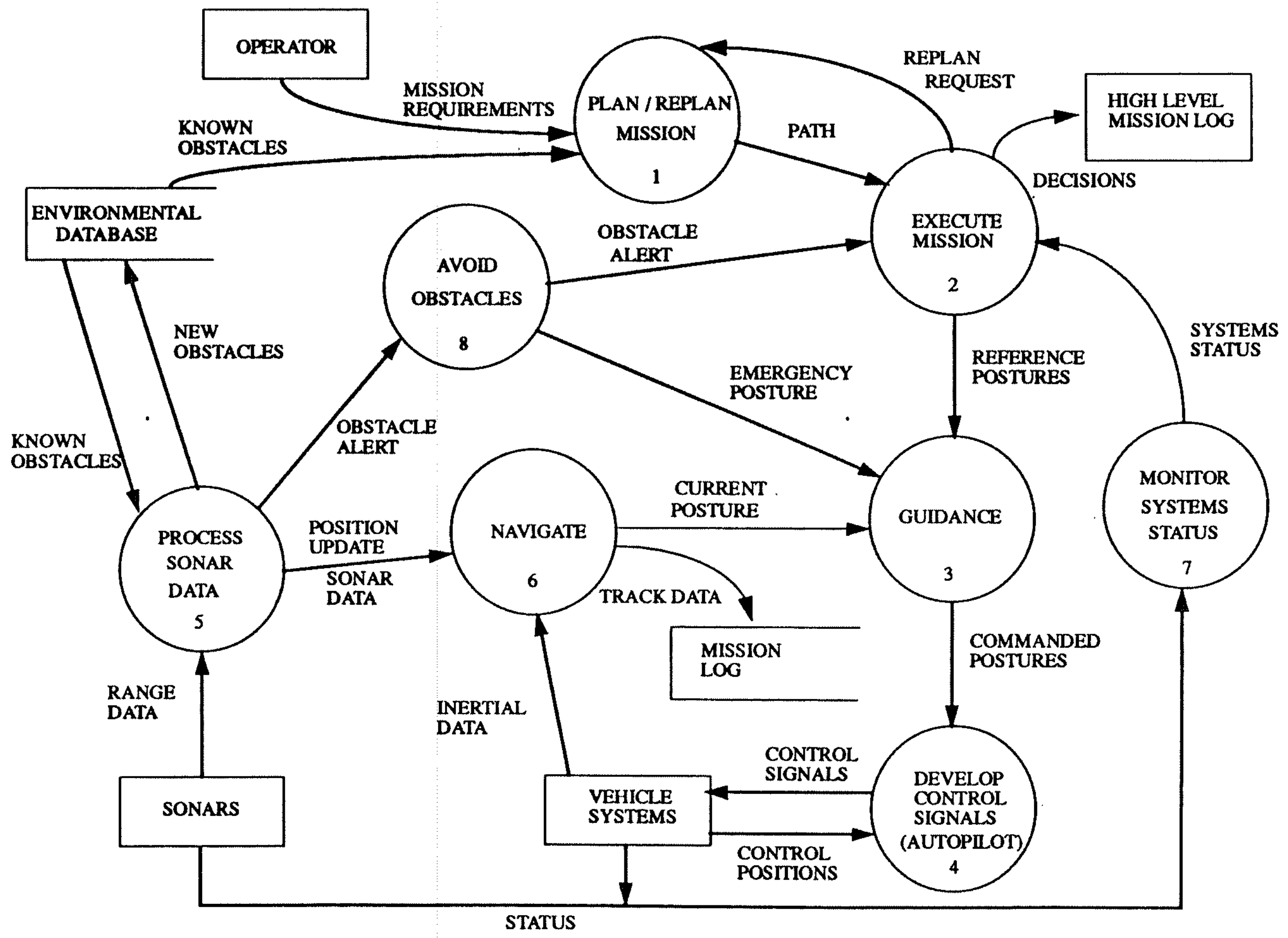


Figure 1. AUV System Dataflow Diagram

are contained in the rule base, which is partitioned into divisions of vehicle maneuvering rules, system monitor rules, navigation rules, environmental hazard rules and specialized mission rules. Input to the Mission Executor consists in both the internal vehicle configuration and mission plans (designed as vehicle postures of nineteen variables at the various waypoints). Output from the Mission Executor consists of final reference postures passed directly to the lower level Guidance system. Lower level Guidance reacts by controlling the autopilot at the next level. Figure 2 shows this decision process.

Input mission postures are first given to a Mission Interpreter which places a posture into the proper object format and designates the high-level classification of the configuration as transit or specialized mission. It further determines a lower level of configuration as a turn, ascent, dive or surfacing based on the succeeding posture. Navigation rules determine whether the next waypoint will be made on time. Obstacles or elapsed time may determine that a new or updated waypoint be constructed. The Navigator Module (external to the Mission Executor) is invoked by SKIPPER for this purpose. In the event that an obstacle or obstacles force a detour in the path execution of the AUV, an Obstacle Avoidance DecisionMaker invokes the replanner (also external to the Mission Executor) to plan a new route to the goal mission area. The new route is evaluated for both proximity to the old route and ability of the AUV to reach the destination and carry out the mission with available battery power.

Measures of uncertainty are used for initial sonar obstacle determinations which SKIPPER receives from the Obstacle Avoidance Decision Maker. As the classification improves, certainty of the obstacle's location better fixes the progress of the transit or mission. It also allows for determination of whether the obstacle(s) in question requires avoidance maneuvers.

IMPLEMENTATION IN CLIPS 5.0

The decision to build a Mission Executor in CLIPS was made in the fall of 1990 based on the rapid prototyping capability of CLIPS. Its LISP-like rules, relative compactness and low-cost are attractive features for a control system designed to fit in a compact real-time testbed. Further strengthening the argument for CLIPS is an evaluation by William Mettrey of Bell-Northern Research which compared CLIPS against other rule-based tools. CLIPS outperformed three of the other four tools (all commercial) (Mettrey 1991). Balanced with its low-cost, it was clearly the winner.

Initial development actually focussed on modeling the internal world of vehicle systems. The model of this ***internal world*** turned out much like the model Giarratano used for his Joe's Object Oriented Database (JOD) (Giarratano 1991a). The implementation is somewhat different. This is not meant to be a user-interfaced advisory system. Using low salience , a monitor-health-continuously rule checks the state of thirteen instances of various equipment objects (nearly as a background function). The equipment objects all share the common attributes that they are being monitored for their respective high/low redline thresholds and high/low guardline thresholds. The system monitor class is further broken down into a sonar class (there are four sonars on the testbed), a control system class, an onboard computer class, a navigation instruments class with instances of dead-reckoning analyzer (DRA) and Global Positioning Satellite (GPS) receiver, and an environmental sensors class [figure 3]. Through queries and daemons, the changing object states cause pattern matches in the system monitor rules.

Decision-making, while contained in the rule base, is preserved in the object base by the

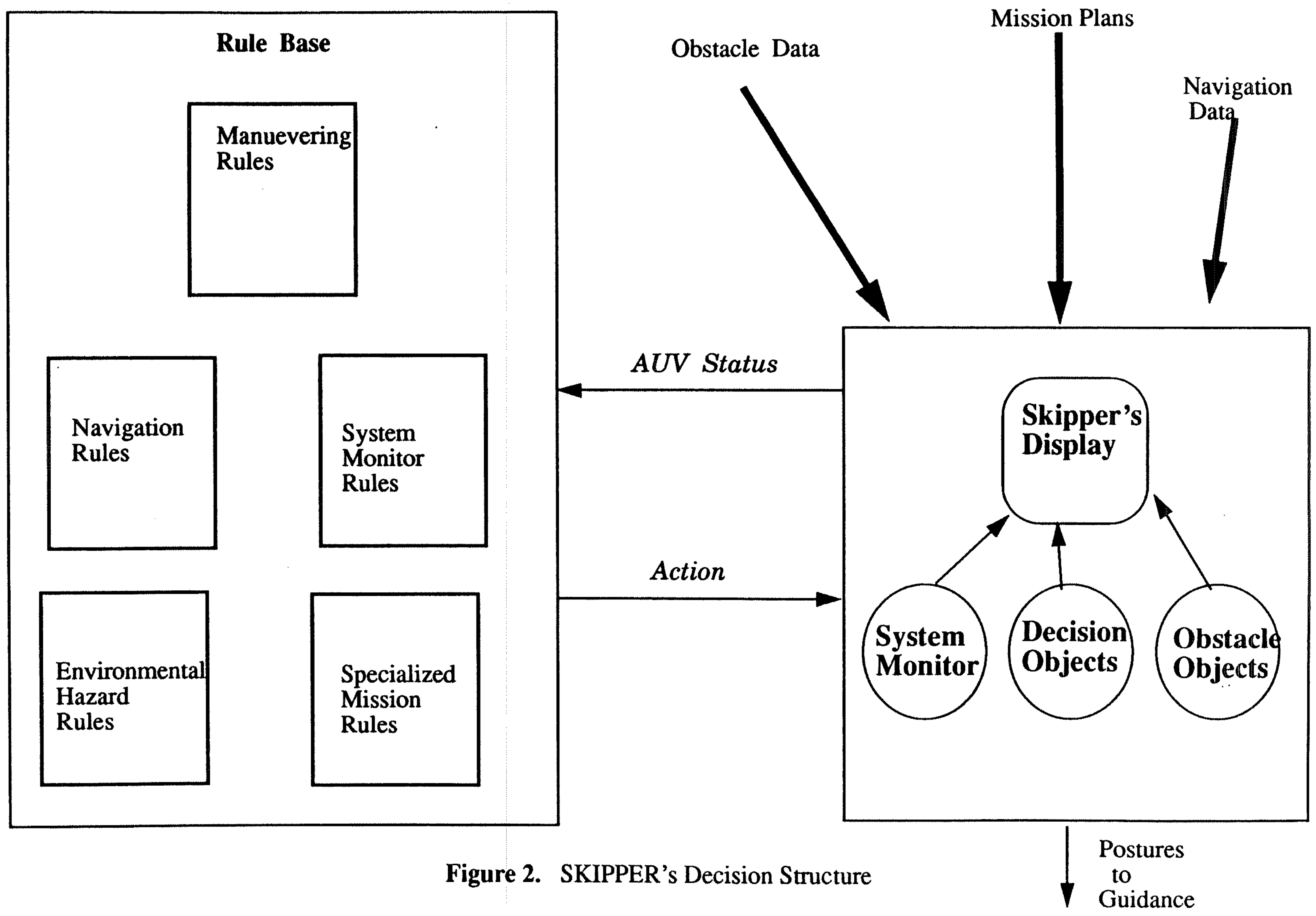


Figure 2. SKIPPER's Decision Structure

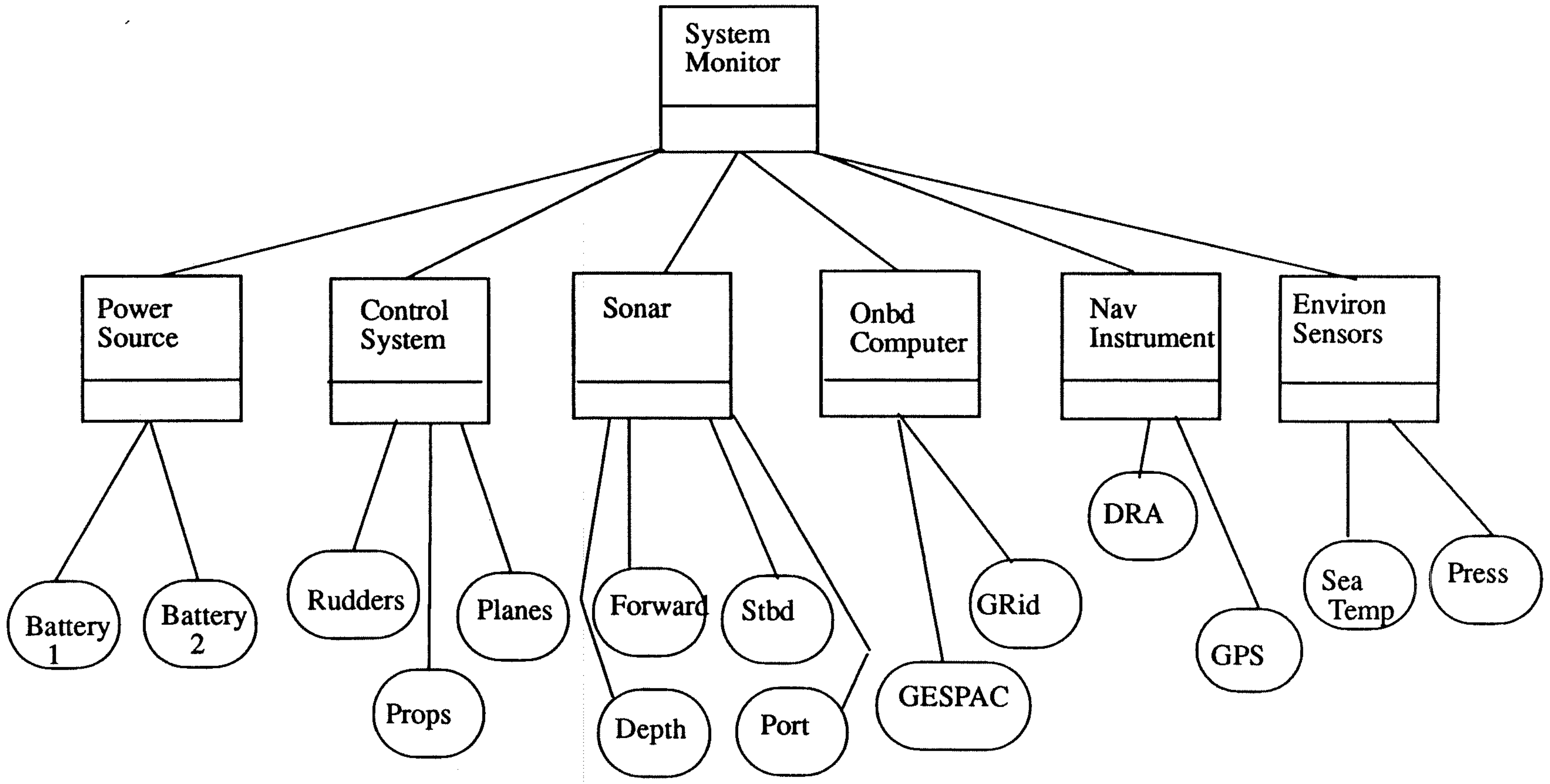


Figure 3. System Monitor Objects

decision-objects. This is because some decisions may require knowledge of previous decisions. This is particularly true for the high-level mission decisions. The design of the decision objects incorporates slots for high-level mission decisions, lower-level manuever decisions, navigation decisions, system-monitor decisions, and special-mission decisions. In addition, provision is made for time-stamping the decision. Literally any decision-change will cause a new decision-object to be created, as a record must be maintained of all decisions. The instance **current** is copied to another unnamed instance using *deffunction* calls, as shown in the following example:

```
(deffunction copy-old-instance (?instance)
  (send (symbol-to-instance-name ?instance) put-mission_decision
     (send [current] get-mission_decision))
  (send (symbol-to-instance-name ?instance) put-maneuver-decision
     (send [current] get-maneuver_decision))
  (send (symbol-to-instance-name ?instance) put-sysmonitor_decision
     (send [current] get-sysmonitor-decision))
  (send (symbol-to-instance-name ?instance) put-navigation_decision
      (send [current] get-navigation-decision))
  (send (symbol-to-instance-name ?instance) put-special-mission-decision
      (send [current] get-special-mission-decision))
  (send (symbol-to-instance-name ?instance) put-justification
      (send [current] get-justification))
  (send (symbol-to-instance-name ?instance) put-decision_time)
     (send [current] get-decision_time)))


(defclass DECISION (is-a USER)
  (slot mission_decision (multiple))
  (slot maneuver_decision)
  (slot sysmonitor_decision)
  (slot navigation_decision)
  (slot special_mission_decision)
  (slot justification)
  (slot decision_time))


(deffunction maneuver-decision-change-obstacles (?change ?justification)
  (bind ?name (gensym*))
  (make-instance ?name of DECISION)
  (copy-old-instance ?name)
  (send [current] put-maneuver-decision ?change)
  (send [current] put-justification ?justification)
  (send [current] put-decision_time (time)))
```

Certain physical changes to the vehicle's environmental or internal world may improve the state of the vehicle somewhat. Yet, the mission-world must dominate behavior. If a mission decision was previously made to *continue_with_restrictions or abort_mission*, improvement in the other two worlds may or may not justify improvement to *continue unrestricted.* To prevent a collision of *defrules,* another basis must be used, such as the justification for the continue with restrictions state. Retrieval of the justification for the previous mission status may involve searching back over several state changes. This should not involve a lengthy amount of traversal. This is more easily done with hyperlinks between objects or a simple query rather than a linked-list. The following example of a post-casualty vehicle recovery rule highlights this. While the left-hand side (LHS) conditions indicate that the mission may be fully recoverable, the right-hand side query hunts for the existence of the only possible justification for full recovery. This further requires a call to a deffunction to determine if the mission is physically recoverable in terms of mission parameters mission-critical power and distance/time-to-go (called from the navigation module external to the Mission Executor).

```
(deffunction recovery-mission-evaluation ( ?location )
  (if (or (< (send [battery] get-power_status) ?*mission-critical-power*)
        (> (navigator-update-from ?location)  ?*recovery-time*)) then
    (send [current] put-mission-decision Abort_Mission)
    (send [current] put-justification mission-deviation-nonrecoverable)
   else
    (send [current] put-mission-decision Continue_Unrestricted)
    (send [current] put-justification mission-deviation-recoverable)))


(defrule vehicle-recovery-state
  (mission_status Continue_with_Restrictions)
  (system-monitors normal)
  (location ?location)
  (or (redundant_system_online ?system)
      (normally-operating ?system))
 =>
  (do-for-instance ((?ins DECISION)) (eq mission-deviation
                   (send ?ins get-justification))
  (recovery-mission-evaluation ?location )))
```

Certain high-level behaviors, such as the overall mission decision are modeled using the Artificial Neural Paradigm implementation suggested by Giarratano (Giarratano 1991). This application of salience is useful in differentiating between a high-level, less frequent macro-action and a lower-level frequently performed action. The philosophy for using salience in this manner is that a situation (pattern match) which may cause a mission abort usually requires immediate or timely reaction and certainly takes precedence over a routine action such as a normal turn or depth change in a deep-water open-ocean environment. The emergency-action rule must be guaranteed firing before other semantically lower-priority rules on the *agenda*. This (however loosely) heuristically models a submarine commander's "situational awareness" in an emergency

```
(defrule emergency -evasive-maneuver
  (declare (salience 1000))
  (obstacle-proximity  ?direction  danger-close)
  (maneuver-available ?maneuver)
  (system-monitors  ?status)
  (not (previous_mission_decision abort-mission))

  =>
 (assert  (emergency-guidance ?maneuver))
  (assert  (mission-decision  alter-track))
  (Replanner get-new-route  ?position))



 (defrule battery-power-guardline
  (declare  (salience  ?*sysmonitor-salience*))
  (mission-percentage ?percent&:(< ?percent  70))
  (battery ?number  at-guardline)
  =>
  (bind ?*sysmonitor-salience* (+  ?*sys-monitor
    salience   100))
  (assert   (mission-status critical)) )
```

Figure 4. Setting Precedence with Salience in SKIPPER

[figure 4].

Salience is also used in some background functions such as the sequencing of the mission timer and the loop which causes the slots of the respective system monitors to be queried on a nearly continuous basis. Still, it is used sparingly. SKIPPER still retains a strong declarative nature. The rest of the rule base pattern-matches on the objects are of normal undeclared salience.

CONCLUSION

Successful software for an AUV must incorporate techniques from artificial intelligence, real-time processing, environmental sensing, and vehicle maneuverability into a compact integrated package. This is due to an AUV's lack of human control during mission execution and the inability for human intervention in the event of unforeseen problems. In addition, many tasks are knowledge-intensive and require domain-specific information. Therefore, the ability to include autonomous intelligent decision-making on an AUV is essential for its satisfactory performance. With the accumulated experience in submarine operation, we believe many of the onboard problem-solving and reasoning can be adequately modeled using a rule-based system. The Mission Executor is designed to (1) monitor relevant vehicle variables, component parameters, and environment data; (2) ensure the progress of pre-planned mission execution; and (3) in the event of unplanned interruptions during a mission, be able to diagnose the problematic situations and enable the vehicle to adapt to the unexpected environment by manipulating and changing vehicle and mission parameters.

A prototype for the Mission Executor has been completed and will be incorporated in the testbed as dependent modules are finished. The design is one that is extensible. Further, its object-oriented nature allows for incremental construction and testing of modules in relative isolation. The specific mission modules are areas for more fine-grained research. Because of the specialized nature of each of the mission modules, they are excellent areas for application of object-oriented tools like CLIPS 5. 0.

ACKNOWLEDGMENT

This paper was prepared in conjunction with research funded by the Naval Postgraduate School.

REFERENCES

Bellingham, J. G., T. R. Consi, R. M. Beaton and W. Hall (1990). Keeping Layered Control Simple, *Proceedings of IEEE Symposium on Autonomous Underwater Vehicle Technology*, Washington, DC, June 1990.

Blidberg, D. R., S. Chappell, J. Jalbert, R. Turner, G. Sedor, P. Eaton (1990). The EAVE AUV Program at the Marine Systems Engineering Laboratory, *Proceedings of the IARP Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October, 1990.

Brooks, R. A.(1986). A Layered Intelligent Control System for a Mobile Robot, in Gangeras and

Girald (eds), *Robotics Research*, MIT Press, Boston.

Cloutier, M. J. (1990). *Guidance and Control System for an Autonomous Vehicle*, M.S. Thesis, Naval Postgraduate School, June 1990.

Giarratano, J. C. (1991). *CLIPS User's Guide Volume 1: Rules, CLIPS Version 5.0*, NASA-Lyndon B. Johnson Space Center Information Systems Directorate Software Technology Branch, January 1991.

Giarratano, J. C. (1991a). *CLIPS User's Guide Volume 2: CLIPS Object Oriented Language*, NASA-Lyndon B. Johnson Space Center Information Systems Directorate Software Technology Branch, April 1991.

Healy, A. J. , R. B. McGhee, R. Cristi, F. A. Papoulias, S. H. Kwak, Y. Kanayama and Y. Lee (1990). *Proceedings of the IARP Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October, 1990.

Lee, Y., Luqi and R. B. McGhee (1991). Automating the Construction of Real-Time Software for an Autonomous Underwater Vehicle through Prototyping, *Proceedings of the 7th International Symposium on Unmanned Untethered Submersible Technology*, Durham, New Hampshire, 23-25 September 1991.

Mettrey, W. (1991). A Comparative Evaluation of Expert System Tools. *IEEE Computer*, Vol. 24, No. 2, February 1991, pp. 19-31.

Zheng, X., E. Jackson, and M. Kao (1990). Object-Oriented Software Architecture for Mission-Configurable Robots, *Proceedings of the IARP Workshop on Mobile Robots for Subsea Environments*, Monterey, California, October, 1990.

SESSION 2 B

N92-16574

THE AUTOMATED ARMY ROTC QUESTIONNAIRE (ARQ)

David L. H. Young

U.S. ARMY
HQ TRADOC BLDG 5 #G103
ATTN: ATRM-K
FT. MONROE VA 23561

Abstract. The Reserve Officer Training Corps Cadet Command (ROTCCC) takes applications for its officer training program from college students and Army enlisted personnel worldwide. Each applicant is required to complete a set of application forms prior to acceptance into the ROTC program . These forms are covered by several regulations that govern the eligibility of potential applicants and guide the applicant through the filling out forms . These forms and regulations are maintained by personnel at Army Education Centers, college ROTC departments and career placement offices. Because these individuals are not normally involved in ROTC admissions completion process, they are not thoroughly familiar with the filling out of forms or the eligibility criteria established by the regulations. Eligibility criteria changes as Army regulations are periodically revised. The distribution of revised regulations is a costly venture for ROTCCC. It results in missing or outdated regulations being maintained at the colleges and education centers. Outdated information results in a loss of applications attributable to frustration and error.

ROTCCC asked the Artificial Intelligence Center at Fort Monroe, for an inexpensive and reliable way of automating their application process. After reviewing the process, the Center determined that an expert system with good end-user interface capabilities could be used to solve a large part of the problem. The system captures the knowledge contained within the regulations, enables the quick distribution and implementation of eligibility criteria changes, and distributes the expertise of the admissions personnel at Cadet Command to the education centers and colleges worldwide.

The expert system uses a modified version of CLIPS that was streamlined to make the most efficient use of its capabilities. A user interface with windowing capabilities provides the applicant with simple and effective way to input his/her personal data.

DESIGN METHODOLOGY

The process of encoding the ROTC application form questions into CLIPS code was a straight-forward task. The true complexity occurred in trying to streamline CLIPS, the ROTC CLIPS code, and designing a compatible windowing interface. The design is based on an iteration process handling data based on the concept of a LIFO queue. The data is collected and analyzed with the appropriate results being displayed and the data on a potential cadet is saved.

PRECEDING PAGE ~~BLANK~~ MISSING NOT FILMED

The design steps are :

(a) stage one : understanding ROTC policy with the help of a domain expert;
(b) stage two : what was ROTC looking for in an automated program;
(c) stage three : designing prototype knowledge bases that demonstrate stage one and two are mastered;
(d) stage four : embedding the ROTC policy for GREEN TO GOLD, High School, and College Students into CLIPS code and testing with help of a domain expert;
(e) stage five : designing prototype C functions to handle problems that CLIPS cannot;
(f) stage six : using C functions to store collected information into a data file;
(g) stage seven : streamline C prototype functions and design a generic end-user interface for CLIPS;
(h) stage eight : modify ROTC CLIPS code to take advantage of CLIPS hieratical lookup table, the improved C functions, and the end-user interface commands;

The ROTC process is a vast and complex system of paper work. The ARQ was created to reduce this process. The ARQ does not cover all possibilities at this time. As the ARQ expands, it becomes more and more thorough.

This program is not meant to replace the current ROTC system already in place but to aid and reduce paperwork. None of the benefits listed by this program are a guarantee, as ROTC reserves the right of all final decisions.

ARQ SPECIFICATIONS

The ARQ was required to do the following :

(a) operate in a 640 k ram environment on a 286 or 386 IBM / Compatible machine.
(b) use a language easy to learn, update, and maintain a program.
(c) be efficient and thorough.
(d) save the information collected on a potential candidate.
(e) for the program to fit on one or two disk.

The design and structuring was the author 's.

DESIGN STRUCTURE OF ARQ

The design of the AUTOMATED ARMY ROTC QUESTIONNAIRE (ARQ) takes advantage of what CLIPS has to offer. Each feature of CLIPS was used where it was most beneficial to the program or saved memory. All C code written for the ARQ does functions that CLIPS does not provide.

The Rules are structured to assert new facts or carry out specific commands. Ninety percent of all printout statements have been removed from the rules. This allowed for smaller rules and more rules could be added and processed.

With a few exceptions, all statements, questions, and instructions are kept in a text file accessed by the hieratical lookup table. Exceptions are :

(a) statements found in the code.
(b) one line statements printed in color to attract the user's attention.

Information collected on a user is stored as facts. Select facts shall be used to "fire" other rules. Each program iteration stores the user's information to a DBASE file for future reference by ARMY ROTC.

Program templates provide quick and direct checking or modifying a stored information. One template retains the scores for select data comparison. The programmer can conveniently modify the scores list when test values change without changing the program or a particular rule. Some lists are not templates, and are stored as facts to save on memory. They are only traversed for particular value.

The Facts architectural design is modeled after the OOD (OBJECT ORIENTED DESIGN) principle to allow facts to integrate with future object oriented versions of CLIPS. Some facts do not follow this principle. They consist of one or two straight-forward words and control switching between processing phases.

The end-user interface is a popup windowing package written in Microsoft C 5.1 (a TURBO C compatible version is under development.) The windows for ARQ are basic and limited. (More advance window functions are under development.) The popup windows and menus displaying information to the screen allow for a friendly interface between the user and the computer. Window functions are called from CLIPS and do not consume much memory. Due to the time limit and all the possible combinations of facts, rules, and window functions, the actual number of windows open at one time is currently unknown. The program has ninety-two rules, the facts vary in number, and a maximum of four windows are opened at once.

There are other C functions for interfacing and are needed to run the ARQ.

The ARQ screens three different ROTC candidate profiles:

(a) The GREEN TO GOLD candidate (active duty enlisted soldier.)
(b) The On-Campus College candidate.
(c) The college bound High School candidate.

There are general questions which apply to all the prospects which are followed by three different sets of questions which were tailored to address the unique characteristics of a given profile. There is a rule for each profile which asserts a group of facts spontaneously. There is a question protocol. The facts channel relevant and related rules into proper firing sequence.

ARQ operates in a modified CLIPS 4.3. ARQ can be upgraded to run with CLIPS 5.0 assuming no memory limitations occur. All C code will have to be modified. The CLIPS Ver. 4.3 is the version of choice. Version 5.0 in a 640k ram environment requires further testing.

THE ARQ OPERATION

The ARQ asks the user questions by using popup windows and menus to obtain information. The questions asked, pertain only to the users. The number of questions vary depending on the user's responses. Each response is used for firing new rules or is stored as a fact. As the user answers the questions, informative statements will appear in a popup window. When the user finishes the questionnaire, input (data) is gathered into one template. The template is then written as a record in a DBASE III file. ARQ then prompts the user to run again.

If for any reason the program is exited before completion all data collected is destroyed upon exiting. No information will be saved to the DBASE III file.

BENEFITS OF USING THE ARQ

The ARQ allows users to quickly determine eligibility for the ROTC program. This makes the process less confusing. It provides the ARMY with a reliable upto date record of all individuals who are eligible and interested in ROTC. ARMY ROTC can follow up and refine the leads provided by ARQ.

The simplicity of ARQ means that anybody can use it without instructions.

ARQ serves as a tutorial to teach, inform, and update ROTC education and guidance counselors.

Common "What if" scenarios now have solutions. Estimating or second guessing is not necessary.

FUTURE GOALS

The next version should contain additional profiles thus making ARQ more thorough. The rules should be divided into single or like candidate files and a main rule file used for running the ARQ. The main file would give a menu selection of all the candidate types, select, and load in the corresponding candidate rule file and run.

A more advance windowing package built for CLIPS with graphics pictures should tempt more users to try the ARQ. The graphics should enhance the program and make the ARQ more palatable. Advanced window functions will make CLIPS more user friendly.

The ability to provide multiple drive access is needed for writing output to disk storage or for files located in different drives and directories.

The incorporation of the ROTC LOGO would give the ARQ a more authentic feel to the user.

Long term goals are to send and maintain the ARQ electronically world wide and to upload and down-load the ARQ on a main-frame.

The long term goals will involve a substantial investment in technical and manpower resources.

SUMMARY

ARQ is a versatile and portable tool. It has the flexibility to keep expanding and still be extremely efficient and reliable.

The CLIPS language is easy to learn and makes it easy to update the ARQ.

Since the ARQ reduces paperwork, the ROTC application process can execute faster and smoother.

N92-16575

DECISION BLOCKS: A TOOL FOR AUTOMATING DECISION MAKING IN CLIPS

Christoph F. Eick
Department of Computer Science
University of Houston
Houston, TX 77204-3475
e-mail: ceick@cs.uh.edu

Nikhil N. Mehta
GE Government Services
1050 Bay Area Boulevard
Houston, TX 77058
e-mail: nmehta@146.154.10.168

Abstract. The human capability of making complex decision is one of the most fascinating facets of human intelligence, especially if vague, judgmental, default or uncertain knowledge is involved. Unfortunately, most existing rule-based forward-chaining languages are not very suitable to simulate this aspect of human intelligence, because of their lack of support for approximate reasoning techniques needed for this task, and due to the lack of specific constructs to facilitate the coding of frequently reoccurring activities in decision making processes. The paper advocates to extend CLIPS by a new component called decision block to provide better support for the design and implementation of rule-based decision support systems. A language called BIRBAL†, which is defined on the top of CLIPS, for the specification of decision blocks is introduced. Empirical experiments involving the comparison of the length of CLIPS-program with the corresponding BIRBAL-program for three different applications are surveyed. The results of these experiments suggest that for decision making intensive applications a CLIPS-program tends to be about three times longer than the corresponding BIRBAL-program.

I. INTRODUCTION

Very often in human life we are forced to make guesses in order to decide where certain objects are located, for reconstructing events that happened in the past, or for building plans in order to achieve a certain goal. The human capability of making complex decisions is one of the most fascinating facets of human intelligence. Usually, decision making involves multiple knowledge sources from which the expert extracts different clues by frequently using a set of fuzzy rules, which encode the expert's general and domain-specific knowledge. Finally, the expert comes up with a decision by combining the evidence received from the individual clues.

The problem of how to design and implement larger systems that rely on these approaches has widely been ignored by current research (some exceptions to this point will be discussed below). There is a lack of programming languages that integrate these approaches of the rule-based expert system shells that support reasoning involving imperfect knowledge adequately, and of knowledge engineering methodologies that can cope with large amounts of imperfect knowledge. Programming involving imperfect knowledge, will be referred to as fuzzy programming in the following sections, still seems to be quite far away for commercial applications. Experimental

† Birbal was a famous minister in the 16th century India, who served as an advisor to king Akbar.

systems that do support reasoning involving imperfect knowledge such as PROSPECTOR([14]), or MYCIN([10]), mostly use "single-valued approaches" for reasoning under uncertainty; that is, they assign a probability-like value to each predicate of interest. However, single-valued approaches have problems coping with ignorance and with different degrees of reliability in different knowledge sources. Two-valued approaches that intend to overcome these problems have been advocated in the literature; most of these approaches use Dempster/Shafer's theory of evidence as the underlying knowledge representation framework ([1], [24], [28]) or they assign priorities to rules and use these priorities in a pragmatic way when combining evidence [36]. These approaches are capable of assigning --- in addition to probabilities --- reliabilities to predicates and rules and have no difficulties with representing ignorance, which makes them very attractive for automating human decision making in uncertain environments. The RUM-system([7]) advocates the use of a 3-layered reasoning strategy, which distinguishes between representation, inference, and control, which is defined on the top of a two-valued approach. Two other experimental systems, INFERNO([27]) and ARIES ([1]), that rely on two-valued approaches have been described in the literature. Finally, some efforts have been made to integrate approximate reasoning methods with Prolog ([4], [21]).

The main topic of this paper is the discussion of techniques and concepts that facilitate the implementation of rule-based decision support systems that have to cope with imperfect knowledge. A language construct, called decision block, that facilitates the automation of decision making is proposed, and its features and its integration with rule-based forward chaining languages are discussed in some detail. The paper is organized as follows: Section 2 introduces decision blocks and a language BIRBAL that supports decision blocks. Section 3 discusses the implementation of BIRBAL.

II. DECISION BLOCKS

In this section we are going to provide the programmer with a language construct, called decision block. This construct eases the automation of the more general aspects of decision making by reducing the number of rules as well as the complexity of individual rules. Decision Blocks rely on the technique called decision making by evidence combination (DBE) that can be characterized as:

(1) Rules in a rule-set are assumed to be independent, providing positive or negative evidence for or against making a certain decision in a certain situation. Rules approximate the basic principles of a particular domain.
(2) Smooth decision making is supported. If the left hand side (LHS) of a rule is only partially true the amount of evidence provided by the rule will be decreased.
(3) After the rules of a rule-set have been processed completely, the evidence provided for different decisions is combined and the best decision is selected. That is, a two-layered inference strategy is used that separates decision making from decision execution.

When using this approach, no artificial dependencies between rules need to be introduced, and errors in a rule-set can be more easily detected, because each rule-set returns a ranking of the available decisions and no longer only the chosen decision. Using the above approach rule-sets that encode decision making processes look as follows:

(R_1 (if A) (then provide evidence for D_1 with amount a1))
(R_2 (if C) (then provide evidence for D_1 with amount a2))
(R_3 (if B) (then provide evidence for D_2 with amount a3))
(R_4 (if "true") (then provide evidence for D_3 with amount a4))

In general, assuming that the decision with the highest amount of positive evidence is chosen, a particular selection a1,...,a4 is correct, as long as it satisfies the following equations:

$$a1 \oplus a2 > a3$$
$$a3 > a1,\ a3 > a2$$
$$a1 > a4,\ a2 > a4$$

The symbol $\oplus$ refers to the operator that combines evidence received from different rules in the context of the underlying method for approximate reasoning. For example, the first equation expresses that the result of combining the amount of evidence a1 and a2 has to be greater than the amount a3, reflecting that if both A and C are present D1 and not D2 should be selected. Or, to give another example, a3 has to be greater than a2, because D2 is preferred if B and C, but not A are observed.

So far, our discussion abstracted from the underlying approach for approximate reasoning. We consider a method M to automate approximate reasoning to be a pair M=(O, T) that consists of a set of Operators O operating on the type T. In order to be suitable for decision making by evidence combination, we require that O provides at least the following operators:

$\wedge$	$\in$	$T \times T$	$\rightarrow$ T is called the and-operator
$\vee$	$\in$	$T \times T$	$\rightarrow$ T is called the or-operator
not	$\in$	T	$\rightarrow$ T is called the not-operator
$\longrightarrow$	$\in$	$T \times T$	$\rightarrow$ T is called the modus-ponens-operator
$\oplus$	$\in$	$T \times T$	$\rightarrow$ T is called the combination-operator
$\subseteq$	$\in$	$T \times T$	$\rightarrow$ BOOLEAN. Furthermore, $\subseteq$ has to be an order relation: it has to be reflexive, anti-symmetrical, and transitive.

T represents the type used by the underlying methodology to measure the truth of imperfect knowledge. In a Bayesian system T would be set to [0 1], in the case of MYCIN certainty factors T would be [-1 1], and in the case of two-valued interval approach T would be:

$\{(x,y) \in \Re \mid 0 \leq x \leq y \leq 1\}$; where $\Re$ is a set of real numbers.

In general, various abstract data-types have been proposed in the literature, e.g. ([1], [6], [7], [10], [27]), that are suitable for decision making by evidence combination. In the following we assume that the above operators are applied in infix-form. For example, if the following rule R

(R (if (A and (B or not (C))) then (provide evidence for D amount x))

is processed, its amount of evidence for D would be computed as follows:

$(a \wedge (b \vee (not(c))) \longrightarrow x$

in which $a, b, c, x \in$ T, describes the belief associated with A, B, C, and with the rule R respectively. Or, in our original example D2 is considered to be better than D1 if:

$$(a \longrightarrow a1) \oplus (c \longrightarrow a2) \subseteq (b \longrightarrow a3)$$

In summary, when automating decision making relying on the above framework, we will first apply the operators to process the LHS of a rule, then we will use the operator $\longrightarrow$ to compute the evidence provided by a particular rule, and then we will combine the evidence inferred by different rules for the same predicate using the operator $\oplus$ for each decision candidate. Finally, the order relation $\subseteq$ is used to select the best decision candidate(s).

Decision blocks enable programmers of decision support systems to identify independent units of decision making. More specifically, a decision block consists of (see Figure 1):

Decision Block

Environment	Decision Candidates	Decision Making Policy	Rule-Set
	Actions		

Figure 1. Components of a Decision Block

(1) A *decision specification* that enumerates decision candidates from which the best decision(s) has (have) to be selected, and which associates an *action* with each decision candidate which will be executed in the case that a particular decision is chosen.
(2) An *environment* which consists of a set of context variables that describe the context in which a particular decision has to be made.
(3) A *rule-set*, whose members provide positive and/or negative evidence for or against the particular decision candidate. Rule-sets are further subdivided into disjoint *ruleclusters* that represent rules that are related to the same knowledge source. Evidence provided by rules from belonging to different ruleclusters is considered to be independent evidence, whereas rules belonging to the same rulecluster are assumed to provide dependent evidence. Furthermore, *exceptions* can be associated with rules. Exceptions are represented by augmenting production rules by an exception LHS and an alternate right hand side (RHS). In the case that the rule's LHS and the exception's LHS are satisfied, the corresponding alternate RHS is executed.
(4) A *decision making policy* that encodes the decision procedure for selecting a decision or a set of decisions after all clues relevant for the decision making have been analyzed.

Phase 1	Phase 2	Phase 3	Phase 4
Evidence Generation	Evidence Combination	Decision Selection	Decision Execution
Rules and their associated exceptions are evaluated giving preference to the most specific exception.	Evidence received from different rules is combined.	A set of decisions is selected depending on the decision making policy.	The selected decisions are carried out by executing their associated actions.

Figure 2. Phases of a Decision Block

Decision blocks automate rule-based decision making by using a four layered inference strategy, as depicted in Figure 2. First, rules and their associated exceptions are evaluated, giving priority to the most specific exception which is applicable for the rule. Second, dependent evidence within each rulecluster, and independent evidence associated with different ruleclusters is combined. For combining dependent evidence a second evidence combination function was provided, which is given in Appendix 1, and whose properties are discussed in more detail in ([15]). Third, a set of decisions is selected according to the strategy outlined in the decision block's policy. Decision making policies supported by decision blocks, which are evaluated with respect to the operator $\subseteq$ include: select the best decision, select the best n decisions, select all decisions, or the best n decisions better than a threshold value α $(0 \leq \alpha \leq 1)$. Fourth, the selected decisions are carried out, executing the actions associated with the selected decisions.

Figure 3. gives an example (for detailed explanation of different examples see [25]) of a typical decision block, named scholarships, which encodes the awarding of scholarships to the students on the basis of their academic performance, experience as teaching assistant and age.

```
(fuzzy functions
   (curr-perform (arity 1))
   (overall-perform (arity 1))
   (exp-as-TA (arity 1))
   (financial-need (arity 1))
   (oldness (arity 1)))
(decision-block scholarships
   (environment (?univ))
   (action-specifications
    (qualification
       ((qual-of ?ssno ?univ) with (?lb ?ub))
       =>
       (printout t "Qualified Student is " ?ssno ?univ)))
  (decision making policy (select ALL decisions with mv > 0.8))
  (rule-set criteria
    (sc1
       (student (ssno ?ssno) (univ ?univ) (cgpa ?cgpa) (ogpa ?ogpa))
       (%curr-perform ?cgpa)
       (%overall-perform ?ogpa)
       (test (and (between ?cgpa 0.0 4.0) (between ?ogpa 0.0 4.0)))
       =>
      (infer (qual-of ?ssno ?univ) with (0.9 1.0)))
   (sc2
      (student (ssno ?ssno) (univ ?univ) (expTA ?expTA))
      (%exp-as-TA ?expTA)
      =>
      (infer (qual-of ?ssno ?univ) with (0.8 1.0)))
   (sc3
      (student (ssno ?ssno) (univ ?univ) (fin-need ?fneed))
      (%financial-need ?fneed)
      =>
     (infer (qual-of ?ssno ?univ) with (0.75 1.0)))
  (sc4
     (student (ssno ?ssno) (univ ?univ) (age ?age))
     (%oldness ?age)
     =>
     (infer (qual-of ?ssno ?univ) with (0.65 1.0)))))
```

Figure 3. Example of a Decision Block: scholarships

Before a decision block can be used, all fuzzy functions (that is, functions that return intervals as results) with it's arity (i.e., number of arguments) have to be identified. The action-specifications of the decision block enumerates the decision candidates, which are represented as LHS patterns

((qual-of ?ssno ?univ) with (?lb ?ub)),

and RHS gives the action(s) which has(have) to be performed, if the corresponding decision is selected. For example, if a student from the university of Houston with ?ssno equal to 123456789 is selected depending upon the decision making policy then "Qualified Student is 123456789 Houston" is printed out. The environment definition defines the context variables used by the decision block. These variables have to be initialized by the call of the decision-block. These variables are local to the decision-block and need to be passed (in this case ?univ) whenever the decision-block is invoked. The decision making policy specifies what decisions need to be selected from the set of decisions taken, here the decision making policy applied is specified as

"(select ALL decisions with mv > 0.8)"

i.e. the combined evidence received from each of the rules from the rule-set should have the mean-value greater than 0.8.

Now let's see what each rule in the rule-set expresses. First we will see intuitively what is being signified by intervals specified (interested reader should refer to the Appendix 1) with the Two-valued approach we have taken. We can express the fact that we know nothing about certain proposition P by assigning interval [0 1], i.e., unknown can be directly expressed in the interval approach. Also the usage of classical probability becomes the special case of Two-valued approach because we can express single probability values by assigning same lower and upper bounds, e.g., [0.4 0.4], where uncertainty is 0 and reliability is 1, and we are sure about the proposition. Intuitively speaking if the difference between the lower and upper bound increases the uncertainty of the proposition increases and if the difference between them decreases certainty about the proposition increases. Also the evidence provided for the proposition is considered as negative if the mean-value of the interval is less than 0.5.

The LHS of any rule could be any valid CLIPS LHS pattern plus the fuzzy function(s) (C functions), if any, identified by '%' followed by the arguments of the functions. Let's take the first rule sc1 which provides the evidence towards the qualification of the student depending on the academic performance of the student, i.e., student's current GPA and overall GPA. The performance of the student depending on GPA is difficult to quantify since the student having overall GPA of 3.75 and other having 3.80 lies in the same category. Also same can be said for current GPA , but the combined performance might differ extensively since the current GPA will affect the overall GPA drastically depending upon the value. Hence the combined performance is fuzzy. The infer statement in the first rule suggests that it has higher contributing power towards the final decision since the interval applied to it is [0.9 1.0]. The second rule sc2 will contribute towards the final decision depending upon the experience as teaching assistant but the rule has lower contributing power than the first one since the interval applied to it is [0.8 1.0]. Similarly other rules will contribute towards the final decision.

The evidence provided by the LHS fuzzy predicates such as performance, experience, financial-need and age will be used in computations using logical connectives ($\wedge$, $\vee$, not) which in our case is $\wedge$, and modus ponen functions ($\longrightarrow$) and new evidence interval will be computed. This new value will be contributed by each rule towards the final decision, i.e., qualification of the student for receiving a scholarship. Each rule can either provide positive or negative evidence. Next the evidence provided by each rule is combined repeatedly using the operator $\oplus$ until the evidence provided by each rule towards the final decision is combined. In the third phase depending upon the decision making policy the decision regarding the qualification is made from the set of competing candidates and in the last phase the action specified by action-specifications is taken, in our case it's simply printing out all the candidates whose mean-value of the interval is greater than 0.8. In general the action-specifications can be used to sort the candidates in the ascending order of mean-values or could involve further computations for the final selection or could invoke another decision-block or simply prints the first 10 candidates called for the interview putting other candidates on the waiting list, etc. We can see from the previous discussion how smooth the entire process is and how simple it is to code the decision-block. We will see in the next section implementation of the decision-block, that if we need to do all that is specified in the previous discussion using pure CLIPS how much extensive coding is required and burdensome for the programmer which is not the case with the decision block since the compiler takes away that burden from the programmer.

III. IMPLEMENTATION OF DECISION BLOCKS

In this section, we will discuss the implementation of a rule-based language that supports decision blocks, and will report on some empirical results concerning the benefits of decision blocks. More specifically, we will report on the integration of decision blocks into a rule-based forward chaining language called CLIPS([19]), which was developed by NASA. The extended language is called BIRBAL. A BIRBAL-programmer can in addition to CLIPS rules define decision blocks and call these decision blocks within a CLIPS-program.

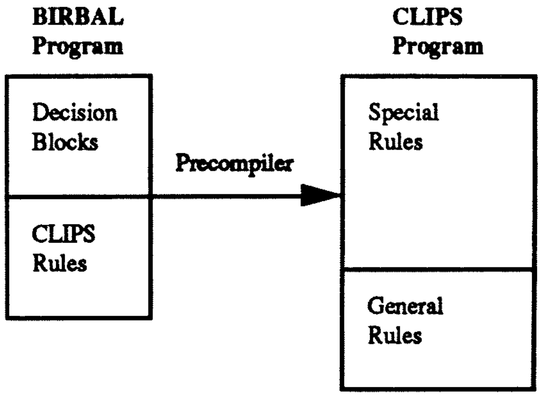


Figure 4. Implementation of Decision-Blocks

A precompiler has been provided, as a part of a Master's thesis ([25]), that maps a BIRBAL-program that consists of decision blocks and CLIPS rules into a program that uses pure CLIPS (see Figure 4). The precompiler is written in C has about 1400 lines of symbolic code. It was developed using the UNIX-compiler generating tools *lex* and *yacc*. The developed precompiler relies on a rule-mapping that maps BIRBAL-rules into CLIPS rules; that is, every rule defined within a decision block is transformed into a single CLIPS-rule that simulates its behavior. However, the generated CLIPS rules are much larger in size, which can be attributed to:

- CLIPS which does not support reasoning under uncertainty. This means that computations involving operators for approximate reasoning have to be provided manually for each rule.
- CLIPS does not support context variables, which implies that they have to be stored as assertions in the CLIPS working memory. The variable management is further complicated by the fact that there could be multiple active decision blocks, which use equally named variables.
- Exceptions are not supported; therefore they have to be programmed out by the programmer.
- It has to be made sure that evidence providing rules are only fired in phase 1 of the execution of a decision block; or, in more general terms, the context under which evidence-providing rules are allowed to fire has to be expressed by adding additional LHS conditions to the generated CLIPS-rule.

In order to make the above comments more transparent and to give the reader a better feeling what a programmer has to do if no decision blocks are provided, let us see how our precompiler maps the rule (refer to Figure 5).

In CLIPS's syntax a rule's LHS and RHS are separated by => symbol, and all condition elements that appear in a rule's LHS are assumed to be connected by 'and'. Furthermore, in CLIPS match-variables are prefixed by '?'. The above rule starts with two control assertions (active-db_ scholarships) and (status_ scholarships phase 1) that make sure that the generated rule is only fired when the decision block scholarships is active and in phase 1. The rule sc1 refers to one context-variable. Context variables are represented in the CLIPS environment by assertions of the form (var_ <db-name> <var-name> <var-value>); e.g., (var_ scholarships univ Houston), expresses that variable 'univ' of decision block scholarships has the value Houston. The values of the context variables have to be retrieved by adding one LHS condition to the rule. Furthermore, the LHS conditions of the BIRBAL-rules have to be simulated. Non fuzzy-conditions do not pose any particular problem; they are simply copied into the generated CLIPS-rule. In this particular case, it is the test pattern which checks whether the GPA of the student is within the valid range or not. The implementation of the fuzzy conditions is much more complicated. First we assume that the C-functions that implement the definition of curr-perform and overall-perform has been provided by the user.

```
(sc1
   (student (ssno ?ssno) (univ ?univ) (cgpa ?cgpa) (ogpa ?ogpa))
   (%curr-perform ?cgpa)
   (%overall-perform ?ogpa)
   (test (and (between ?cgpa 0.0 4.0) (between ?ogpa 0.0 4.0)))
   =>
   (infer (qual-of ?ssno ?univ) with (0.9 1.0)))
The above rule sc1 is transformed into the following CLIPS-rule:
(defrule sc1-phase1
   (declare (salience 555))
   (active-db_ ?db-name_&scholarships)
   (status_ ?db-name_ phase ?phase_&1)
   (var_ ?db-name_ univ ?univ)
   (student (ssno ?ssno) (univ ?univ) (cgpa ?cgpa) (ogpa ?ogpa))
   (test (and (between ?cgpa 0.0 4.0) (between ?ogpa 0.0 4.0)))
   =>
   (bind ?l1-u1_ (curr-perform ?cgpa))
   (bind ?l1_ (nth 1 (str-explode ?l1-u1_)))
   (bind ?u1_ (nth 2 (str-explode ?l1-u1_)))
   (bind ?l2-u2_ (overall-perform ?ogpa))
   (bind ?l2_ (nth 1 (str-explode ?l2-u2_)))
   (bind ?u2_ (nth 2 (str-explode ?l2-u2_)))
   (bind ?y1_ (* ?l1_ ?l2_))
   (bind ?y2_ (* ?u1_ ?u2_))
   (if (and (!= ?y1_ 0) (>= (+ ?y1_ 0.9) 1) (>= (+ ?y2_ 1.0) 1)) then
      (bind ?lb_ (/ (- (+ 0.9 ?y1_) 1) ?y1_))
      (bind ?ub_ (/ (- (+ 1.0 ?y2_) 1) ?y2_))
      (assert (sc1 ?db-name_ 2 qual-of ?ssno ?univ with ?lb_ ?ub_))))
```

Figure 5. Rule Transformation from Decision Block rule to CLIPS rule.

Each of these functions returns an interval, measuring the performance of the student for which it were called. Second, the approximate reasoning methods described in section II, have to be used to compute the rule's evidence for the qualification of the students. In this particular case two fuzzy conditions are used, which means that the belief in the rule's LHS can be equated to the academic performance of a particular student, if the test conditions for GPAs are satisfied. The first eight RHS actions call the C-functions curr-perform and overall-perform, bind the returned intervals, and bind the lower and upper bounds to variables ?y1_ and ?y2_ after applying logical connectives (see Appendix 1) to the returned intervals. The rest of the rule's RHS simulates the operator '-→' and computes the rule's amount of evidence. In case that the amount of positive evidence is [0 1] then the rule does nothing; otherwise it asserts an assertion specifying the amount of evidence provided for the decision candidate. For example, if the rule sc1 provides a positive evidence of [0.75 1.0], the following pattern

"(sc1 scholarships 2 qual-of 123456789 Houston with 0.75 1.0) "

would be asserted. These assertions are used later in phase 2 to combine the different pieces of evidence associated with the same student. It is also important to note that the generated code would become much more complicated in case that the LHS of the rule references more than two fuzzy conditions connected by different logical connectives and with the presence of exception conditions (see [25]). In that case, it would be necessary to add code that simulates the combination of $\vee$, $\wedge$, and *not* operators.

In general, the example demonstrates that implementing the approximate reasoning methods by hand is highly complicated and time consuming, if no methods for approximate reasoning have been integrated into a rule-based language. Even worse, these computations have to be provided for every rule of a rule-set, even if the computations are similar or the same.

Moreover, the precompiler transforms a decision block's decision making policy into a rule that generates assertions, parameterizing the computations of general rules that simulate the selected decision making policy. Also the decision blocks decision specification is transformed into rules, whose right hand sides consists of the actions associated with a decision candidate, whereas the LHS of the generated rule checks for assertions that state that the corresponding decision has been selected by the decision making policy. Additionally to the rules that are generated specifically for a particular decision block, our implementation relies on a set of general rules that perform general tasks such as evidence combination, switching between the phases of the implementation of a decision block, management of calls of decision blocks, and removal of trash (see Figure 6).

We also made some experiments comparing the length of a decision block with the length of the generated CLIPS program that implements the decision block. These experiments included a slightly more complex version of the scholarships decision block, the decision blocks that simulate the assignment of scholarships at a university, and a larger decision block for a problem of medical parasitology. The results are summarized in Table 1. We claim that these numbers give a good indication of the benefits of decision blocks with respect to a programmer's productivity in decision making intensive applications, such as those analyzed in the benchmark. A pair (a, b) in the Table 1 indicates that the corresponding program has 'a' lines and 'b' characters.

In general, the experiment suggests that the generated CLIPS program is about 4 times longer than the corresponding decision block if DBE is implemented, which makes it quite apparent why the conventional approach has been used quite frequently in practice, and not decision making by evidence combination (DBE). Due to the lack of supportive constructs in commercial forward-chaining languages the received programs tend to be very long, if DBE is used. However, if higher order constructs such as decision blocks are integrated into a rule-based languages, this remark is no longer true. We even go further and claim --- that programs developed using decision blocks tend to be significantly shorter than programs that were developed using the conventional approach. Moreover, the development of a precompiler was not very complex, which demonstrates that decision blocks can be provided at a low cost on the top of rule-based languages such as CLIPS([19]), OPS([18]), or ART([2]), and that decision blocks can be easily integrated into forward chaining systems. In summary, the availability of decision blocks or similar constructs is an important prerequisite to develop decision support systems at reasonable cost that rely on decision making by evidence combination.

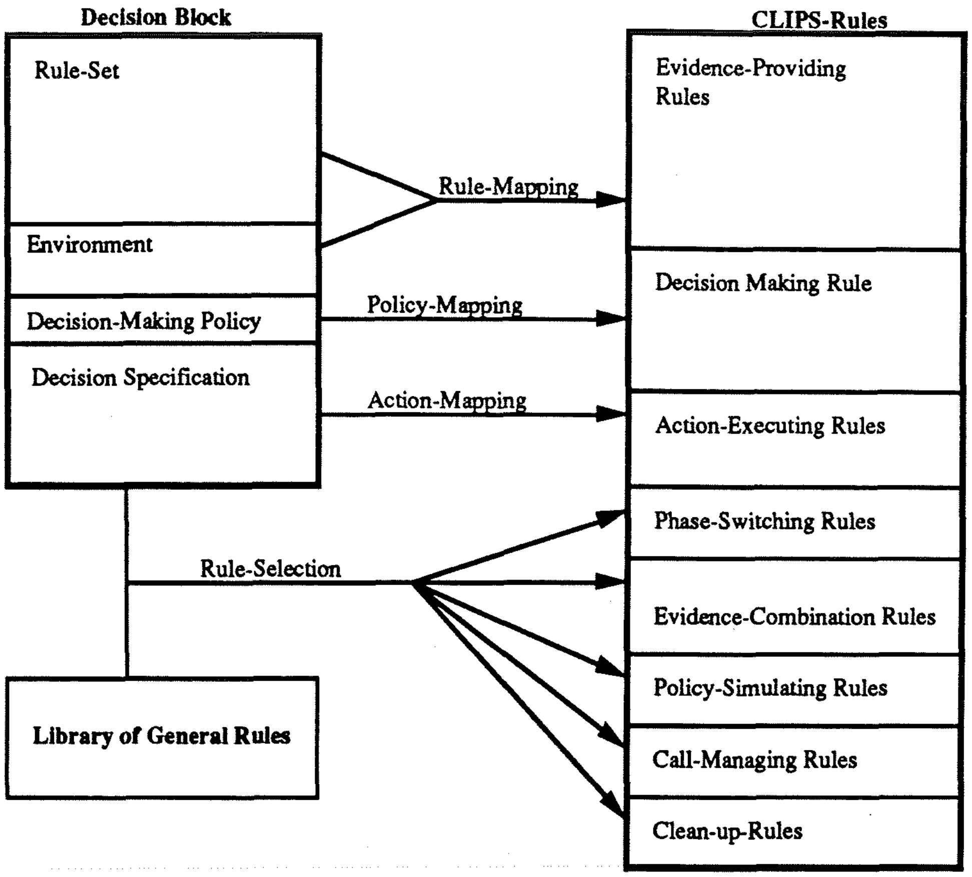


Figure 6. Mapping Decision Blocks to a Rule-Based Language

Example / Language	Opening-bid in the game of Bridge	Medical Parasitology	Scholarship Awarding
BIRBAL	(45, 1175)	(167, 5234)	(37, 1062)
CLIPS	(174, 5743)	(701, 22486)	(180, 4720)

Table 1. Comparison of the Complexity of CLIPS and BIRBAL

IV. CONCLUSION

This paper studied the problem of designing and implementing complex decision support systems. Computerized decision making that treats decision making as a problem of evidence combination was introduced. A language construct called decision block that facilitates the implementation has been proposed. A language called BIRBAL that integrates decision blocks with CLIPS has been provided. Decision blocks offers several advantages over classical rule-based languages such as ART, CLIPS and OPS for automating complex decision making processes. First, the availability of exception handling facilities and evidence combination techniques enables one to program close to the expert level, which is very important for explaining the system's behavior and for system validation. Second, our approach supports smooth decision making. Minor changes will only slightly affect the final ranking of the decision candidates. Third, decision blocks provide encapsulation and allow to modularize complex decision making processes. Fourth, we claim that decision blocks increase a programmer's productivity significantly because the precompiler takes care of many tasks such as reasoning under uncertainty, combination of evidence, removal of trash, ranking of decision candidates, or execution of decisions, which no longer have to be coded by a BIRBAL-programmer. Finally, we showed that decision blocks can be provided at a low cost on the top of existing rule-based programming languages, such as CLIPS.

REFERENCES

[1] L. Appelbaum, and E.Ruspini, "ARIES An Approximate Reasoning Inference Engine," in *[18]*, pp. 745-765.

[2] Inference Corporation, *ART Reference Manual.* Los Angeles Inference Corporation, 1986.

[3] J.F. Baldwin, and N. Gould, "Feasible Algorithms for Approximate Reasoning using Fuzzy Logic," *Fuzzy Set Systems*, vol. 3, pp. 225-251, 1980.

[4] J.F. Baldwin, "Evidential Support Logic Programming," *Fuzzy Sets and Systems*, Vol. 24, pp. 1-26, 1987.

[5] V. Barker, and D. O'Connor, "Expert Systems for Configuration at Digital XCON and Beyond," *CACM*, Vol. 32, No. 3, pp. 298-318, 1989.

[6] A. Basu, and A. Dutta, "Reasoning with Imprecise Knowledge to Enhance Intelligent Decision Support," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 19, No. 4, pp.756-770, 1989.

[7] P. Bonissone, S. Gans, and K. Decker, "RUM A Layered Architecture for Reasoning in Uncertainty," in *Proc. 10th IJCAI-conference*, Milan, pp.891-898, 1987.

[8] P. Bonissone, D. Cyrluk, J. Goodwin, and J.Stillman, "Uncertainty and Incompleteness Breaking the Symmetry of Defeasible Reasoning" in *5th Workshop on Uncertainty in AI*, pp. 34-45, Detroit, 1989.

[9] L.Browston, R. Farrell, E. Kant, and N.Martin, *Programming Expert Systems in OPS5*, Reading Addison-Wesley, 1985.

[10] B. Buchanan, and E. Shortliffe, *Rule-Based Expert Systems*, Reading Addison Wesley, 1984.

[11] P. Cheeseman, "Probabilistic versus Fuzzy Reasoning," in L. Kanal, *Uncertainty in AI*, Amsterdam North Holland, pp. 85-102, 1986.

[12] P. Cohen, *Heuristic Reasoning about Uncertainty An Artificial Intelligence Approach*, New York Pitman Publishing Limited, 1985.

[13] A.P. Dempster, "A Generalization of Bayesian Inference," *Jour. Royal Stat. Soc.*, B, V 30, pp. 205-247, 1968.

[14] R. Duda, J. Gaschnig, and P. Hart, "Model Design in the PROSPECTOR Consultant System for Mineral Exploration," in Michie's *Expert Systems in the Micro Electronic Age*, Edinburgh University Press, 1979.

[15] C.F. Eick, "Uncertainty Management for Fuzzy Decision Support Systems," in *Proc. 4th Workshop on Uncertainty in AI*, St.Paul, August 1988, pp. 98-108.

[16] C.F. Eick et al., "Computer Bridge - A Challenge for AI," in Z. Ras (eds.), *Methodologies for Intelligent Systems, 5*. New York North-Holland, pp. 59-67, 1990.

[17] C.F. Eick, Yao and H. Fu, "More Flexible Use of Variables in Rule-Based Programming," in *Proc. 2nd Int. Symposium on Artificial Intelligence*, Monterrey, Oct. 1989.

[18] C. Forgy, "OPS83 Report," Technical Report, Dept. of Computer Science, Carnegie-Mellon University, 1983.

[19] J. Giarratano, and G. Riley, *Expert Systems Principles and Programming*, Boston PWS-Kent Pub. Co., 1989.

[20] M. Gupta, A. Kandel, W. Brandler, and J. Kiszka, *Approximate Reasoning in Expert Systems*, Amsterdam, North Holland, 1985.

[21] C.J. Hinde, "Fuzzy Prolog," *Int. Journal of Man-Machine Studies*, pp. 569-595, 1986.

[22] A. Kandel, *Fuzzy Mathematical Techniques with Applications,* Read. Addison Wesley, 1986.
[23] R. Loui, "Evidential Reasoning in a Network Usage Prediction Testbed," *Proc. 4th Workshop on Uncertainty in AI,* St. Paul, 1988, pp. 257-265.
[24] S. Lu, and H. Stephanou, "A set-theoretical framework for the processing of uncertain information," in *Proc. AAAI-conference,* Austin, 1984, pp. 216-221.
[25] N.N. Mehta, "BIRBAL - A Rule-Based Language for Decision Making," Master's Thesis, University of Houston, December 1990.
[26] J. Prugh, "A Knowledge-Based Approach to Bridge Defense," Master's Thesis, University of Houston, May 1989.
[27] J. Quinlan, "INFERNO - A Cautious Approach to Uncertain Inference," *Computer Jour.*, Vol. 26, pp. 255-266, 1983.
[28] G. Shafer, *A Mathematical Theory of Evidence.* Princeton University Press, 1976.
[29] N. Shirouzu, *Norihiko Time for Some Fuzzy Thinking,* in TIME September 25, 1989, pp. 79.
[30] E.Soloway, J.Bachant, and K. Jensen, "Accessing the Maintainability of XCON-in-RIME Coping with Problems of a VERY Large Rule-Base," in *Proc. 6th National Conf. on Artificial Intelligence,* Seattle, 1987, pp. 824-829.
[31] H. Stephanou, and A. Sage, "Perspectives on Imperfect Information Processing," *IEEE Trans. on Systems, Man, and Cybernetics,* Vol. 17, no. 5, pp. 780-798, Sept. 1987.
[32] D. Touretzky, *The Mathematics of Inheritance Systems,* Los Altos M. Kaufman Pub., 1986.
[33] E. Trillas, and L. Valverde, "On Mode and Implication in Approximate Reasoning," in *[18],* pp. 157-166.
[34] C. Tsen, "A Knowledge-Based Approach to Bridge Bidding," Master's Thesis, University of Houston, June 1988.
[35] D. Vaughan, B. Perrin, R. Yadrick, and P. Holden, "Comparing Expert Systems Built using Different Uncertain Inference Systems," in *Proc. Fifth Workshop on Uncertainty in AI,* Detroit, August 1989, pp. 369-376.
[36] R. Yager, "Nonmonotonic Reasoning via Possibility Theory," in *Proc. 4th Workshop on Uncertainty in AI,* St.Paul, 1988, pp. 368-373.
[37] L.A. Zadeh, "The Role of Fuzzy Logic in the Management of Uncertainty in Expert Sys.," in *[18],* pp. 3-31.

APPENDIX 1. BIRBAL'S UNCERTAINTY MANAGEMENT

This section describes the operators that are supported in BIRBAL for approximate reasoning. BIRBAL relies on a two-valued interval approach to automate approximate reasoning which measures the *belief* that a certain proposition P is true by assigning an interval [a b] to P, expressing the following semantics:

(1) The probability that P is true is at least a.
The *confirmation* of P is a: conf[a b] = a.
(2) The probability that P is false is at least (1-b).
The *disconfirmation* of P is (1-b): disconf([a b]) =1-b.
(3) The *uncertainty* of our belief concerning P is measured by unc([a b]) = b-a.
(4) The *mean-value* of our belief concerning P is measured by $\left\{\frac{a+b}{2}\right\}$: mv([a b]) = $\left\{\frac{a+b}{2}\right\}$.

For example, if we assign an interval [0.40 0.99] to P we express the following, the confirmation of P is 40% and the disconfirmation of P is 1%. That is, 40% of the probability is assigned to P, and 1% of the probability is assigned to (not P). It is unknown how the remaining probability (59%) is distributed we don't know how much of this probability is assigned to P and how much is assigned to (not P). The uncertainty is 59%, and the mean-value is 69.5%. A special case is the interval [0 1], which expresses the fact that we know nothing about a proposition P. We used the following operators for $\vee$, $\wedge$, not, $\rightarrow$, $\oplus$, and $\subseteq$ in the project, whose definitions are as follows:

Let [a b], [c d], $[l_1\ u_1]$, $[l_2\ u_2]$ be intervals:

$[a\ b] \wedge [c\ d] := [\min(a, c)\ \min(b, d)]$

$[a\ b] \vee [c\ d] := [\max(a, c)\ \max(b, d)]$

$\text{not}([a\ b]) := [1\text{-}b\ \ 1\text{-}a]$

$(a\ b) \longrightarrow (c\ d) :=$ if $(a \neq 0 \wedge (c+a \geq 1) \wedge (b+d \geq 1))$

then $\left[\frac{c+a-1}{a}\ \frac{b+d-1}{b}\right]$

else [0 1]

$$[l_1\ u_1] \oplus [l_2\ u_2] := \left[\frac{l_1 \times u_2 + l_2 \times u_1 - l_1 \times l_2}{1 - l_1 \times (1-u_2) - l_2 \times (1-u_1)} \quad \frac{u_1 \times u_2}{1 - l_1 \times (1-u_2) - l_2 \times (1-u_1)}\right]$$

$$[a\ \ b] \subseteq [c\ \ d] := \frac{a+b}{2} \leq \frac{c+d}{2}$$

The operators for and-, or-, and negation are identical to those advocated by Fuzzy Logic -- only generalized in the context of intervals. The modus ponens operator $\longrightarrow$ is used as follows to associate evidence with predicates appearing on the LHS of rules. For example, if we have a rule:

(R (if E) (then (infer H with [c d]))),

and our belief in E is measured by [a b], then the above rule provides evidence for H, whose amount is measured by: (a b) $\longrightarrow$ (c d).

The proposed modus ponens operator generalizes a modus ponens operator given in ([33]) for intervals. The proposed operator supports smooth decision making, as can be seen for the rule r1, given below

(r1 (if (tall $x)) (then (infer (strong $x) with (0.5 0.9))))

Varying our belief assigned to the tallness of a person, we receive,

Interval for the Tallness	**r1's Positive Evidence**
[1.0 1.0]	[0.500 0.900]
[0.9 0.9]	[0.444 0.889]
[0.8 0.8]	[0.375 0.875]
[0.7 0.7]	[0.285 0.857]

Note that the mean-value of the rule's conclusion decreases if the probability of the LHS decreases means that conclusions based on uncertain knowledge are less reliable and provide less positive evidence. In the above framework negative evidence is treated as positive evidence for the negation of the hypothesis of interest, e.g., when processing: (r1 (if E) (then not(H) with (0.7 1))), we will infer an interval that describes the positive evidence for not(H), and convert this interval to negative evidence for H by applying the negation function, introduced before, to the received interval. In the example, if E is definitely true [1 1], an interval [0 0.3] will be associated with H.

The operator $\oplus$, we use for combination of evidence, is Dempster's rule of combination ([13]). Intervals can easily be interpreted as probability assignment functions. If an interval [a b] is assigned to a predicate P the corresponding probability assignment function would be: m(P) = a, m(not(P)) = 1-b, m(θ) = b-a, and m($\emptyset$) = 0. By applying Demster's rule of combination the above formula is obtained. Finally, we use the mean-value of intervals to rank decision candidates, giving preference to decisions with the highest mean-value for the combined evidence. For example, if an interval of [0 1] is associated with D1 and an interval of [0.1 0.8] is associated with D2, D1 is preferred because its mean-value is 0.5, whereas the mean-value associated with D2 is 0.45, which is less than 0.5.

N92-16576

AUTOMATED PREDICTIVE DIAGNOSIS (APD): A THREE TIERED SHELL FOR BUILDING EXPERT SYSTEMS FOR AUTOMATED PREDICTIONS AND DECISION MAKING

Michael Steib
Vitro Corporation

Abstract. The APD software features include: On-line help, Three-level architecture, (Logic environment, Setup/Application environment, Data environment), Explanation capability, and File handling. The kinds of experimentation and record keeping that leads to effective expert systems is facilitated by: a) a library of inferencing modules (in the logic environment), b) An explanation capability which reveals logic strategies to users, c) Automated file naming conventions, d) An information retrieval system, e) On-line help. These aid with effective use of knowledge, debugging and experimentation. Since the APD software anticipates the logical rules becoming complicated, it is imbedded in a production system language (CLIPS) to insure the full power of the production system paradigm of CLIPS and availability of the procedural language C. This paper discusses the development of the APD software and three example applications: toy, experimental, and operational prototype for submarine maintenance predictions.

INTRODUCTION: FROM SHELL TO TOY TO OPERATIONAL PROTOTYPE

This paper describes the Automated Predictive Diagnosis (APD) environment for development of expert systems and discusses three example applications. It is for readers familiar with production system programming and the development process for building expert systems. After a preliminary look at the output of an operational prototype, the paper begins the discussion with a presentation of the assumptions on which the development was based and an overview of the Automated Predictive Diagnosis software. Once its features and how it is used are described, the architecture of the APD environment and its filing system are explained. This explanation is followed by a recount of the development from the APD shell to Toy Application to an Operational Prototype based on an Experimental Environment with real data. The Operational Prototype for Submarine Maintenance generates printouts of the following type:

```
 SHIP: shp    REFIT: rf                              NAME: name
                                                     DATE: date

 name of part 1
VALUE: value1  THRESHOLD: threshold1
 ***SAT***                      UNTIL: mm-yyyy

 name of part 2
VALUE: value2  THRESHOLD: threshold2
               ***UNSAT***
```

... and so on for the parts of the steering and diving of the submarine. The command to run the computer program and shp, rf, name, and date from the heading on the top two lines are all that is input from the terminal. Of these, only the refit number and the ship number are used by the processing of the software. The sat unsat designations relate to satisfactory and unsatisfactory status of the respective parts of the ship, shp, at the date determined by rf. The latter two, name and date, are used only for identification. Computer files support the rest of the information: derivations to obtain trendable data from sets of untrendable data, parameters needed to run the APD software and predictions. These files contain constant defaults and updated measurement information. The "UNTIL" field is left blank in cases where no prediction can be estimated as to when trendable values derived from data sets would cross the threshold.

A demonstration at this level would consist of a command to start the program, entering the four, shp, rf, name, and date, letting the software run and taking the printout from the printer. In the following discussion, APD refers to the environment to build The APD First Toy Application, The APD Experimental Environment for Submarine Predictions, and The APD Operational Prototype for Submarine Maintenance. It begins with the assumptions on how APD is to support the standard evolutionary incremental build strategy for developing expert systems.

PHILOSOPHY OF THE SOFTWARE: BASIC ASSUMPTIONS WITHIN AN EVOLUTIONARY PROCESS MODEL

The philosophy of the APD software stems from a desire to support standard expert system development procedures like the Evolutionary Spiral Model being formalized by the Software Productivity Consortium, Herndon, VA . The APD concept is based on the following assumptions:

1) Unbundled software capabilities are becoming more readily available, so that the APD software should leave graphic user interface or data base management capabilities to other packages with open interfaces.
2) APD is to be used for expert systems projects where an incremental, evolutionary development process is needed. There will be emphasis on risk management and need for repeated experimental runs with changes in logic, parameters, and data. A standard knowledge acquisition process for expert systems is assumed. 2a) Incremental testing of expert systems should be facilitated. 2b) Software maintenance to update and improve the expert systems is needed. 2c) Programming language capabilities and the production system paradigm are important.
3) The ability to run on 640K is needed.
4) Performance time is not a constraint. In the Operational Prototype for Submarine Maintenance saving operator time is emphasized while saving computer time is not. In the Experimental Environment, ease of maintenance, automated filing, modularity, computer memory, minimal input from a terminal and the ability to scale up were given higher priority than performance time. APD supported this. Of course, in some situations, fast performance saves operator time.
5) Reusable modules are an advantage. A judicious choice of logic modules coupled with an application specific choice of setup parameters should result in the creation of new applications by non-programmers.
6) Standardized utilities across logic modules is an advantage.
7) Predictive diagnosis has priority over other expert system application areas.

The state of the art in expert system shells leaves many of the above to the programmer and often emphasizes coordination with other software (all too often interfaces with particular software as opposed to open interface), or quick prototyping. APD supports these indirectly. Programming power is derived from CLIPS and C, coordination with other software packages is left to open interfaces and portability of the C language, and quick prototyping is supported by the CLIPS production system paradigm and the APD reusable logic and setup modules. To date, no English like user interface has been added to the front of APD. Part of the rational for APD is the desire to support the rigors of the standard expert system development process rather than relegate development to application-programming environments that do not require it. An overview of the APD shell elucidates the development path from the assumptions to the printout of the introduction.

THE APD PURPOSE: AUTOMATED INFERENCING/DIAGNOSIS

The general purpose of the APD software is to automate inferencing processes. The goal is software that automates the process of deriving conclusions from assumptions/data. The APD environment supports experimenting with the process and improving domain assumptions. The software also facilitates record-keeping on domain assumptions and conclusions related to different data sets. More specifically, the purpose is to help automate the process of developing/deriving predictions, especially those related to maintenance needs, from diagnostic domain data and information. The APD software incorporates a methodology that facilitates modularization, a tiered sharing of setup and logic modules, helpful utilities shared across applications, and levels of transparency. Users and programmers may work with changes to the software on three main levels: (1) programming changes (minimum transparency), (2) leaving the programs as is and editing setup files to alter program behavior, and (3) answering queries from front end programs that automatically change these setup files (maximum transparency).

TECHNICAL APPROACH: ROBUST, YET USER-HELPFUL

The APD software is designed to facilitate the development of software systems for real-world situations where the logical rules governing the situation may become complicated by exceptions, complex interrelationships, uncertainty, and sometimes differing ideas and opinions. These kinds of complications are expected under assumption 2 above and suggest the following characteristics of the APD software:

1) The logical rules are embedded in the production system paradigm of the CLIPS language so that:

a) The full power of the production system (rule based) language plus a conventional procedural language, C, is available,
b) Porting to an even more powerful language (Inference Corporation's ART) is facilitated,
c) Source code in C for the production system language is available.

These advantages help to handle possible complex relationships and exceptions to logical rules.

2) Features that facilitate the experimentation suggested by assumption 2, the convenience of assumption 6 and the reusability of assumption 5 include:

a) APD shares reasoning techniques between applications via a library of inferencing modules,
b) An explanation capability reveals reasoning strategies to users,
c) Automated file naming conventions associate assumptions, data, and conclusions via file names,
d) An information retrieval system for data, assumptions, and logic responds to user requests,
e) On-line help is available via a menu system.

These features aid in debugging and possible indecisions and misunderstandings concerning characteristics of domain logic and data. They facilitate the kinds of experimenting and record-keeping that lead to resolutions via the empirical evidence from instantiations of special cases. The remaining assumptions (1, 3, 4, 7) which lead to the coordination with the unbundled approach to software, ability to run on 640K of memory and favoring the expert system application area of predictive diagnosis are supported by the design details and implementation of APD and applications. Thus the APD software uses the off-the-shelf capabilities of CLIPS and facilitates repeated additions as the logic, setup, and data modules are built and saved. The CLIPS Help Facility is used to furnish on-line help for the APD software, its organization, methodology, and capabilities, as well as for the CLIPS software.

USE: CREATING MODULES AND RUNNING APD

There are two major steps in using the APD software. Step 1 is preparing the modules for a particular application area and application situation. Step 1 requires the least amount of work when the needed logic, setup, and data modules already exist for the application situation. If the data module does not exist, then it must be created. This may be done by:

1) Using an existing data module that works with the setup and logic, and
2) Changing the data element values, manually or programmatically.

Similarly, if the setup module for the application area does not exist, it can be created using an existing setup module that works with the logic.

Finally, the most work is required if none of the existing logic modules can be set up to serve the application. Creation of the logic module often requires more than changing another logic module template. It could require a production system programming effort to either alter another logic module or develop one using logical rules gleaned from a knowledge acquisition process. These jobs require a programmer and domain expert. Developing the logic module in an abstract form for use with other applications, although labor-intensive, helps increase the usefulness of the APD software. In the above cases, certain syntax requirements must be followed to ensure coordination with the APD software features: explanation capabilities, data and information retrieval capabilities, and file handling capabilities.

Step 2 is running the software with the appropriate logic, setup, and data to automatically derive conclusions. The files created in step 1 are loaded into CLIPS with APD software, and run to generate the files that contain the conclusions along with files that contain the information and data for automated explanation and data retrieval capabilities. User queries and menu choices can be used. The CLIPS Help Facility can be used to furnish APD as well as CLIPS on-line help.

COMPARISON TO OTHER SOFTWARE: PRODUCTION SYSTEM POWER

Although APD software capabilities may be developed and programmed in several programming paradigms (including functional, procedural, and object-oriented languages), APD was developed and coded in the production system paradigm of the language, CLIPS, to ensure that the full power of CLIPS is available. This CLIPS availability gives the APD software an advantage over off-the-shelf products. Of course, this advantage is not needed unless the complexity of an application requires it. Off the shelf products often provide English like user interfaces that require only minimal programming efforts. However, the cost of continuing past the simplistic environment of the first quick prototype may include an expensive customizing effort or redo. With the source code of the off-the-shelf product unavailable and yet the methodology already in place, the first prototype may need to be abandoned in favor of a more robust programming environment like that furnished via the APD/CLIPS environment.

ARCHITECTURE: INFERENCING MODULES AND UTILITIES

The APD inferencing software is organized into three major sets of modules: logic, setup, and data. The modules in the logic set provide general logic structures used for differing applications. Each module in setup tailors a logic module for a specific application area. Each module in data furnishes the measurements for a specific situation in the application area. Thus a logic module could be used with several application areas, each characterized by a setup module. Each setup module could be used with several situations each characterized by a data module.
Program modules that are useful across application areas and situations relate to filing, file naming conventions, on-line help, query capabilities, explanation capabilities, and data handling. These features as well as the design of the APD software support the expert system development process. The programs and files of APD are in the following directory structure:

The directory APD is the top directory. It contains the other directories.
The addhelp directory contains help files to furnish information to users on CLIPS and APD.
The data directory contains the data files.
The help directory contains programs and utility files to help manipulate the environment where APD is run.
The loaders directory contains programs and files to determine the selection of which data, setup and logic files are included in a run.

There are two methods suggested for running a choice of modules. One uses loader files which specify modules in the APD system to load into CLIPS and run. The other uses a program which loads the modules of a user specified list. In this method, the lists are all in one file, a list of lists. In each of these methods there is the opportunity to run modules in small enough sets to accommodate the 640K memory constraint of DOS. Indeed in the Operational Prototype a batch file does this with the second method. The batch reuses the loading program and takes the place of the user to set up the software sequentially, part after part.

The directory, loaders, contains a file to tell APD which type computer is being used. Some of the development was done on the Macintosh, some on the IBM PC AT. The filing software was made general and instantiated to the particular operating system. Except for this, the filing system is the same on the two platforms.

The logic directory contains the logic program modules.

A simple example of a bit of logic is that a flag is set if a data element value crosses a threshold. A setup file then tells which data element and what value to use for the threshold check. In the following a detailed example of logic is given in the Toy Application.

The programs directory contains the programs for explanation capabilities, retrieval capabilities, comment capture, and a filing system.

For each of the following functions, there is a module in the programs directory:

Prepare the data.
Extract the data from prepared data bases.
Derive the trendable data in the Experimental Environment and Operational Prototype.
Automatically alter the setup files to operator specification.
Automatically reinitialize the setup files for reuse.
Translate the results for display to operator.
File the information for query response and explanations.

Save operator comments.
Display requested results and information.
Explain the results of a run.
Provide a menu for choosing desired information retrieval.
The menu options are chosen with numbers or first letters:

1 (show data elements)
2 (show data value)
3 (show groups)
4 (show result explanation)
5 (show results)
6 (show satisfied criteria)
7 (show satisfied groups)
8 (show tests)
9 (show test criteria)

Explain certain CLIPS error messages for expected user errors.
Help coordinate between APD modules.

The results directory contains the directories, outputs and comments.
Outputs contains the logical results of runs.
Comments contains the user comments.
Comments is usually used only during experimental and test runs.
The setup directory contains the files that are used to instantiate the logic programs.
For instance, if a logic program sets a flag when the value of some data element crosses a threshold, then the setup file instantiates the logic with a data element and value for the threshold.

When automated filing is chosen, it uses this directory structure to organize the run results: outputs and comments stored in the results directory.

The design of the APD software supports modularization and layering. The modularization is implemented with the division of the software into data files, setup files, logic programs, and utility programs. This modularization is augmented by layering. Layering relates to the level of transparency with respect to details of functionality. For the APD software there are three main levels of transparency. These three are variations on instantiating the core logic programs. The first instantiation level is the programming which creates the logic programs. These programs comply with formats needed to use the APD utilities. The programming effort requires enough knowledge of how the logic programs work to yields general programs. They are instantiated on a second level by setup files and data files. This requires less knowledge about the details of APD implementation. However, this editing still requires knowing format requirements for the APD files. The third level of transparency is furnished by programs which automatically insert the setup and data information. At this third level a user answers queries on what to use in the setup and data files and does not need to be concerned with the formats of the files since the input into these files is automated.

These three levels have the customary property: each layer needs to interface with only the next layer down. So to write new programs to query for the setup and data information, the formats for these files must be understood, but not the implementation of the logic programs. Using modularization and layering in this context furnishes an environment for reusable modules, standardized utilities and accommodation of a 640K memory constraint.

THE FILING SYSTEM: AUTOMATED DEVICE FOR ASSOCIATING INPUTS AND OUTPUTS

The APD filing has conventions for directory structure and file names in DOS. The diagnostic logic, setup and data files have common name extensions. The setup files are named nnnapdst.eee where nnn is chosen by the user (a number between 001 and 999 is often used) and eee is the file name extension of the logic file. Similarly, the data file is named nnnapdat.eee, where nnn is chosen by the user (a number between 001 and 999 is often used) and eee is the file name extension of the logic file.

Other files have a name that uses the first two characters to designate what type of file they are, the next three characters to designate the associated setup file, the next three characters to designate the associated data file, and finally, the extension of the logic file. For instance, if the setup file is 001apdst.d05, the data file is 002apdat.d05, and the logic file has file name extension d05, the facts file created by APD would have name ft001002.d05. The user is given the option of not using these filing conventions. The Software queries for either user supplied names or the OK to use automated naming for the results files. For the comments files, the operator supplies the names and depends on the APD software to capture comments and "process location" of comments.

DEVELOPMENT EXPERIENCE: A SEQUENCE OF PROTOTYPES FROM TOY TO OPERATIONAL

The development of the APD software led to an Operational Prototype application for day to day use in predicting submarine maintenance requirements. It began with the implementation of a preliminary shell for development of experimental environments. This was tested with the Toy Application Program. Then the prototype Experimental Environment for predicting submarine maintenance requirements for steering and diving parts was developed. Finally the modules from the Experimental Environment were used to develop a day to day Operational Prototype. This prototype assesses the steering and diving parts, ship by ship, as they arrive and generates a printout like in the introduction to this paper. The following discusses the development of these applications.

REQUIREMENTS: DATA, SETUP, LOGIC, AND PROGRAMS

The programs that furnish a foundation of utilities for APD applications require a compliance to format in the three APD modules: data, setup, and logic. So the programs and files of the three modules are implemented with adherence to the APD formats as well as CLIPS syntax. The following are the example applications built in CLIPS within the APD environment.

THE TOY APPLICATION: BASIC TEST AND DEMONSTRATION OF THE APD SOFTWARE

The first example application was a toy program. In this program data was entered at a terminal since the data base size was manageable. The program was run with test data and the APD shell was debugged. This was accomplished with a domain expert other than end users. However the development of the Experimental Environment and Prototype Application used the target operators as domain experts. The Toy environment depends on checks to see if values exceed thresholds. If a sufficient number of the measurements do exceed the related thresholds, then a "check group" of such threshold checks is called satisfied. When this happens, a postponement of maintenance is recommended. The following presentation of the toy logic is designed to be readable, yet somewhat like the implementation code. The ?'s signify variables.

SAMPLE RULES IN THE COMPROMISE LANGUAGE FOR TOY LOGIC ENVIRONMENT

DEFER_MAINTENANCE_1 Suggests postponing maintenance

IF
1. lead time for ?x is ?lead_time for ?maintenance planning
2. ?x is scheduled for next ?maintenance at time ?maintenance_time
3. present time is ?present_time with ?present_time < ?maintenance_time -?lead_time
4.result number ?n - ?x at time ?present_time has ?maintenance status not needed for a time interval of ?time_till_maintenance_is_needed
5. ?x has ?maintenance scheduled at intervals of length ?time_between_maintenance_performance

THEN
IF
the time interval ?time_till_maintenance_is_needed > (?maintenance_time - ?present_time) + ?time_between_maintenance_performance
ASSERT
result number ___ - at time ?present_time the condition of ?x indicates that maintenance ?maintenance should be postponed at least until the regularly scheduled maintenance time for ?maintenance after this next one derives from result number ?n
AND
IF
the time interval ?time_till_maintenance_is_needed > (?maintenance_time - ?present_time)
ASSERT
result number ___ - at time ?present_time the condition of ?x indicates that maintenance ?maintenance may be postponed until (?present_time + ?time_till_maintenance_is_needed) derives from result number ?n
END OF RULE

INCREMENT COUNTER FOR SATISFIED CRITERIA Increments counter by 1 for each time the criterion for a check in a check group is satisfied

IF
1. result number ___ - ?x has check counter ?check_counter for check group ?check_group at time ?present_time
2. for ?x the check ?check_test in check group ?check_group at time ?present_time satisfied the criterion

THEN
ASSERT
result number ___ - ?x has check counter (?check_counter + 1) for check group ?check_group at time ?present_time
END OF RULE

QUALIFICATION FOR A TIME PERIOD BEFORE MAINTENANCE Notes status of an item in the case when there is satisfaction of criteria for all the checks in a group which is associated with a time period before maintenance is needed.

IF
1. the number of check tests for ?x in check group ?check_group is ?number_of_checks
2. result number ?n - ?x has check counter ?number_of_checks for check group ?check_group at time ?present_time
3. for ?x the check group ?check_group is associated with maintenance ?maintenance and with postponement time interval ?time_till_maintenance_is_needed

THEN
ASSERT
result number ___ - ?x at time ?present_time has ?maintenance status not needed for a time interval of ?time_till_maintenance_is_needed derives from result number ?n
END OF RULE

SIMPLE ABOVE THRESHOLD Notes satisfaction of criterion that a data element has value above a threshold.

IF
1. ?check_test is a check in the check group ?check_group for ?x
2. the criterion for ?check_test is that data element ?data is greater than the threshold ?threshold
3. the data element ?data > ?threshold at time ?present_time

THEN
ASSERT
for ?x the check ?check_test in check group ?check_group at time ?present_time satisfied the criterion
END OF RULE

The following are example setup and data to be used in running the toy example application.

```
;;EXAMPLE SETUP INPUT

(deffacts example-setup "sets up the software for the checks and groups of checks to be made"
(the setup file is "001apdst.d05") (lead time for part_1 is 2 for replace_bushings planning)
(part_1 has replace_bushings scheduled at intervals of length 20)
(the number of check tests for part_1 in check group check_bushings is 3)
(result number -1 part_1 has check counter 0 for check group check_bushings at month apd_date
0 derives from counting satisfied criteria)
(for part_1 the check group check_bushings is associated with maintenance replace_bushings and
with postponement time interval 40)
(test_1 is a check in the check group check_bushings for part_1)
(the criterion for test_1 is that data element measurement_1 is greater than the threshold 1)
(test_2 is a check in the check group check_bushings for part_1)
(the criterion for test_2 is that data element measurement_2 is greater than the threshold 3)
(test_3 is a check in the check group check_bushings for part_1)
(the criterion for test_3 is that data element measurement_3 is greater than the threshold 1))

;;EXAMPLE DATA INPUT

(deffacts example-data "data to be checked"
(the data file is "001apdat.d05")
(part_1 is scheduled for next replace_bushings at month apd_date 5)
(the present is at apd_date 0)
(the data element measurement_1 has value 2 at month apd_date 0)
```

(the data element measurement_2 has value 4 at month apd_date 0)
(the data element measurement_3 has value 2 at month apd_date 0))

EXPERIMENTAL ENVIRONMENT FOR PREDICTIONS: HANDLING REAL DATA

The second example application was with real submarine measurements taken at the steering and diving parts and stored by computer. Availability of this computer readable information and the larger size of the data set suggested a departure from the terminal input of the Toy example. The APD Experimental Environment included programs to automatically create a data input file from the computer readable measurements, and a C program to format the data for use by the APD prototype software. Since the prototype required that the data input file receive trendable data, the software was designed to transform the sets of measurements to trendable derived data. The trends of the derived data were used to determine predictions on when predetermined thresholds were crossed. These threshold crossings were the desired outputs. So, the success of the Experimental Environment for Submarine Maintenance Predictions depended on:

(1) deriving trendable data elements and values,
(2) the trending methods, and
(3) the threshold values. All three of these were supplied by users who served as domain experts. The system was implemented with:
(1) data preparation software in the "C" language to create files that facilitated APD extraction,
(2) setup files to determine how the data extractions and derivations were made,
(3) data extraction and derivation programs that extracted the data from the prepared source files and derived the data elements for trending,
(4) data files for storing the derived data
(5) setup files to determine thresholds and how the trending was to be done,
(6) logic programs that did the trends and compared values with thresholds,
(7) Programs that saved and explained results.

Note that the setup files of (2) and (5) furnished convenient means of changing the way that the extraction and derivations of (3) were made and the way that the trending and comparisons of (6) were done. These setup files furnish convenient generality, a way of using (3) and (6) in different application environments. The choices made available by the setup for extraction and derivation include:

the list of ships to be processed,
constraints on what data is acceptable,
designation of file to store information on inappropriate data,
location of data elements in the lines of the original files,
lines of original file to be processed,
years to be processed,
where to find dates in the original data file,
designation of original file of data,
designation of storage file for derived data,
names of the data elements,
configuration of data use by derivation program,
names of data to be used in result files.

The choices made available by the setup for the trend logic include:

group result names (or a "not used" designation),
names of tests made in the data groups,
criteria names for the checks in data groups,
parameters for estimating last maintenance actions,
parameters for appearance of printouts,
names for files produced,
type of trending.

In runs through the submarine maintenance measurements where there was enough data to yield a sequence of the derived data over time, trends were made. Predictions on timing of maintenance requirements were presented in the form of:

(1) pictures fashioned for ascii character printouts,
(2) tables to be used by graphics packages and
(3) information to be used by the APD utilities.

The utility programs, in turn, responded to user queries on what data values, setup parameters, and logical inferencing was used to get the predictions. The following files were generated by the APD software:

(1) the derived data files,
(2) the "unusable data" files and
(3) the setup files with parameters of the run.

For later reference, data, setup, logic and predictions were associated via file naming conventions for debugging. This will also be useful if and when improvements or updates to the domain assumptions are considered. The main part of these assumptions relate to selection of derivations and trend logic to apply to the different data sets. Since the original data elements were not trendable in a meaningful way, they were grouped into sets which led to files of derived data which was trendable. Creation of the setup files, both for extraction and derivation and for the trend logic was done via editing template setup files. The output files were handled by the automated filing system (except when capture of operator choice of file names was being tested). The Operational Prototype is a sequence of special cases of the software in the Experimental Environment and the following section on this Prototype augments this discussion.

OPERATIONAL PROTOTYPE FOR SUBMARINE MAINTENANCE: A BATCH CONFIGURATION

The third example application of the APD system uses the modules of the second example in a batch mode and assumes all setups constant in the sense that they are not changed as in experimentation with different parameters and logic. These setups do vary from part to part, automatically. When in the batch the user is not bothered with choices. The batch program is set up for the routine job of deciding where to apply maintenance resources for ships when they arrive. It uses a setup which asks the user to stipulate only the ship and date. For this date and for each part, the system returns either unsat or sat depending on whether the derived value has crossed the threshold or not. A special module to present this information part by part for this special status date was added to the Operational Prototype. (The experimental environment presents the date when a value crosses a threshold rather than the status at a "special status date.")

The Operational Prototype batch run does the following:

> It erases the output files from previous runs.
> It erases the data files from previous runs.
> It reinitializes the setup files for data extraction and derivation of trendable data.
> It prepares the data for the APD Operational Prototype.
> It queries the user for the ship, special status date, user name and date of the run and files this information to be sent to the printer.
> It edits the setup files to include the information on ship and special status date.
> For each part:
>> It selects the data derivation for the part.
>> It selects the extraction and derivation setup file.
>> It selects the trending setup file.
>> It makes the prediction on when the derived data is expected to cross threshold.
>> It files the results in the results directory and generates the printout.

Some thresholds are lower thresholds, some are upper thresholds. The trends are based on trendable values derived from the data elements extracted from the data bases. The different data element values are date dependent. A date is considered a valid date only if all the data elements required to derive a trendable data value for that date are found in the data base. The predictions, along with the derived data values at all the valid dates are filed. The predictions are saved for the printout and the values at valid dates are saved for inclusion in visual explanations of the predictions. Also the program for comparison of derived values and threshold at the special status date is included in the run and the resulting sat-unsat status at the special status date is filed for printout. This is done for each special status date that is valid. The file for saving information for the printout is considered complete when the above has been executed for all parts. The input to the batch run consists of four pieces of information. They are the (1) ship and (2) special status date for stipulations of the setup files (3) users name and (4) date of the run. The latter two are for identification on the printout. The ship and special status date are constant throughout the batch run. The output is a list of the parts, the status, sat or unsat, and for each part marked satisfactory, a prediction as to when the derived value for the part will cross threshold (unless there is not sufficient data).

CONCLUSION: APD FACILITATES THE EXPERT SYSTEM DEVELOPMENT PROCESS

The APD software furnishes automated capabilities which aid in the execution of the evolutionary process of expert system development. Storage of information that characterizes experimental runs, retrieval of information on runs, explanation of the inferencing, modular organization, and levels of transparency are emphasized to help make accepted expert system practices convenient without losing the power to work with the complexity of real world problems. The APD shell was used with three main prototypes: a Toy Application, an Experimental Environment with real data, and an Operational Prototype. On-line help was coordinated with the CLIPS help facility. The software was developed on an IBM PC AT, a Zenith Laptop IBM AT compatible, and a Macintosh. Its modularization supports running on only 640K of memory by running small enough groups of modules, one after another. In the Operational Prototype, this was done in batch mode.

REFERENCES

Boehm, B. (1988). A Spiral Model of Software Development and Enhancement, *Computer,* May: 61-72.

Software Productivity Consortium. (1991). *Evolutionary Spiral Process Guidebook,* SPC-91076-MC. Herndon, Virginia

59-61
57689
p. 10

N92-16577

ICADS: A COOPERATIVE DECISION MAKING MODEL WITH CLIPS EXPERTS

cB609194

Jens Pohl and Leonard Myers

CAD Research Unit
California Polytechnic State University, San Luis Obispo

Abstract. This paper describes a cooperative decision making model comprising six concurrently executing domain experts coordinated by a blackboard control expert. The focus application field is architectural design, and the domain experts represent consultants in the areas of daylighting, noise control, structural support, cost estimating, space planning, and climate responsiveness. Both the domain experts and the blackboard have been implemented as production systems, utilizing an enhanced version of the basic CLIPS package. Acting in unison as an Expert Design Advisor, the domain and control experts react to the evolving design solution progressively developed by the user in a 2-D CAD drawing environment. A Geometry Interpreter maps each drawing action taken by the user to real world objects, such as spaces, walls, windows, and doors. These objects, endowed with geometric and non-geometric attributes, are stored as frames in a semantic network. Object descriptions are derived partly from the geometry of the drawing environment and partly from knowledge bases containing prototypical, generalized information about the building type and site conditions under consideration.

INTRODUCTION

Commencing in 1987 with the participation of an interdisciplinary team of researchers the CAD Research Unit at the California Polytechnic State University, San Luis Obispo, undertook the development of a prototype working model of a computer-based environment supportive of the design function as it is practised in architecture and engineering. Impetus for the project came directly from the shortcomings of current CAD systems that focus almost entirely on the production of drawings rather than the design decisions that produced the artifacts represented by the drawings.

The CAD Research Unit team proposed an intelligent computer-aided design system (ICADS) model comprising three integrated components: an intelligent CAD DBMS, an Expert Design Advisor, and a Multi-Media Presentation Facility (Fig.1).

The CAD DBMS component provides on-line access to information resources in direct support of the design function. From the perspective that design involves predominantly the refining, adapting and combining of prototype solutions of previous similar projects, the ICADS model includes several knowledge bases in the CAD DBMS component. These knowledge bases are intended to capture design context experience and serve as a basis for reasoning during the earliest decision-making stages.

Within the CAD DBMS component the evolving design solution is represented as a network of hierarchically related objects. Collectively, these objects provide a comprehensive description of the current state of the design solution. Individually, the description of each object consists of information units or design entities comprising an integrated set of geometric definitions and non-geometric attributes. This information representation drives a network of intelligent design tools (IDTs) coordinated within the Expert Design Advisor by a Blackboard Control System.

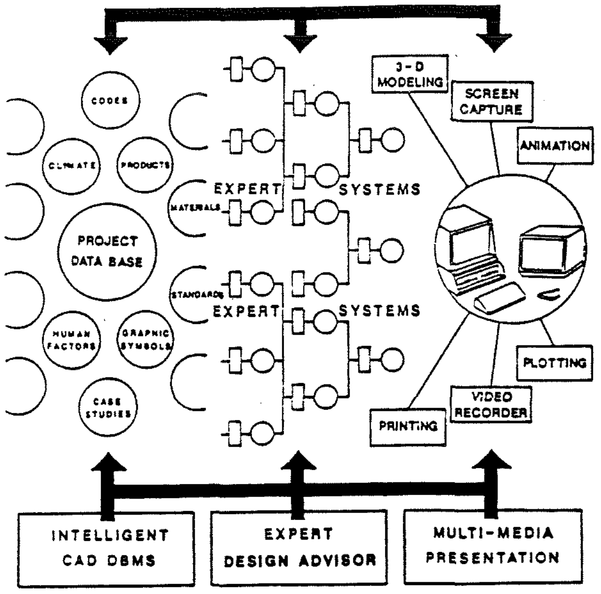


Figure 1.
Conceptual ICADS Model

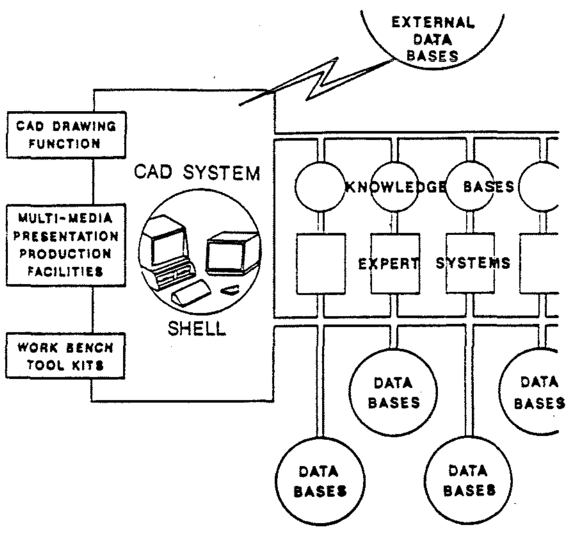


Figure 2.
Intelligent CAD System Shell

The availability of appropriate design knowledge is central to any computer-based design environment. In this respect the ICADS model is foremost an information management and synthesizing system, and may be viewed as a shell that binds together an assortment of internal and external design resources (Fig.2). The drawing function, which dominates existing CAD systems, assumes the secondary role of providing a medium for visualizing and communicating the design decisions that have been made within a largely non-graphical problem-solving context.

The work described in this paper represents the first version of an ICADS working model (Pohl et al. 1988, Myers and Pohl 1989). ICADS DEMO1 has been developed mostly with off-the-shelf software systems. Where the necessary tools were inadequate, enhancements were added and modifications made. The relational DBMS, SQL-RT (Oracle), was found to be entirely adequate for accommodating the Building Type, Site/Neighborhood and Reference elements of the design knowledge component of the ICADS model. The CLIPS expert system shell, developed by the Artificial Intelligence Section at NASA/Johnson Space Center, was used for all parts of the Expert Design Advisor (NASA,1989). Availability of CLIPS source code ('C' language) allowed several enhancements to be made to the basic CLIPS package to permit the Blackboard Control System to execute in a distributed environment with several CPUs.

The CAD Research Unit was fortunate to obtain permission from Accugraph Corporation (El Paso, Texas) to incorporate its MountainTop computer-aided drawing package in ICADS DEMO1. Some modifications were made to this commercially available CAD system to provide direct access to the data structure of the drawing currently displayed on the screen.

THE ICADS EXPERT DESIGN ADVISOR

The principal component of the ICADS DEMO1 model is the Expert Design Advisor, consisting of six domain experts (IDTs), the Blackboard Control Expert, two knowledge bases and several sources of reference data (Pohl et al. 1989).

ORIGINAL PAGE IS
OF POOR QUALITY

The scope of the implementation environment has been restricted in terms of both the breadth of information available to the designer and the range of design functions supported. The information resources provided by the working model are drawn from the architectural design application area and include Building Type and Site/Neighborhood knowledge bases, as well as a Reference database containing material and constructional information.

A schematic diagram of the ICADS working model is shown in Fig.3. The Expert Design Advisor is responsible for the evaluation of the evolving design solution and the resolution of conflicts that may arise when solutions in one domain interfere with solutions in another domain. It consists of advisory components, a control expert and operational components, as shown below:

advisory components:	Geometry Interpreter
	Access domain expert
	Climate domain expert
	Cost domain expert
	Lighting domain expert
	Sound domain expert
	Structure domain expert
control expert:	Conflict Resolver
operational components:	semantic network of frames
	Message Router
	Attribute Loader

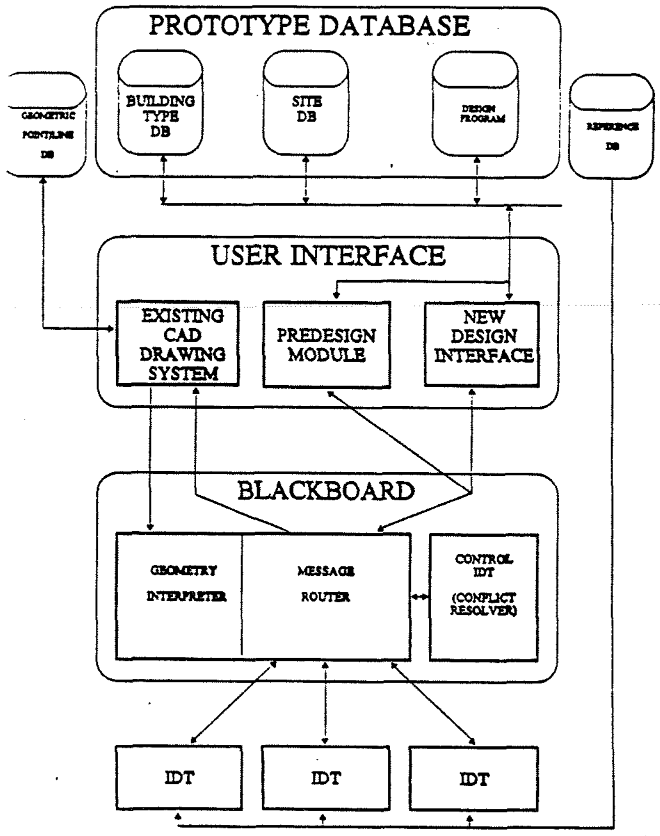


Figure 3.
ICADS DEMO1 System Diagram

Central to the operation of the Expert Design Advisor is a semantic network of frames that represent the current state of the design solution within the context of the project. The term semantic network is used here to refer to a classification framework of design object frames. Each frame incorporates slots that may contain several types of information, such as values of geometric and non-geometric attributes of the current solution model, and relation linkages (Fig.4).

The attribute values represent a fact-list that drives the domain experts and the Conflict Resolver. Indeed, attribute slot names have identical counterparts among the fact names in the latter. This provides a direct interface mechanism that allows the domain experts and the Conflict Resolver to react quickly to any changes in the current state of the design solution. Values in the fact-list are derived from two sources:

1. Geometric facts describing the geometry of the design solution are extracted by the Geometry Interpreter in terms of the nature, physical dimensions and relative locations of the following seven design objects:

   FLOOR, SPACE, WALL, DOOR, WINDOW, SEGMENT and SYMBOL

   The SEGMENT object refers to any part of a WALL object that is demarcated either by the intersection of another wall or has been drawn by the designer as a distinct wall component. The SYMBOL object represents directly by name any closed shape or icon within a SPACE object (e.g., column, chair, table).

2. Attribute facts describing the context of the project and the non- geometric characteristics of the current design solution. These attribute facts are derived from the Building Type and Site/Neighborhood knowledge bases, directly or indirectly through the extrapolation of several information items. Non-geometric attribute values are included in the fact-list in association with the following five design objects:

   PROJECT, NEIGHBORHOOD, SITE, BUILDING and FLOOR

The differences between the geometric and non-geometric design object sets are entirely consistent with the nature of the information they contribute to the fact-list. The design knowledge bases that support the design process in the ICADS model encompass a much wider view of the design space than can be represented by any instance of the geometric design solution. For example, while regional and neighborhood parameters are an important part of the design decision-making process they are no longer discernable as discrete information items in the drawing of the geometric model. At that level they are embedded under several layers of synthesis and are therefore an implicit rather than explicit part of the geometry of the artifact.

In the ICADS working model the distinction between design context and design solution has led to the separation of the semantic network of design objects into two logical sections (Fig.5).

PROJECT DESIGN OBJECT FRAMES: comprising one frame for each design object represented in the design program (design specifications), which is a subset of the Building Type and Site/Neighborhood knowledge bases. Slots in these frames are used to store non-geometric attributes that have either direct equivalents in the Building Type and Site/Neighborhood knowledge bases, or are inferred from several values by the Attribute Loader.

SOLUTION DESIGN OBJECT FRAMES: comprising one frame for each (geometric) design object analysed by the Geometry Interpreter. Slots in these frames represent the geometric descriptions of the particular design object identified by the Geometry Interpreter and the solution evaluation results generated by the domain experts under the coordinating role of the Blackboard.

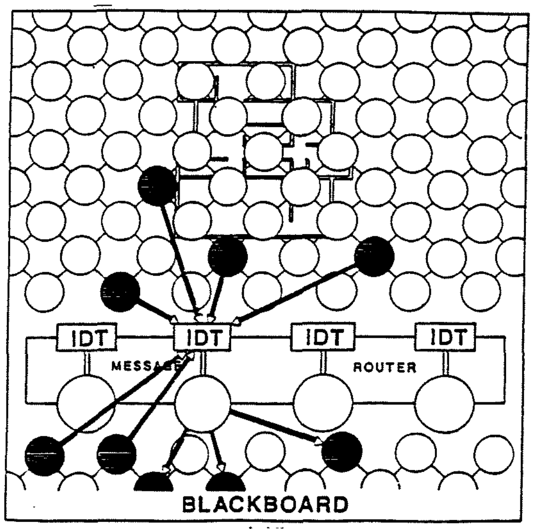


Figure 4.
Semantic Network of Design Objects

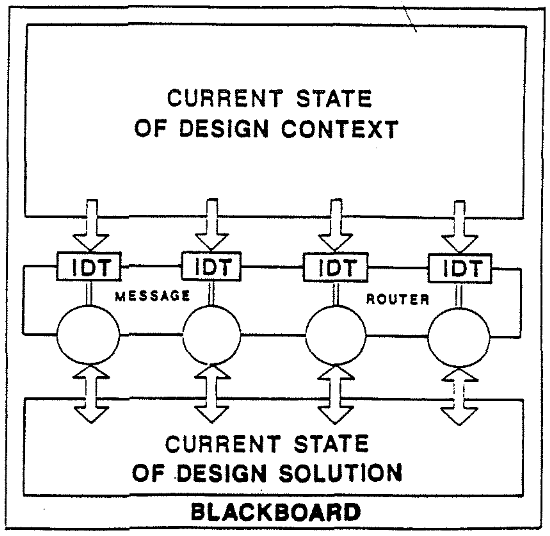


Figure 5.
Logical Division of Design Objects

In the current ICADS working model the slot values of the 'project design object' frames cannot be changed by the designer during the design process. They are established by the Attribute Loader at the beginning of a design session and remain as static members of the input templates of individual domain experts, and to a lesser extent the Conflict Resolver, throughout the design session.

THE DESIGN KNOWLEDGE BASES

The structure of the Building Type and Site/Neighborhood knowledge bases in the ICADS model have been reported previously (Pohl et al. 1988). These knowledge resources are intended to capture the experience and standard solution strategies associated with a given building type, and the specific conditions of the site and its surrounding environment, respectively. Collectively, they provide views of the design project from several vantage points represented by different interest groups (e.g., owner, user(s), designer, community and government authorities).

The Building Type knowledge base is included in the Expert Design Advisor to provide prototype information relating to the type of building under consideration. In this context a prototype is defined as a body of knowledge relevant to the definition and solution of a related set of design problems. The prototype includes generalizations derived from specific instances, elements of previously tested solutions, decriptions of typical solution components, and solution boundary constraints. The boundaries within which the prototype is applicable is provided by the Site/Neighborhood knowledge base, in terms of the requirements and characteristics of the owner, and the physical, environmental, social and economic context of the project location.

THE DOMAIN EXPERTS

The current state of the design solution, represented as object descriptions (containing both geometric definitions and non-geometric attributes), drives six domain experts with evaluation capabilities in the areas of space access determination, construction cost projections, daylighting, sound control, structural system selection and thermal behaviour. Each domain expert, executing continuously in background under a separate process, extracts information pertaining to the particular design object under consideration from the available design knowledge bases and commences to evaluate the design solution based on its expertise (Fig.4).

For example, the Lighting expert will evaluate the degree to which each space in the current design solution satisfies the requirement for daylight. The evaluation consists of two components. First, the requirements are established. This may be a trivial task, requiring only the generation of a simple query to a Reference database to obtain recommended task and background illumination levels for the type of space under consideration. Or, it may be a much more complicated undertaking involving the analysis of qualitative and quantitative design criteria such as:

OWNER considers energy efficiency to be very important;
USER GROUP (A) consider energy efficiency to be optional;
USER GROUP (B) consider energy efficiency to be desirable;
DESIGNER considers energy efficiency to be important;

Recommendations for each SPACE based on past experience:

---

x% of background illumination by daylight;
y% of task illumination by daylight;

Second, the Lighting expert will estimate the daylight illumination on the workplane at the center of each space in two parts. The Daylight Factor is estimated based on the geometry of the space, the geometry of windows in external walls and the reflectances of the internal wall, ceiling and floor surfaces. This Daylight Factor value is converted into an equivalent illumination level subject to an external daylight availability calculation for a particular month, day and time.

The results obtained by each domain expert are added to the appropriate design object frames. In the current ICADS working model both the input and output templates of each domain expert are predetermined sets of attributes and only the values of these attributes are variable. The Conflict Resolver, resident in the Blackboard, examines the values posted by the domain experts and arbitrates conflicts. For example, the Sound expert may have generated the requirement that the north wall of the conference room should have no windows. This is in conflict with the current design solution (based on the Geometry Interpreter) and the Lighting expert who has determined that the '% of background illumination by daylight' for this room is already 15% below the 'requirement'. Based on its own rules the Conflict Resolver determines that the windows in the north wall should be reduced by 20% and double glazed to minimize noise transmission. Apparently the reduction in the availablity of daylight is warranted in view of the noise transmission problem.

The Blackboard posts these new values to the appropriate frames and thereby initiates a new round of evaluations by those domain experts whose previous results are now in conflict with the Blackboard's determination. If the Blackboard had decided that the windows must be deleted from the north wall of the conference room then it would have requested permission for this radical action from the designer. The interaction between the designer and the Blackboard is limited to extreme circumstances in the current ICADS working model. Such circumstances may arise:

1. If the decision of the Blackboard requires a modification of the drawing. In the above example, during the conceptual design stage a 20% reduction in the window area of the

north wall can be accommodated without modification of the 2-D representation of the space. However, the deletion of all windows from the wall would require the drawing to be changed.

2. If the Blackboard cannot resolve a conflict set. Again, in the conference room example, it is conceivable that certain design specifications could mandate daylighting and sound control requirements that will not allow a compromise to be made. Under these conditions, the Blackboard will interrupt the designer and request guidance.

At any time during this evaluation process the designer can request to review the current conflict state of the Expert Design Advisor (i.e., the interactions of the Conflict Resolver with the six domain experts). This is accomplished through the Design Interface on a second monitor.

THE CONFLICT RESOLVER

The Blackboard Control Expert is primarily implemented as the Conflict Resolver in the ICADS DEMO1 model. While it is envisaged that 'planning' will play an important role in future implementations of the ICADS model, at this time the resolution of conflicts appears to be sufficient to coordinate the activities of the advisory components in the Expert Design Advisor.

The principal purpose of the Conflict Resolver is to assert 'current value' frame slots, representing the current state of the evaluation process performed by the domain experts, onto the semantic network resident in the Blackboard. To accomplish this the Conflict Resolver requests from the Message Router all of the 'solution design object' frames which contain results generated by the domain experts. Current values fall into one of three basic categories: values which result from solutions proposed by a single domain expert; values which result from solutions proposed by several domain experts for a common current value; and, values which must be inferred from solutions proposed by several domain experts.

In the case of the first category, which represents solution values unique to a single domain expert, the Conflict Resolver does not change the values proposed by the expert. The proposed solution values are simply asserted as current values into the appropriate frame slots. In the second category two or more domain experts propose differing values for the same solution parameter. In such direct conflict situations it is the responsibility of the Conflict Resolver to either determine which of the values is most correct or to derive a compromise value. The process of resolution may cause the Conflict Resolver to change several current values in addition to those in direct conflict. An example of such a change is given below:

```
Structural expert:     'suggested roof material type' = A
Climate expert:        'suggested roof material type' = B

        Rule 1: if A == B then
current value          'suggested roof material type' = A

        Rule 2: if A == timber and B == concrete then
current value          'suggested roof material type' = concrete

        Rule 3: if A == concrete and B == timber then
current value          'suggested roof material type' = concrete
                       and
current value          'suggested roof struct.system' = concrete plate
                       and
current value          'required roof struct. depth'  = 4
                       and
current value          'roof insulation thickness'    = 3
```

The Conflict Resolver incorporates resolution rule sets which determine the best current values from those proposed. There is a resolution rule set for each possible direct conflict. In the development of each rule an attempt has been made to achieve a desirable balance between the various design issues. At this level the Conflict Resolver can be consider an expert whose knowledge is the ability to achieve this balanced integration. In the above example, let us assume that the Structural expert proposes 'timber' (A) and the Climate expert proposes 'concrete' (B). Under these circumstances the Conflict Resolver recognizes that:

- the solutions for the roof structure proposed by the Structural and Climate domain experts are substantially different;

- the 'concrete' solution proposed by the Climate expert suggests a need for thermal storage;

- the 'timber' solution proposed by the Structural expert cannot be readily modified to provide thermal storage;

- in most cases a structural timber system can be replaced by a concrete system (there are exceptions to this rule of thumb (e.g., seismic risk) and the Structural expert should be able to recognize such circumstances and, if necessary, refuse to agree with the Conflict Resolver's proposed compromise);

- the energy conservation savings provided by a passive thermal building are likely to exceed the higher capital costs of a concrete roof system.

In the third category the Conflict Resolver deals with proposed solution values that are indirectly in conflict with other proposed solution values and current values. The resolution rules for this category allow the Conflict Resolver to make the necessary modifications to any of the values involved. Under these conditions, in addition to changing proposed solution values the Conflict Resolver may also change current values as shown in the following example.

```
current value 'roof construction system'       = A
current value 'roof U-value (BTU/HR-SF-F)'= B
current value 'roof insulation thickness'      = C
current value 'roof thermal lag (HR)'          = D
```

Climate expert: 'suggested roof construction system' = E

Rule 1: if A <> E then
look up Reference database and change B, C, D to appropriate values for the suggested roof construction system 'E'

In this example the Climate expert has suggested a new roof construction system. The Conflict Resolver recognizes that several current values must be changed so that they match the new roof construction system. Similar to the direct conflicts discussed under the second category, each indirect conflict must also have a set of resolution rules.

Not all conflicts can be resolved by the system. In some cases, usually those requiring changes to the drawing or 'project design object' frames, the Conflict Resolver will ask for assistance from the designer. Under these circumstances operation of the Expert Design Advisor is suspended until the designer responds.

SOME IMPLEMENTATION ISSUES

The model of the ICADS Expert Design Advisor described above presents several issues of concern that are of an operational nature.

The first concern is to prevent the Conflict Resolver from entering into an 'endless argument' state that may arise when two domain experts always return with conflicting values for the same solution parameter. For instance, in reference to the previous example the Structural expert may insist for good reasons that the roof construction system should be 'timber'. For reasons that are different but just as persuasive, the Climate expert may be unwilling to deviate from its original proposal of a 'concrete' roof system. The two domain experts are now locked into an ad absurdum argument. In the current ICADS working model the Conflict Resolver monitors this type of situation by checking for repetitive cycles in current values.

A second concern is related to the timeliness of the conflict resolution process. In the implemented model of the Expert Design Advisor, the Conflict Resolver will not post a current value to the Blackboard until it has seen all of the 'solution design object' frames which are involved in a given conflict. In a full ICADS implementation it may be desirable for the Conflict Resolver to post a current value based on partial information (i.e., based on only some of the required 'solution design object' frames). This current value could be updated when the Conflict Resolver receives additional information. In the implemented model the Conflict Resolver waits until it has all of the necessary values from the domain experts before asserting a new current value.

The potential for a 'cascading' condition which may arise when a minor change by a domain expert causes a major re-evaluation of the current solution by several domain experts, is another concern. In the DEMO1 implementation of the Expert Design Advisor the possibility of such a condition occurring is exacerbated because domain experts will fire on any change to a current value and the Conflict Resolver will respond to any proposed changes posted by the domain experts. An attempt has been made to forsee events that could conceivably lead to this undesirable condition. Where appropriate, tests have been included in the rules of domain experts to determine whether the current divergence between the most recent suggestion and the corresponding current value (proposed by the Conflict Resolver) is sufficiently large to warrant further action by the domain expert.

CONCLUSION

The implementation of the first prototype working model represents a three-year milestone in the ICADS project. Although ICADS DEMO1 is limited in scope, it will provide a vehicle for the collection of a body of knowledge relating to the performance characteristics of a computer-based design environment. Extensive explorations can now be conducted to determine the impact of an Expert Design Advisor that dynamically responds to the actions of the designer in its continuous, unobtrusive evaluation of the evolving design solution.

REFERENCES

Myers L. and J.Pohl (1989); 'ICADS DEMO1: A Prototype Working Model'; Fourth Eurographics Workshop on Intelligent CAD Systems, Paris, France, April 24-7. NASA (1989); CLIPS Architecture Manual: Version 4.3; Artificial Intelligence Section, Lyndon B.Johnson Space Center, NASA, USA.

Pohl J., A.Chapman, L.Chirica, R.Howell and L.Myers (1988); 'Implementation Strategies for a Prototype ICADS Working Model'; Technical Report: CADRU-02-88, CAD Research Unit, Design Institute, California Polytechnic State University, San Luis Obispo, California.

Pohl,J., L.Myers, A.Chapman and J.Cotton (1989); 'ICADS: Working Model Version 1'; Technical Report: CADRU-03-89, CAD Research Unit, Design Institute, School of Architecture and Environmental Design, California Polytechnic State University, San Luis Obispo, California.

OMIT

SESSION 3 A

113

PAGE 112 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

S10-61

57690

N92-16578

p-8

CB609194

A CLIPS/X-WINDOW INTERFACE

Kym Jason Pohl

CAD Research Unit
California Polytechnic State University, San Luis Obispo

Abstract. This paper describes the design and implementation of an interface between the CLIPS expert system development environment and the graphic user interface development tools of the X-Window system.

The underlying basis of the CLIPS/X-Window interface is a client-server model in which multiple clients can attach to a single server that interprets, executes and returns operation results, in response to client action requests. Implemented in an AIX (Unix) operating system environment, the interface has been successfully applied in the development of graphics interfaces for production rule cooperating agents in a knowledge-based CAD system. Initial findings suggest that the client-server model is particularly well suited to a distributed parallel processing operational mode in a networked workstation environment.

INTRODUCTION

Graphic user interfaces are gaining in importance in all computer application areas. Once the almost exclusive domain of CAD users such as architects and engineers, who have traditionally used drawings to visualize design solutions, they are today the preferred medium for virtually all computer-user interactions. A recent study of experienced and novice workstation users in business offices indicated a significant increase in productivity and quality of tasks performed with graphical user interfaces, for both groups (Temple et al. 1990).

Clearly, navigation through applications and the selection of functional options in a window environment with pointing devices is far superior to typed commands and keyed data entry. However, an equally important advantage of graphical user interfaces is the greatly increased 2-D and 3-D visualization capabilities that can be integrated into the application environment. Particularly with the emergence of more complex application systems involving expert systems, distributed databases and integrated parallel processing in networked workstation environments, the need for complex data display capabilities has become no less important than operational efficiency.

Several considerations drove the development of the GXI graphics interface builder. First, it was recognized that data displays should convey not only values but also the context in which the data values exist (Tufte 1990, Abler 1989). Unfortunately, the majority of data displays seen today, such as tables and point line graphs, are two-dimensional in character. This does not recognize the fact that the human user lives in a three-dimensional world and has excellent facilities for perceiving and reasoning about complex solid images. With the decidedly higher level of computer-based graphics capabilities available today data displays should no longer suffer from these limitations. Utilizing advanced graphics programming tools, such as MIT's X-Window system, data can now be expressed in three-dimensional graphs or real world solid objects.

Second, recent advances in computer hardware, examplified by IBM's RS/6000 and Hewlett Packard's HP-700 workstations, provide support for more sophisticated applications software systems. Representing a third generation of workstations with greatly improved reduced

PAGE 114 INTENTIONALLY BLANK

PRECEDING PAGE BLANK NOT FILMED

instruction set computing (RISC) technology, these computers provide speeds in excess of 50 million instructions per second (MIPS) and more than 64 million bytes (MB) of fast memory. Combined with a multi-tasking operating system, such as UNIX, these workstations are capable of supporting applications software packages in which multiple processes interact with each other. For example, knowledge-based design systems in which several expert systems interact with each other through a blackboard control mechanism, while they evaluate the evolving design solution in real-time (Myers et al. 1991). While multi-tasking has opened a wide range of new applications opportunities, it has also added a new perspective to the object-oriented software design approach. Software objects can be treated as semi-autonomous processes, executing concurrently on one or more workstations and communicating with each other through sockets or similar inter-process communication facilities.

Third, an increasing involvement of applications experts with limited computer science knowledge and skills in software development is establishing a demand for higher level programming tools. It can be argued that since this trend is likely to lead to more useful and effective applications software, every effort should be made to provide software tools that are relatively easy to apply and yet do not unnecessarily constrain the applications developer within simplistic structures and paradigms.

The fourth motivating factor for the development of GXI deals with a crucial limitation of many AI language environments. In the current state of AI, most high level programming environments such as the CLIPS expert system shell are limited in that they have virtually no graphics facilities. Therefore, the programmer is considerably restricted in the quality of the interface that can be presented to the user. To solve this limitation, a set of routines may be developed to extend the CLIPS language to include the interface facitilies offered in such graphical environments as X-Window. Such an extension would allow for considerably more robust and interactive applications in an AI environment.

Within the context of these considerations the development of the GXI interface builder was undertaken by the author in response to several needs that arose in the CAD Research Unit of the School of Architecture and Environmental Design at Cal Poly. The interdisciplinary nature of the various project teams established the need for a set of higher level tools that could be used by architects and engineers for prototyping software modules. Typically, this work includes expert systems, databases and procedural programs. Any meaningful evaluation of these prototype models requires the involvement of practising architects and engineers, who would be unduly influenced in their assessments by operational complexities and unrealistic user interfaces. For this reason the availability of a graphics interface builder, serving as a high level development tool for applications experts with limited software engineering background, became a high priority requirement.

ALTERNATIVE APPROACHES

Two main approaches were considered during for the design of the GXI interface builder. The first was based on the concept of providing a set of client graphics calls along with their implementation in one physical process. This approach brings with it some distinct advantages. It allows the client requestor and the graphics server to exist as one cohesive process. Accordingly, all graphics actions can be centralized on a single machine alleviating the need for the software developer to deal with the complexities of a networked environment. The second advantage is also related to programming simplicity. The single process approach allows the programmer to link directly to a library of robust graphics routines, for creating interactive menuing systems and graphical objects.

However, the single process approach also has several inherent disadvantages. As mentioned earlier, both the graphics requestor and the graphics server reside in the same process. Therefore, several graphics applications running concurrently cannot be physically or logically connected. Each exists as an independant entity completely insulated from the other. This must inevitably lead to duplication of code and sequential processing of graphics requests.

A second deficiency arises when the application system resides in a networked environment. The single process approach makes provision only for dealing with the local domain environment. In view of the numerous advantages of telecommunication networks, the requirement for interprocess communication across a network has become a high priority consideration. The single process approach provides no facilities for interprocess communication on the current machine let alone across a network. This limitation simply becomes too costly when application systems of larger size are considered.

The third, and perhaps most serious disadvantage of the single process approach is related to the architecture of the X-Window graphics system, which was mandated as a precondition for the targeted user environment. X-Window provides a collection of graphics primitives to the application programmer. However, these primitives exist in basic form and require extensive programming knowledge of the X-Window graphics environment. While the GXI interface builder was expected to utilize these tools extensively, it was considered important that the complexities of the low level tools be hidden from the software developer.

It is relevent, at this point, to briefly discuss the method used by X-Window to accept, perform and reply to application graphics requests. When the application makes a request of the X-Window server an event is placed on an output queue located on the server side. This output queue is unique to each application client and can be readily accessed by its owner. Event structures contain all of the information required by the X-Window server to bring into existance the particular request or event. When the application wishes to execute an event, it simply removes this prepackaged event from the queue and dispatches it through a series of calls to the X-Window server. To simplify this process even further, the application has the option of entering into an event loop which monitors the application's output event queue. When an event is loaded onto the queue, it is immediately dispatched. At any given time if there are no more events left on the queue, the application may choose to block (ie., sleep) until another event is posted to the queue. Such events can be sent as the result of either a client graphics request or a mouse interaction by the user with an active menu button.

Adhering to this single cohesive unit approach means that the X-Window server must give up control to the application once it has fulfilled the clients request. This has two serious shortcommings. The first is related to a peculiarity common to most large scale graphic tools systems. As mentioned earlier, the X-Window system is a complex system consisting of several processes which have the ability to communicate with each other across a network. Using the single process approach, control moves from the client application to the request server upon the issuence of a graphics request. With control delegated to the server side, the client is put to sleep while the server attempts to carry out the particular request. This is analogous to the situation which arises when a program wishes to read a number from the keyboard. Once the program has issued the appropriate system input call, control moves from the program to the operating system. The client program is simply blocked until the user enters a character and presses the 'return' key. As soon as the user has completed the entry sequence, control is returned to the program. However, during the interval the operating system is able to capture other requests or events independent of the particular client application. This is due to the fact that the operating system itself consists of several processes. It is therefore apparent that in the case of an operating system, control actually resides in several places at the same time. The ability of the operating system to be receptive to multiple client application requests is typical of most client-server relationships.

The X-Window environment is also based on the client-server model. In this environment it is by no means a trivial matter to remove control from the X-Window system. Any attempt to artificially break the event catching and dispatching loop of X-Window can cause serious synchronization problems between the client and the server. For example, it is certainly possible for a series of queued menu creation events to be dispatched in non-chronological order resulting in the display of a menu without buttons. The ability of X-Window to block itself and subsequently wake itself up again, is typical of many large scale graphics tool systems. It was considered important that the GXI interface-builder be compatible with this kind of control environment.

This concept of movement of control throughout the system illustrates the second shortcomming of the single process approach. Once the graphics server has carried out the particular client request, there is no other option but for the server to relinquish control to the client. This means that the server has been literally put to sleep, and remains in this dormant state until the the client makes another request to the server. Since the very essence of an interactive interface is predicated on continuous receptiveness to I/O activity, the single process approach is highly undesirable.

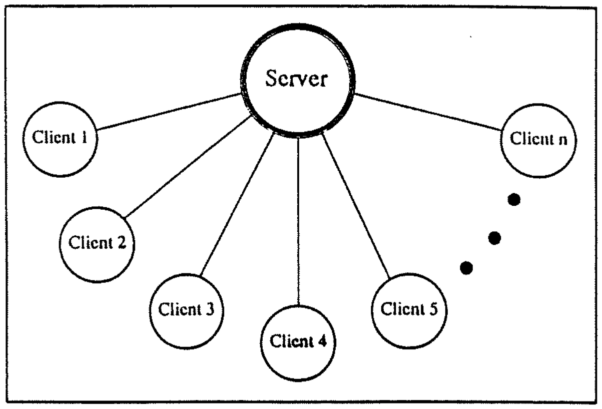


Figure 1.
Client-Server Model

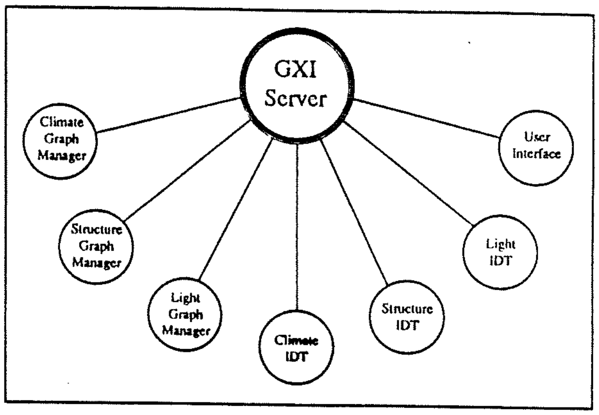


Figure 2.
GXI Application

A second, and considerably more favorable, approach to solving the design problem posed by the GXI interface-builder involves multiple processes (Fig.1). One of these processes is a graphics request server. Similar to the previous models, client applications make graphics requests of the server which in turn performs the necessary work and returns the results (ie., a handle to the graphics object). The clients making the requests need not be concerned how their graphics requests are being carried out. Thus, the client application is removed from the complexities of internal graphics processing and representation.

The advantages of the client server approach are threefold. The first advantage deals with the problem of control. As mentioned earlier, serious synchronization conflicts can arise when control is forced upon the system. Using a client-server design, the server along with each of its client applications resides in its own process. Therefore, similar to the operating system case discussed previously, control resides in several places concurrently. These independent entities communicate with each other through a message passing procedure, thus allowing the server the freedom of blocking and subsequently waking itself. Therefore, the server can be receptive to any interactive activity taking place under the direction of the user, independantly of the current state of the clients.

Second, the logical and physical connection which was lacking in the single process server design is now present in an organized and complete fashion. There exists only one graphics server which performs all of the graphics work requested by each client. This is true no matter which client on which networked machine is actually making the request. The server can be thought of as an invisible workhorse running in background somewhere on the system. Each client simply requests a connection to the server at the beginning of the session. Since all requests are made to a centralized server, there now exists a logical and physical connection between each of the server's clients. Even though all information is channeled through the server, graphics data can be easily passed from one client to another via the server.

The third advantage deals with the potential for distributing the work load generated by a sizable applications system over several networked workstations. The distributed processing model allows the client processes and displays to be assigned independently of each other to any machine/monitor on the network.

IMPLEMENTATION OF THE GXI INTERFACE-BUILDER

The GXI server is designed to accept any number of client applications written in either the 'C' programming language or the CLIPS expert system shell (Fig.2). Since the design of GXI is based on a client-server model, it adheres closely to the guidelines commonly set forth for such environments; namely, multiplicity of clients, parallel request handling, management of the client environment and a common communication protocol (Stevens 1990, Tanenbaum 1987). The two principal entities of the client-server model, the server and the client, must be designed to support a cooperative environment in which requests from its various clients are satisfied with minimum delay.

Typically the server exists in an endless loop performing three basic functions: accepting a client request; performing the requested work; and, then returning the results of the operation to the client. To allow parallel request handling, these three functions are divided between a mother server and a client request handler.

The first function, to accept requests from any of the clients, is accomplished through the use of internet sockets. The server simply waits on the socket for the next request. Once a prospective client has requested a connection to the server, the mother server forks an identical image of itself to handle the request of that particular client. This child process is referred to as a client request handler.

In the GXI implementation, as soon as the client request handler has been created, the mother server is free to return to the top of the loop where it waits for the next client connection request. The child server performs the same three functions as the mother server, however, dealing exclusively with its own client. This request handler will repeat these functions until the client requests termination of its GXI session. At this point the child server terminates itself.

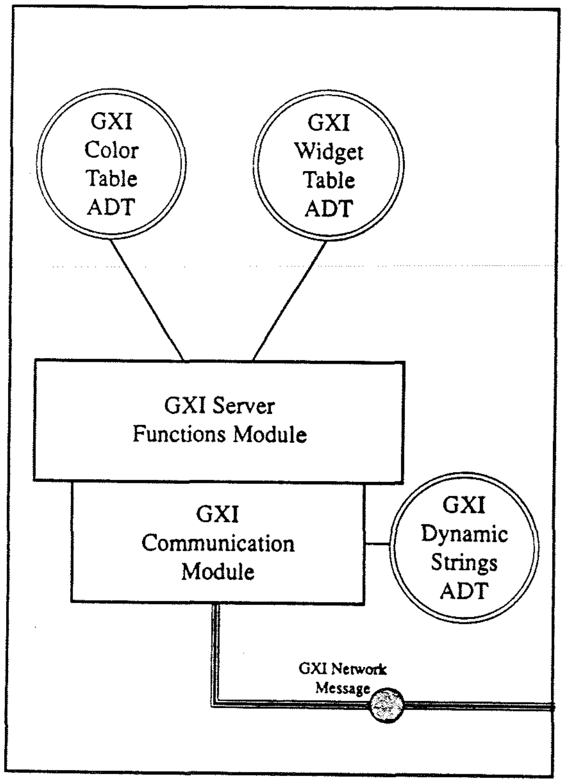


Figure 3.
GXI Server Architecture

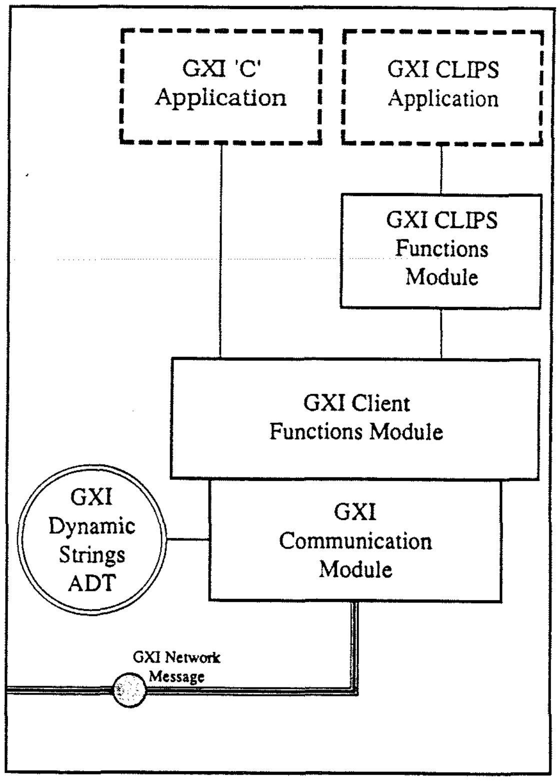


Figure 4.
GXI Client Architecture

To aid the child server in managing the particular environment its client is creating, it keeps an environment table. Among other information this environment table keeps track of the relationship of the graphic objects the client has created and the specific handle of each object that was returned to the client upon creation (Fig.3). To be more specific, each graphic object, such as a menu or a graph, has an associated handle which is used by the client to identify that particular object at a later time.

In addition to these tasks the client request handler also performs maintenance on the client's graphic environment. This may take the form of redrawing a newly exposed region of an object, or maintaining a color table. By caching the color values in a hash table a significant increase in performance is achieved.

The tasks described above for the child server are performed in a manner which is transparent to the client. A communication module which handles the dialogue between the client and the server is attached to each client (Fig.4). When the client issues a request to the server, a message is formulated according to a common communication protocol. This protocol is designed to allow for the transfer of both static and dynamic information. For example, client requests dealing with the number of buttons to be displayed in a menu or the number of points to be used for defining a polygon are not predefined from the point of view of the server. They require a dynamic data transfer capability that must be accommodated by the communication protocol. Once the message has been sent to the server, the client assumes a sleeping state until the results of the requested operation are returned.

The following is an example of an interractive session between a client and the GXI server. In this example the client creates a simple user interface in the GXI environment.

```
(deffacts buttons (Buttons      "Access Database"
                                "File System"
                                "Help"
                                "Exit")
)

(defrule CreateThermalInterface
        (Buttons $?MenuButtons)
=>
        ; ********* Request a Connection to GXI    *************

        (bind ?Sheet  ( XCInit          ?Client ?ColorFile ?XPos ?YPos
                                        ?Height ?Width ?BorderWidth
                                        ?BorderColor ?BkgColor
                      )
)
        ; ********* Create an Interface Banner    *************

        (bind ?Banner ( XCBanner        ?Sheet ?BannerText ?BorderWidth
                                        ?Length ?TextColor ?BkgColor
                                        ?FontStyle
                      )
)

        ; ********* Create the Main Menu          *************

        (bind ?Menu   ( XCMenu          ?Sheet NULL ?Banner ?BorderWidth
                                        ?BorderColor ?TextColor ?BkgColor
                                        ?Horizontal ?NumButtons
                                        $?MenuButtons
                      )
)
        ; *********  Assert the New Environment     *************

        (assert(ThermalSheet  ?Sheet))
        (assert(ThermalBanner ?Banner))
        (assert(ThermalMenu   ?Menu)
)
```

At this point the client may choose to read from the menu it has just created. After the user has made his or her menu selection, GXI utilizes the extensive pattern matching capability offered in the CLIPS environment. The client rule having the appropriate button pattern will fire thus performing the action corresponding to the selected button. Each menu button should have an associated action rule adhering to the following format:

```
(defrule AccessDatabase

        ; *********  Match on the Database Button    *************

        ?Selection <- (Thermal "Access Database")
=>

        ; *********  Perform Thermal Database Access *************
        ;    ...
        ;    ...

        (retract ?Selection)
)
```

TYPICAL APPLICATION

The GXI interface builder was first applied in the computer-aided design field, in the domain of architectural design. In the building design process it is useful for the designer to examine hourly temperature data for an average year for the site. By comparing these values with the range of human comfort, the designer can establish which times during the year require heating, and which require cooling. This information is usually presented in a table with rows of months, and columns of hours in the day.

However, using the client-server model of GXI, these climatic data are passed from an expert system, to a graphic display program connected as a client to the GXI server. Using the lines, polygons, and other primitives provided by GXI, the data are represented graphically as a 3-D contour model, with the x-axis representing months, the y-axis representing hours of the day, and the z-axis representing temperature. By filling each polygon in the contour model with a color relating to its level of comfort (shades of blue for cold areas and red for hot areas), the relationship of temperature to human comfort is also incorporated. More detailed information about a certain month or time of day, can be displayed by taking sections of the contour model. These sections, selected by the user, are displayed as 2-D graphs, and multiple sections can be chosen and superimposed on each other to compare different months or times of day. Each graph can be resized and rotated, and by saving data from the expert system in a file, multiple climates can be displayed side by side, allowing comparison of different sites.

The entire client is mouse driven, using buttons, dialog boxes, and pull-down menus provided by the GXI server. By using GXI to display climatic data graphically, the user is able to examine hundreds of points of data, define custom views, and quickly evaluate the climatic conditions of the site.

A second application involved the generation of space layouts during the earliest stages of building design. GXI was used to build a graphic display facility for a layout advisor written in CLIPS (NASA 1989). In this case, GXI provided a rich selection of graphic object manipulation tools embedded as user defined external functions in the CLIPS programming environment.

CONCLUSION

The GXI graphics interface builder responds to the needs of an increasing number of domain experts who wish to build technically sophisticated applications software, incorporating high quality graphical user interfaces, without having to deal with the complexities of low level procedural languages. The client-server implementation model was found to be particularly suitable to multi-process applications in which the performance of the application system as a whole depends largely on the efficacy of interprocess communications.

REFERENCES

Abler F.(1989); 'Metashapes: Voxel Data Analysis for Computer Aided Design'; Design Methods and Theories, 23(4) (pp.1088-99).

Byrd L.(1991); 'PROLOG and Client/Server Information Systems'; Computer Language, March (pp.37-43).

Myers L., J.Pohl and A.Chapman (1991); 'Computer-Based Intelligent Design Assistance: Concepts and Strategies'; First International Conference on Artificial Intelligence in Design, Edinburgh, Scotland, June 25-27.

NASA (1989); 'CLIPS Reference Manual (Version 4.3)'; Artificial Intelligence Section, Lyndon B.Johnson Space Center, NASA, May.

Stevens W.(1990); 'UNIX Network Programming'; Prentice Hall (pp.137-169, 258-339).

Tanenbaum A.(1987); 'Operating Systems: Design and Implement- ation'; Prentice-Hall (pp.40-2, 51-75).

Temple, Barker and Sloan Inc.(1990); 'Smile when you say GUI'; Trends to Watch, Computer-Aided Engineering, September (pp.23).

Tufte E.(1990); 'Envisioning Information'; Graphics Press.

511-61

57691

N92-16579

p-16

APPLICATION OF MACHINE LEARNING AND EXPERT SYSTEMS TO STATISTICAL PROCESS CONTROL (SPC) CHART INTERPRETATION

A1 277627

Mark Shewhart

Air Force Logistics Command (AFLC)
Acquisition Logistics Division
Joint Technology Application Office (ALD/JTI)
Artificial Intelligence Support Center (AISC)
Wright Patterson AFB, Ohio 45433

Abstract. Statistical Process Control (SPC) Charts are one of several tools used in Quality Control. Other tools include flow charts, histograms, cause-and-effect diagrams, check sheets, Pareto diagrams, graphs, and scatter diagrams. A control chart is simply a graph which indicates process variation over time. The purpose of drawing a control chart is to detect any changes in the process, signalled by abnormal points or patterns on the graph. The Artificial Intelligence Support Center (AISC) of the Acquisition Logistics Division (ALD/JTI) has developed a hybrid machine-learning/expert-system prototype which automates the process of constructing and interpreting control charts.

INTRODUCTION

The Air Force Logistics Command (AFLC) has provided TQM and Quality Control training to its employees for several years now. In particular, Statistical Process Control has been emphasized in this effort. While many data collection efforts have been undertaken within AFLC, the SPC Quality Control tool has been under-utilized due to the lack of experienced personnel to identify and interpret patterns within the control charts. The AISC has developed a prototype software tool which draws control charts, identifies various chart patterns, advises what each pattern means, and suggests possible corrective actions. The application is easily modifiable for process specific applications through simple modifications to the knowledge base portion using any word processing software.

The remainder of this paper consists of the following sections :

(1) **CONTROL CHARTS**
(2) **SOFTWARE FUNCTIONALITY**
(3) **SOFTWARE DESIGN**
(4) **MACHINE LEARNING**
(5) **EXPERT SYSTEM**
(6) **CONCLUSION**

Section (1) provides a more in-depth explanation of the purpose of control charts. Section (2) details the initial functional requirements for the SPC software, and section (3) outlines the design approach used to implement the system requirements. Sections (4) and (5) examine in detail the roles of machine learning and expert system techniques respectively. Finally, section (6) offers some basic conclusions resulting from this effort. Two attachments are included after the references. **ATTACHMENT A** provides a list of the chart patterns of interest and their methods of identification. **ATTACHMENT B** enumerates and explains the twenty statistical features used by the machine learning tool.

CONTROL CHARTS

An example of a control chart is given below in **FIGURE 1**. A run chart is a plot of a process measurement (e.g. bore diameter or time to process an insurance claim for example) on the vertical axis (y-axis) against time on the horizontal axis (x-axis). A control chart is simply a run chart with statistically determined upper (Upper Control Limit - UCL) and lower (Lower Control Limit - LCL) lines drawn on either side of the process average. These limits are calculated by running a process untouched, taking samples of the process measurement, and applying the appropriate statistical formulas (references [3-9]).

The random fluctuation of points within the limits results from variation built into the process. Such random variation is natural, results from common causes within the system (e.g. design, choice of machine, preventative maintenance, etc.), and can only be affected by changing the system itself. However, points which fall outside of the control limits or which form "unnatural" patterns indicate that some of the variation within the process may be due assignable causes. Assignable causes of variation (e.g. measurement errors, unplanned events, freak occurrences, etc.) can be identified and result from occurrences that are not part of the process.

The purpose of drawing the control chart is to detect any unusual causes of variation in the process, signalled by abnormal points or patterns on the graph. The AISC developed software tool automatically identifies nine types of patterns which indicate the presence of assignable causes of variation in a process. Examples of these patterns are given in **FIGURES 2 - 10**. Each such pattern is associated with generic advice about what may be happening at that point in the process. More detailed information about each of the nine patterns is given in **ATTACHMENT A.**

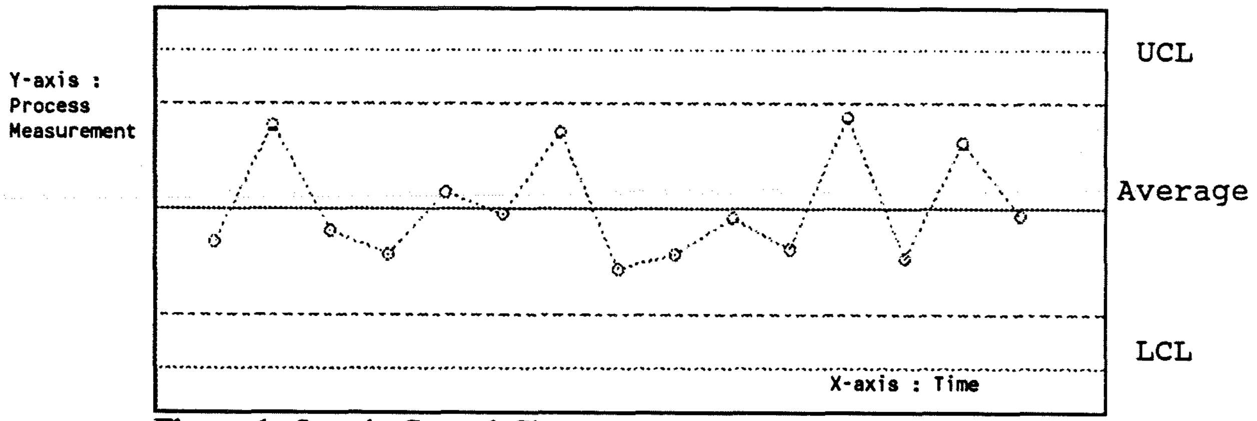


Figure 1. Sample Control Chart

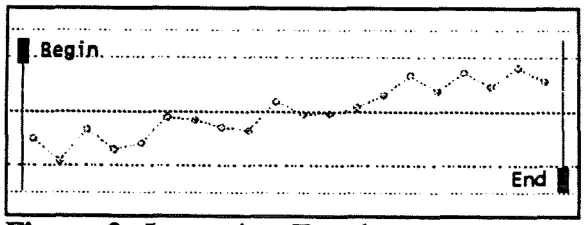


Figure 2. Increasing Trend

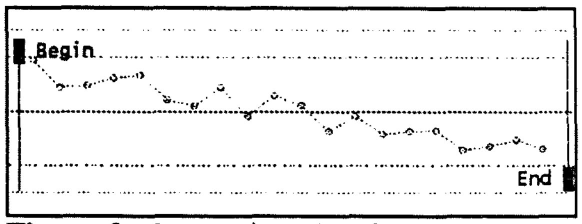


Figure 3. Decreasing Trend

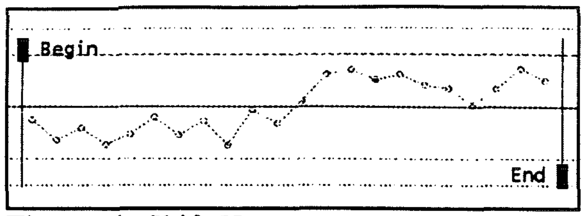


Figure 4. Shift Up

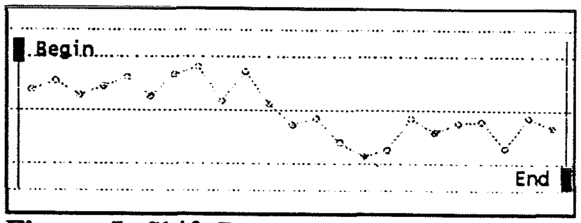


Figure 5. Shift Down

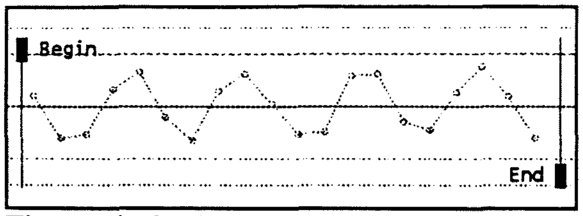


Figure 6. Cycle

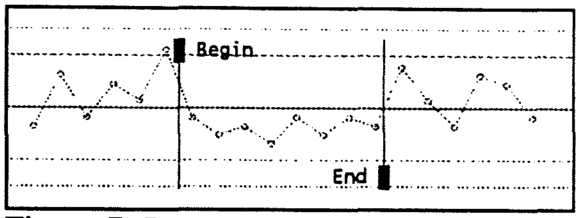


Figure 7. Run

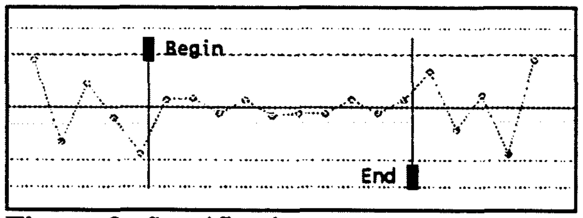


Figure 8. Stratification

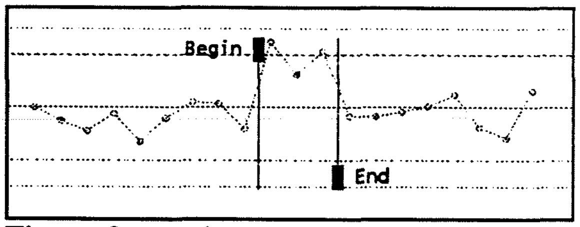


Figure 9. Freak Pattern

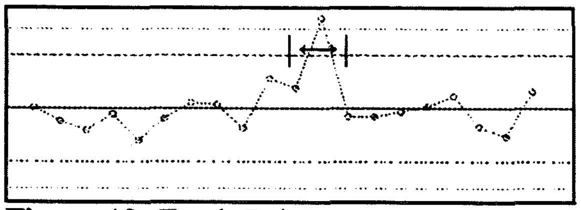


Figure 10. Freak Point

SOFTWARE FUNCTIONALITY

An overview of the functionality of the application (referred to as *SPC*) is given below :

(1) *SPC* determines which type of control chart is appropriate by asking a series of questions about the nature of the user's process data. The appropriate control chart is selected from the following types of charts (See References [3,4,5,6]) :

(a) X-Bar R Chart
(b) p Chart
(c) pn Chart
(d) u Chart
(e) c Chart

(2) *SPC* graphically displays the chart(s) selected in (1).

(3) *SPC* identifies the following patterns in the chart(s) which indicate the presence of assignable causes of variation :

(a) increasing trends
(b) decreasing trends
(c) shifts up
(d) shifts down
(e) cycles
(f) runs
(g) stratification
(h) freak patterns
(i) freak points

(4) *SPC* graphically displays and highlights each chart pattern identified in (3).

(5) *SPC* displays text in a window-like fashion which provides generic advice on the meaning of each chart pattern identified in (3).

SOFTWARE DESIGN

The basic approach to developing *SPC* was to integrate machine learning, expert systems, and conventional programming techniques. The machine learning portion of *SPC* was developed using the Abductory Induction Mechanism (AIM) by AbTECH Inc. The expert system portion of *SPC* was developed using an embedded application of the forward chaining expert system tool CLIPS along with a generic end-user interface also developed by the AISC. Turbo C++ was used as the conventional language into which the machine learning and expert system applications were embedded.

The task for the machine learning portion of *SPC* is to classify every sub-sequence of the control chart according to the presence or absence of five specific chart patterns : increasing trends, decreasing trends, shifts up, shifts down, and cycles. The remaining four chart patterns are identified by conventional methods.

The expert system is initially utilized to help the user select the appropriate type of control chart. This determination is based upon the type of data being collected and the constancy of the sample sizes.

Another function of the expert system is to interpret the classification results of the trained AIM Network. A control chart with 40 data points will generate over 600 classification results; with nine types of patterns this amounts to over 5500 individual pieces of classification information. This interpretation function represents an ideal expert system application. What requires a few hundred lines of difficult-to-comprehend C code can be implemented using an expert system with only three simple rules (**TABLE 9**)! This classification information is boiled down to about one to ten patterns which are reported to the final expert system application.

The final role of the expert system is to provide advice based upon the types of charts and the chart patterns present. The advice currently provided by SPC is of a generic nature. For example,

> *"A shift up in the R chart indicates that the process is becoming less consistent. This may be due to some sudden change in the process."*

However, the knowledge base is designed to allow for quick modifications to provide process specific advice. For example,

> *"A shift up in the R chart has historically been associated (90%) with a loose bearing in the preprocessing machine."*

Conventional software is used to graphically display the control charts, utilize the AIM Networks, provide an end-user interface, and integrate the entire application.

MACHINE LEARNING

Role Of Machine Learning

The task of chart interpretation can be summarized as follows. A control chart is simply a sequence or array of floating point numbers. The art of chart interpretation is to determine whether or not sub-sequences similar to several standard patterns are present within the chart. These patterns include trends, shifts, and cycles.

The function of the machine learning tool is to generate code (trained AIM Networks) which can effectively classify a specific sub-sequence of a control chart (array) according to the presence or absence of several standard patterns. With this classification function generated by machine learning techniques, all sub-sequences of the control chart are exhaustively (conventionally) classified by five AIM Networks. The AIM Network classification results are asserted into the fact-list of the CLIPS expert system application.

Justification For The Use Of Machine Learning Techniques

Machine learning techniques are used to classify five types of chart patterns - increasing trends, decreasing trends, shifts up, shifts down, and cycles. We could find no references which provide an algorithm for determining whether or not a sequence of real numbers is representative of one of these patterns. In fact, most references on control charts define these patterns by example! The most mathematical approaches to this problem are found in references [1,2] on time series analysis and forecasting. Despite being mathematical in nature, these references still do not describe a deterministic decision procedure. Rather, they provide mathematical heuristics. A sampling of these rules-of-thumb for a times series of length N are given below :

(1) The number of increasing steps in an increasing trend may be significantly larger than (N-1)/2.

(2) The number of discordances in a decreasing trend is usually larger than the expected number of discordances in a random sequence which is N*(N-1)/4.

(3) The autocorrelation coefficient sequence of a cycle is usually cyclic.

(4) The average of the first half of a shift down is always greater than the average of the second half.

Notice that most of these heuristics are in the form of rules with confidence factors. This would seem to suggest the possibility of using a production system for the classification procedure. However, it is almost always the case that the pattern-type (the attribute for which we wish to determine a value) is on the left-hand side of the rule.

This is very similar to some medical diagnosis problems whose domain knowledge is in the form "*Disorder A usually causes symptoms 1, 3, & 4 and may cause symptom 2.*" In cases such as these, the best knowledge-based approach is to use some form of a Hypothesize-and-Test (HT) model. Although the HT approach appears to model the domain very well, we did not pursue this option for the following reasons :

(1) We do not have a Hypothesize-and-Test knowledge-based development tool available for use.

(2) To my knowledge, there are no HT systems which can be embedded into an application in a manner similar to CLIPS.

(3) The HT knowledge-based system approach involves the solution of a minimal covering problem. This would probability cause the classification process to be unacceptably slow.

Attempting to implement such applications using a rule-based system with confidence factors ultimately boils down to an iterative process of re-adjusting confidence factors and re-testing the rule base on a set of examples. This iterative process, however, is quite analogous to the process of training a neural network or a machine learning tool on a set of examples. Given this analysis and the fact that most references on control charts define these patterns by example, we elected to implement a portion of the classification process using a machine learning tool.

Representation Of Control Chart Sub-sequence

The function of the machine learning tool is to classify a specific sub-sequence of a control chart according to the presence or absence of several standard patterns. A key question relating to the use of machine learning tools, is how do we represent an arbitrary length sub-sequence of an arbitrary length sequence of numbers as a fixed length vector of real numbers. The approach is to represent a sub-sequence of a control chart as a fixed length vector of statistical features.

Twenty (20) statistical features are extracted from each sub-sequence X[1..N] under consideration. Features 1 - 10 are raw statistical features while features 11 - 20 are Boolean type indicator variables. The features and their definitions are listed in **ATTACHMENT B.**

Training And Test Sets For Machine Learning Tool

Over 70,000 sample chart sub-sequences were generated to train and test the AIM Networks. Most of these sub-sequences were generated by adding random noise to existing control charts with existing patterns. Each chart sub-sequence generated a training/test vector of dimension 25 - 20 real-valued Network inputs (statistical features) and 5 bi-polar (-1 or 1) outputs. One AIM Network was trained for each of the 5 outputs. Each AIM Network required from two to six hours to train on a 386 machine with math co-processor.

Machine Learning Test Results

The results of the AIM Networks applied to control chart patterns not present in the training sets is presented below in **TABLES 1 - 5.**

Actual Pattern	total	# Class. as Inc Trend	# Class. as Not Inc Trd	% Correct
inc trd	1596	1572	24	98.5
not inc	1066	44	1022	95.9
Overall	2662			97.5

TABLE 1. Test Data Set Results For Increasing Trend Network

Actual Pattern	total	# Class. as Dec Trend	# Class. as Not Dec Trd	% Correct
dec trd	1605	1568	37	97.7
not dec	1058	35	1023	96.7
Overall	2663			97.3

TABLE 2. Test Data Set Results For Decreasing Trend Network

Actual Pattern	total	# Class. as Shift Up	# Class. as Not Shft Up	% Correct
Shft Up	789	775	14	98.2
Not SU	1201	10	1191	99.2
Overall	1990			98.8

TABLE 3. Test Data Set Results For Shift Up Network

Actual Pattern	total	# Class. as Shift Down	# Class. as Not Shft Dn	% Correct
Shft Dn	789	774	14	98.1
Not SD	1201	10	1191	99.2
Overall	1990			98.8

TABLE 4. Test Data Set Results For Shift Down Network

Actual Pattern	total	# Class. as Cycle	# Class. as Not Cycle	% Correct
Cycle	11826	10502	1324	89.8 *
Not Cyc	10666	410	10256	96.0
Overall	22492			92.0

* Most (99%) of these errors occurred in the short saw-toothed patterns with added noise. If the saw-toothed pattern is deemed by experts/customers to be very important, a separate network could be developed for the saw-toothed pattern. This would increase the overall cyclic % correct to about 96% and provide a better recognition rate for noisy, short saw-toothed patterns.

TABLE 5. Test Data Set Results For Cycle Network

EXPERT SYSTEM

Role Of Expert System

The role of the expert system in *SPC* is three-fold. One knowledge base helps the user select the type of control chart to be used, another interprets the AIM Networks' classification results, and the third knowledge base provides expert advice on the meaning of any identified patterns.

Selecting Appropriate Control Chart Type

The knowledge base for this portion of the expert system application in *SPC* is given below in **TABLE 7**. In short, the type of control chart is selected based upon (1) whether the data is attribute data or measurement data, (2) whether the logical group size is constant or variable, and (3) whether the (attribute) data is measuring defectives or defects.

Interpreting AIM Network Classification Results

A major issue during the development of *SPC* was how to interpret the AIM Networks' classification results. An example of a portion of the results of the AIM Networks' classification during the exhaustive conventional search is given in **TABLE 6**. The classification results of the AIM Networks are asserted into the CLIPS fact-list in the format :

(chart-type pattern-type begin-index end-index network-score).

```
( X cycle 1 13 0.743 )
( X inc_trend 5 17 0.098 )
( X shift_up 6 18 0.282 )
( X inc_trend 6 17 0.819 )
( X inc_trend 6 16 1.000 )
( X inc_trend 7 17 0.829 )
( X inc_trend 6 15 1.000 )
( X inc_trend 7 16 0.874 )
( X inc_trend 6 14 0.991 )
( X inc_trend 7 15 1.000 )
( X inc_trend 6 13 0.951 )
( X inc_trend 7 14 1.000 )
( X inc_trend 8 15 0.973 )
( X inc_trend 6 12 0.807 )
( X inc_trend 7 13 0.997 )
( X inc_trend 8 14 0.961 )
( X inc_trend 9 15 0.841 )
( X inc_trend 7 12 0.904 )
( X inc_trend 10 15 0.917 )
( X inc_trend 10 14 0.895 )
```

TABLE 6. Sample CLIPS Fact-List Generated By AIM Networks

```
(defrule data_type
   (initial-fact)
=>
   (ask_question "question.idx" "get_type" "data_type") )*

(defrule XBAR_R_Chart
   (data_type value)
=>
   (assert (chart_type XBAR_R)) )

(defrule group_size
   (data_type attribute)
=>
   (ask_question "question.idx" "get_size" "group_size")*
   (ask_question "question.idx" "get_att_type" "attribute") )*

(defrule PN_Chart
   (group_size constant)
   (attribute defectives)
=>
   (assert (chart_type PN_Chart)) )

(defrule C_Chart
   (group_size constant)
   (attribute defects)
=>
   (assert (chart_type C_Chart)) )

(defrule P_Chart
   (group_size variable)
   (attribute defectives)
=>
   (assert (chart_type P_Chart)) )

(defrule _Chart
   (group_size variable)
   (attribute defects)
=>
   (assert (chart_type U_Chart)) )
```

*The function <u>ask question</u> is provided by the CLIPS Application User Interface (AUI) also developed by the AISC at Wright-Patterson AFB, Ohio.

TABLE 7. Knowledge Base To Select Chart Type

Notice in **TABLE 6** that from points 6 to 17 there are 17 sub-sequences which the AIM increasing trend Network gave high scores to! Clearly we cannot report to the user all 17 patterns. The expert system application which interprets the AIM Networks' classification results is composed of three rules :

(1) The first rule eliminates from consideration any pattern whose AIM Network score is below a certain threshold. The sensitivity of the pattern recognition can be adjusted by altering these thresholds in the deffacts statement.

(2) The second rule eliminates from consideration any pattern which is contained entirely within another existing pattern of the same type. It is assumed that the first rule has previously been applied. For example, the fact *(X inc_trend 8 14 0.961)* would be retracted due to the presence of the fact *(X inc_trend 6 16 1.0)*.

(3) The third rule eliminates from consideration any pattern which overlaps another existing pattern of the same type but with a higher AIM Network score. It is assumed that the first two rules have previously been applied. For example, this rule would retract the fact *(X inc_trend 8 14 0.961)* due to the presence of the fact *(X inc_trend 7 13 0.997)*.

Expert Advice On Meaning Of Chart Patterns

The majority of the expert system interaction that the user will see involves explanations and advice regarding any patterns that the AIM Networks have identified as indicators of assignable causes of variation. At the most basic level, this expert knowledge simply consists of triples of the form <*chart-type, pattern-type, advice-text*>. The current AISC SPC software consists of knowledge at this level of complexity only. A sample of the CLIPS implementation of such knowledge is illustrated in **TABLE 8.**

However, the rule-based representation is justified for the following reasons :

(1) The interpretation of control charts with multiple patterns is more complex than simple chart-pattern-advice triples. The representation scheme must be powerful enough to accommodate future enhancements to the system.

(2) One requirement for the SPC software is that it be easily modifiable to process specific applications. Without knowing what type of reasoning process might be required for such customized applications, we selected the more flexible representation scheme provided by a production system.

```
(defrule R_shift_up
   (R shift_up ?a ?b ?score)
=>
   (write_paragraph "advice.idx" "R_shift_up") )*
```

*The function write paragraph is provided by the CLIPS Application User Interface (AUI) also developed by the AISC at Wright-Patterson AFB, Ohio.

TABLE 8. Sample Rule To Provide Expert Advice

CONCLUSION

SPC is a good example of a hybrid system which integrates machine learning, expert system, and conventional programming techniques. It is a classic example of pattern recognition and is an excellent demonstration of problem representation techniques necessary when using machine learning or neural network tools.

```
(deffacts thresholds
   (threshold run 0.99)
   (threshold inc_trend 0.95)
   (threshold dec_trend 0.95)
   (threshold shift_up 0.95)
   (threshold shift_down 0.95)
   (threshold stratification 0.99)
   (threshold freak_point 0.99)
   (threshold freak_pattern 0.99)
   (threshold cycle 0.95) )


(defrule simple_threshold
   (resolve thresholds)
   ?pattern <- (?chart ?type ?a ?b ?score)
   (threshold ?type ?thresh)
   (test (< ?score ?thresh))
=>
   (retract ?pattern) )


(defrule subset
   (resolve subsets)
   (?chart ?type ?a1 ?b1 ?score1)
   ?subset_pattern <- (?chart ?type ?a2 ?b2 ?score2)
   (test (not (and (= ?a1 ?a2) (= ?b1 ?b2))))
   (test (and (<= ?a1 ?a2) (<= ?b2 ?b1) ))
=>
   (retract ?subset_pattern) )


(defrule overlap
   (resolve overlap)
   (?chart ?type ?a1 ?b1 ?score1)
   ?pattern2 <- (?chart ?type ?a2 ?b2 ?score2)
   (test (not (and (= ?a1 ?a2) (= ?b1 ?b2))))
   (test (>= ?score1 ?score2))
   (test (or (and (<= ?a1 ?a2 ?b1) (< ?b1 ?b2) )
             (and (<= ?a2 ?a1 ?b2) (< ?b2 ?b1) ) ))
=>
   (retract ?pattern2) )
```
TABLE 9. Knowledge Base To Interpret Classification Results

Two features distinguish *SPC* from most other control chart software :

> (1) *SPC* automatically identifies and highlights unusual chart patterns. Most related commercial software simply draws the chart and explains to the user what unusual patterns to look for. We found no commercial software which automatically identified trends, shifts, or cycles.

(2) *SPC* provides expert advice on the meaning of all identified unusual chart patterns. Over 50% of available commercial software only construct the control chart for the user and go no further.

The first version of *SPC* is scheduled to be available by September 1991 and will be distributed with an AFLC sponsored course on Statistical Process Control. The AISC plans to provide software enhancements to *SPC* based upon future customer feedback and demand. Also, the AISC hopes to provide some customers with customized versions of *SPC* for process specific applications. Copies of *SPC* and reprints of this paper are available to government agencies upon request.

REFERENCES

[1] Spyros Makridakis and Stephen C. Wheelwright, "Forecasting: Methods and Applications",Wiley/Hamilton, 1978.

[2] Sir Maurice Kendall and J Keith Ord, "Time Series", Oxford University Press, 1990.

[3] SPC Course Materials, Decision Dynamics Inc., 1990

[4] Kaoru Ishikawa, "Guide to Quality Control", Asian Productivity Organization, 1982.

[5] Perry Johnson Inc., "SPC Chart Interpretation", Perry Johnson, Inc., 1987.

[6] J.M. Juran, Dr. Frank M. Gryna, Jr., and R.S. Bingham, Jr., "Quality Control Handbook",Third Edition, McGraw-Hill, 1974.

[7] Western Electric Company, "Statistical Quality Control Handbook", Western Ellectric Co., Inc., 1958.

[8] H. Besterfield, "Quality Control", Second Edition, Prentice-Hall.

[9] Douglas C. Montogomery, "Introduction to Statistical Quality Control".

ATTACHMENT A

Patterns To Be Identified And Methods Of Identification

(1) Freak Point - This is any point which falls outside of the three sigma control limits. This is conventionally identified.

(2) Freak Pattern - This is any sequence of points for which a large percentage fall more than a given amount away from the mean. This definition is vague since many experts and source materials disagree on what conditions to use. This is conventionally identified. The following criteria are used to identify a freak pattern:

(a) Two out of three points in a row outside of the 2 sigma limits. Reference [3].
(b) Four out of five points in a row outside of the 1 sigma limits. Reference [3].

(3) Stratification - Sometimes referred to as "hugging the center line." This is any sequence of points for which a large percentage fall less than a given amount away from the mean. This definition is vague since many experts and source materials disagree on what conditions to use. This is conventionally identified. The following criteria are used to identify a stratification pattern:

(a) Ten or more points in a row which are within the 1 sigma limits.

(4) Runs - This is any sequence of points for which a large percentage fall on the same side of the mean. This definition is vague since many experts and source materials disagree on what conditions to use. This is conventionally identified. The following criteria are used to identify a freak pattern:

(a) More than 5 (some say 7 and others say 8) points in a row on the same side of the mean.

(b) Ten of 12 on the same side of the mean.

(5) Increasing Trends - This pattern is identified with C code generated by the machine learning tool AIM. *Current accuracy is 97.5% based upon a test set of 2662 patterns.

(6) Decreasing Trends - This pattern is identified with C code generated by the machine learning tool AIM. *Current accuracy is 97.3% based upon a test set of 2663 patterns.

(7) Shifts Up - This pattern is identified with C code generated by the machine learning tool AIM. *Current accuracy is 98.8% based upon a test set of 1990 patterns.

(8) Shifts Down - This pattern is identified with C code generated by the machine learning tool AIM. *Current accuracy is 98.8% based upon a test set of 1990 patterns.

(9) Cycles - This pattern is identified with C code generated by the machine learning tool AIM. *Current accuracy is 92.0% based upon a test set of 22492 patterns.

* For further details, see Machine Learning Results.

ATTACHMENT B

Statistical Features Used To Represent Chart Subsequences

(1) RMS_SU - This is the root-mean-squared difference between X[1..N] and an "ideal" shift-up pattern.

(2) RMS_SD - This is the root-mean-squared difference between X[1..N] and an "ideal" shift-down pattern.

(3) A - This is the simple linear regression coefficient when trying to approximate the time series X[t] using X[t] = A + Bt.

(4) B - This is the simple linear regression coefficient when trying to approximate the time series X[t] using X[t] = A + Bt.

(5) SIGMA_1 - This is the standard deviation of the first half X[1..N/2] of the sequence X[1..N].

(6) SIGMA_2 - This is the standard deviation of the second half X[N/2+1..N] of the sequence X[1..N].

(7) R_root_N_r - The percentage of the first N/4+1 autocorrelation coefficients r(k) for which abs(r(k)) > 1.96/sqrt(N).

(8) CHI_SQ_TEST - This is the Box-Pierce Q-statistic which is capable of determining whether several autocorrelation coefficients are significantly different from zero. This is defined in reference [1,p 269]

(9) CONCORD - This is the number of concordances Q in X[1..N] divided by the maximum possible number N(N-1)/2 of concordances. This is defined in reference [2,pp 21-23].

(10) DISCORD - This is the number of discordances P in X[1..N] divided by the maximum possible number N(N-1)/2 of discordances. This is defined in reference [2,pp 21-23].

(11) TEN_PLUS - An indicator variable used to indicate if X[1..N] has length less than ten. This is important since many statistical significance tests are ineffective for small sample sizes.

(12) CCRD_LOW - An indicator variable used to indicate whether CONCORD is less than 0.7. The value of 0.7 was chosen since a database analysis indicated that a high percentage of increasing trends had CONCORD > 0.7.

(13) DCRD_LOW - An indicator variable used to indicate whether DISCORD is less than 0.7. The value of 0.7 was chosen since a database analysis indicated that a high percentage of decreasing trends had DISCORD > 0.7.

(14) HIGH_ISD - An indicator variable used to indicate whether RMS_SD is greater than 1.8. The value of 1.8 was chosen since a database analysis indicated that a high percentage of shifts-up had RMS_SD > 1.8.

(15) HIGH_ISU - An indicator variable used to indicate whether RMS_SU is greater than 1.8. The value of 1.8 was chosen since a database analysis indicated that a high percentage of shifts-down had RMS_SU > 1.8.

(16) GOOD_INC_MM - An indicator variable used to indicate when the sequence minimum was early and the sequence maximum was late. The first 20% and last 20% was chosen since a database analysis indicated that a high percentage of increasing trends had their minimum and maximum within the first 20% and last 20% respectively of the sequence.

(17) GOOD_DEC_MM - An indicator variable used to indicate when the sequence maximum was early and the sequence minimum was late. The first 20% and last 20% was chosen since a database analysis indicated that a high percentage of decreasing trends had their maximum and minimum within the first 20% and last 20% respectively of the sequence.

(18) HIGH_R_root_N - An indicator variable used to indicate whether R_root_N_r is greater than 0.1. The object of introducing this variable was to help draw a distinction between random sequences and cycles. The value of 0.1 was chosen since a database analysis indicated that a high percentage of cycles and a low percentage of random sequences had R_root_N_r > 0.1.

(19) SMALL_A - An indicator variable used to indicate whether the absolute value of A is less than 0.8. The object of introducing this variable was to help draw a distinction between random sequences or cycles and the other chart patterns. The value of 0.8 was chosen since a database analysis indicated that a high percentage of cycles and random sequences and a low percentage of other types of patterns had abs(A) < 0.8.

(20) MAYBE_CYCLE - An indicator variable used to indicate when both R_root_N_r > 0.1 and ABS(A) < 0.8. This is the logical AND of variables 18 and 19.

S12-61

57692

N92-16580

p-8

Application of Software Technology to Automatic Test Data Analysis

JJ 574450

J.R. Stagner

Jet Propulsion Laboratory
California Institute of Technology

Abstract. The verification process for a major software subsystem was partially automated as part of a feasibility demonstration. The methods employed are generally useful and applicable to other types of subsystems. The effort resulted in substantial savings in test engineer analysis time and offers a method for inclusion of automatic verification as part of regression testing.

INTRODUCTION

One area of interest for the application of new software technologies is the automation of labor intensive functions performed as part of the overall testing process. A specific problem area is analysis of the results from test runs to verify correctness of software under test. Currently, most analysis of data resulting from test runs is performed manually. Data from one test run can take up to 60 hours to analyze. Hence, this is a candidate problem area for machine-aided or automatic analysis. The work, described below, was performed as a proof of concept experiment to demonstrate the feasibility and usefulness of rapid prototyping and production system programming techniques in the solution of this recurrent class of test problem. The expected benefits are to conserve test resources and improve the quality and thoroughness of test analysis.

The software selected for this proof of concept experiment was the Data Monitor and Display (DMD) subsystem (Figure-1.), which is part of the new Space Flight Operations Center (SFOC) under development at JPL. The SFOC is a multi-mission ground data system. The DMD is an end-point in the Ground Data System telemetry data flow and provides visibility into the data processing. It's major input is channelized telemetry data (TLM). It's major outputs are displays of the data in human readable form, including a hard-copy dump called the Latest Available Data (LAD). The DMD makes use of several tables, contained within the Channel Parameter Table (CPT), coefficient table (COEF), and

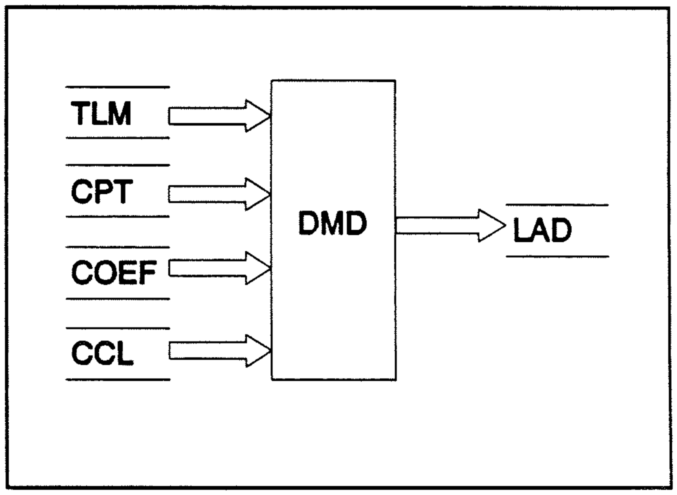


Figure 1. DMD Test Run Data Flow

Channel Conversion Language (CCL) files.

The raw telemetry data (TLM) are in Data Number (DN) units. Data numbers are converted to Engineering Units (EU) by means of a table lookup or polynomial computation. Conversion tables and polynomial coefficients are contained in the COEF file.

The CPT contains, for each channel, the data type (integer, unsigned integer, floating point, status, ASCII, digital data, etc.), the subsystem where it originated (usually a spacecraft subsystem) and any limit tests that are to be performed.

CCL is a language that specifies processing for each channel. Some examples would include channels derived by (1) averaging three other channels, or (2) taking the arc-cosine of a channel and adding a constant factor or (3) multiplying a channel by a constant to convert from radians to degrees.

The LAD contains a dump of raw telemetry (input) DN, DMD computed EU values and alarm states based on DN and information contained in the CPT, COEF and CCL files. This information is provided over a time window established by the test engineer in the data collection process.

To verify that the DMD correctly computed an EU value for a given channel, the CCL listing must be examined to determine if any processing has been applied. Relevant information from the COEF listing is used to manually compute the EU value so it can be compared with the DMD computed value contained in the LAD listing. Similarly, the alarm state indicators in the LAD data must be verified by manually applying the alarm limit criteria found in the CPT listing. The problem of verifying the approximately 3400 data channels, for just one mission (Magellan), becomes apparent.

Manual analysis of test results is usually a straight-forward but laborious task. We would like to have a computer program that does what the test engineer (TE) does. That is, stand in the place of the TE and do the analysis. We would expect it to perform, in minutes, what the TE would do in days and the results would be more accurate and of higher quality. It would be economically feasible to systematically analyze all the data channels, for every software delivery throughout the software lifecycle, instead of using a sampling technique.

A program like that needs the TE's knowledge about all of the various elements that he checks when he analyzes data channels. In the DMD, for example, the analysis program must be able to read the electronic versions of the various listings, determine how to convert digital numbers (DN) to engineering units (EU) and, given an EU, how to determine the alarm state. With this knowledge and supporting facts about alarm limit types and values and coefficients or tables to be used in the DN to EU computations, the program should be able to examine output from DMD, channel by channel, determine what DMD thinks the EU and alarm states are, decide if the TE would agree with the answer produced by DMD and signal the results of this analysis somehow.

PROGRAMMING TOOLS

The programming tools used in this work were AWK, CLIPS and UNIX. AWK (Aho, et al. 1988) is a programming language that is well suited for exploratory programming. AWK syntax is close to C and provides simplified program control, I/O, character string functions, and regular expressions, which make AWK very useful for low-level data filter operations and report generation.

The "C" Language Production System (CLIPS-4.3) (Giarantino 1989) was developed by Johnson Space Center (JSC) and is the expert system shell used for this task. It provides a forward-chaining inference engine and interactive development environment in support of the prototyping, execution and debugging of knowledge bases. CLIPS syntax is similar to other production systems (e.g. OPS5, OPS83, ART). Interactive production system programming techniques supported by CLIPS encourage the acquisition of knowledge about the analyses and sub-problems in manageable units which are encoded in the form of rules. Each rule is easily modifiable, immediately executable and its effects on the analysis can be quickly evaluated. The turn-around time from concept to evaluation is quite rapid. Since rules execute opportunistically, control is hidden and knowledge is explicit. Therefore, rules are generally easier to understand than an equivalent "C" program where program control must also be made explicit within the code.

A significant benefit of both AWK and CLIPS is portability between PC's and UNIX work-stations. Much development was performed on a PC/AT (MSDOS 3.3). The AWK scripts and CLIPS knowledge bases were subsequently ported to the host computer. For some general guidelines on the types of problems suitable for solution using a production system, see (Brownston, et al. 1986) pp 19-29.

The UNIX Cshell (Sobell 1985) was used to integrate all of the AWK data filter and CLIPS analysis functions to define the end-to-end processing task (Figure-3). Cshell provides an impressively simple, high-level and flexible mechanism for integrating CLIPS into a traditional programming environment to perform an analysis task. It also provides many other useful services, in a simplified fashion compared to an equivalent C program. For example, tests were performed to verify that the input files physically exist before the analysis is started. Timing information is displayed for performance measurements. The DN-EU and alarm-limits analyses are forked as separate sub-processes to run in parallel and take advantage of input/output overlap. The entire analysis process can be run in the background, thus clearing the terminal for other work.

APPROACH

Problem Definition

Initially, the problem was ill-defined. The number of analyses was fixed at two, but the number of sub-problems was unknown. This characteristic points to an evolutionary programming approach. Knowledge acquisition was performed by means of an interview, prototype and evaluate cycle. Incremental improvements in the analysis capabilities were made at each cycle. This process quickly produced an end-to-end analysis capability, for a large subset of data types.

The DMD TE was interviewed to gain an understanding of the contents and format of the data to be analyzed and the steps he uses in verifying that the results are correct. A prototype was developed using the AWK and CLIPS programming languages (described below) to create a set of functions that perform the analyses so that the TE could see some results. This initial prototype was built around the CLIPS expert system shell. Thus the initial prototype clarified functional issues and was highly interactive and programmer-oriented.

From this initial prototype, it was learned that the TE really wanted a UNIX command-line function that could be used to analyze multiple sets of input files. Therefore, the final prototype addressed encapsulation issues and became user-oriented, taking into account how the TE wanted to perform actual work. That is, operational details were hidden in order to simplify the user interface.

Merged Data Base

In order to help the test engineer articulate the steps he performs during manual analysis of data, a tool was written (Figure-2) which merged all of the information for each of the given channels from the diverse inputs. The goal of this tool was to provide a displayable database with information clustering and minimal filtering.

The merged database product is probably the best way to begin the knowledge acquisition process for this kind of problem. Bringing all of the relevant information together at a single point is requisite for any kind of analysis program. It also helped the test engineer organize his thinking about how he analyzes channels and avoids references to four separate stacks of printout, a time consuming process. All of the relevant information is collected together in a single display. This minimally filtered product facilitated the discovery process of ways to partition the overall analysis problem into sub-problems and their associated analysis technique. We were able to map these analysis sub-problems into software modules implemented either by AWK programs or CLIPS rule bases.

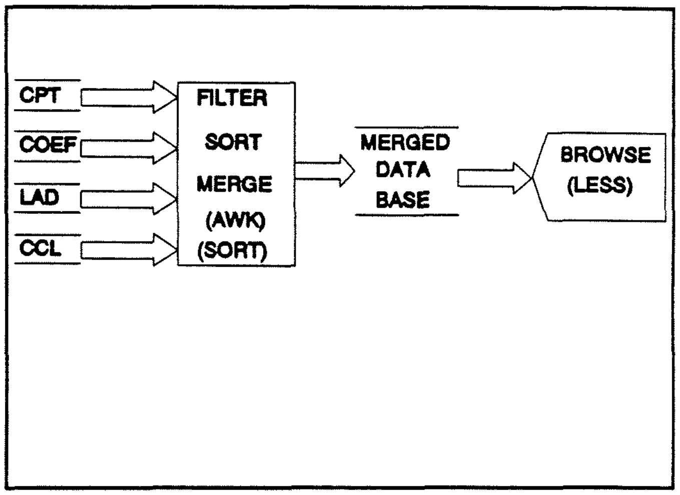


Figure 2. Tool Assisted Test Results Analysis Data Flow

The merged database product proved not only to be useful for gaining insight into the analysis process, but it also provided a dramatic improvement in manual analysis productivity. It is estimated that the manual analysis of 1000 channels previously took 20 man-hours. Using this reformatted data set, manual analysis could be performed in about 4-hours. Improvement in productivity was achieved by consolidation of information between the four different listings and the clustering of information distributed within the listings.

Analysis Data Flow

Figure-3 shows a Data Flow Diagram for the final analysis prototype. Although the DMD data listings are human readable, the formatting did not allow for easy detection of problems by humans. It is easy to miss problems buried in many pages of printout. Embedded character strings add clutter which make it even more difficult to notice anomalies in large volume printouts.

The filter program was designed to automatically sense the type of input and format the relevant data from the CPT and COEF files as CLIPS-world data structures (fact-lists). The relevant LAD data are output as ordinary lines of data to be read by CLIPS, one at a time, and analyzed.

The number of CLIPS-world records are counted for comparison with a count of input COEF, CPT and LAD records. This provides a check to verify that no data were lost in the filtering process. The CPT fact-list and alarm knowledge base are input to CLIPS as part of the initialization procedure for an alarm analysis. Similarly for the COEF fact-list and DN-EU knowledge base.

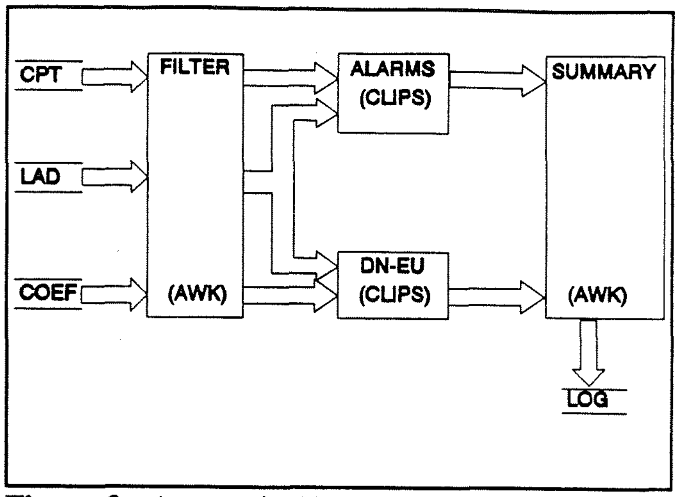


Figure 3. Automatic Test Results Analysis Data Flow

Filtering

Data filtering is a necessary part of the overall problem solution. Too little filtering allows too much bad or irrelevant material to pass and requires more analysis (human or otherwise) to validate the results. Too much filtering can obscure some valid results. The task of filter programs is similar to their electronic counter-parts, that is, to eliminate irrelevant information such as page headings, blank lines, unwanted keyword-like character strings, new-page characters, etc. They also convert multi-line records to a single line format in order to further simplify subsequent processing.

The format structures for CPT, COEFF and LAD data, were similar since they were computer generated. That is, single or multiple-line records with interspersed page headings, footers, and short, blank or null lines. The fields within records sometimes consisted of character string data identifiers, like "RED" for red-alarms and "HI" for the upper alarm limit etc., followed by an equal sign and the data-item itself. Some data items are given as a list, where the list items are separated by a comma or white space delimiters. It should be noted that the test engineer has considerable flexibility in formatting these outputs. For example, he can choose the data-item mnemonic, spelling and case, or none at all.

The filters described below were "tuned" for the particular printout formatting employed by the DMD test engineer. Regular expressions were adequate for filtering these data. A better solution might have been to employ the inferencing capabilities of CLIPS to provide more flexible data-item specification and/or location and extraction.

The CCL specifies all data channel definitions and processing in a FORTRAN-like programming language. The optimum way to extract relevant information from this file is to use a parser. The author decided to avoid building a parser, as part of this initial effort, by focusing on non-derived data channels for which regular expressions could be used to extract the relevant information for analysis purposes. But parsing is a well understood type of knowledge and should be implementable in CLIPS. It is not clear (to the author) that a parser could have been developed faster with CLIPS than using the UNIX utilities lex and yacc. A parser implemented with CLIPS would probably be easier to maintain and would avoid having to use yet another language dialect to implement analysis related processing.

Knowledge Bases

The CLIPS knowledge bases contain the TE's knowledge about the various ways to compute EU from a knowledge of DN and how to determine the different types of alarm states. With this knowledge and supporting facts about actual alarm limit values and tables to be used in the DN to EU computations, the CLIPS analysis functions examine output from DMD, channel by channel, to determine what DMD thinks the EU and alarm states are and if the TE would agree that DMD produced the right answer.

This latter decision was implemented in a straight-forward mathematical way. CLIPS computed values are assumed to be correct. DMD and CLIPS computed values must agree to a certain level of precision in order to be acceptable. This rule takes into account differences in precision between printouts and the CLIPS internal representation of numbers. The great majority of channels tested were determined to be correct, but this precision criteria was good enough to catch the anomalies described below. If such an anomaly is found, a highly visible flag is set in the white space of the printout to catch the TE's attention should he decide to browse the output.

A post-process AWK function extracts those cases that have been flagged as described above and presents not only the analyzed output but also the corresponding information from the original CPT, LAD, and COEF input files. Although simple and unorthodox in implementation, this feature is a kind of explanation facility common to most expert systems. The TE has, in one convenient place, without having to flip through many pages of output, all of the supporting information as to why the test analysis tool thinks there is a problem. Now the TE can be the final authority, as to whether the anomaly is real and decide what action might be required.

Anomalies Detected

In exercising the above tools, some anomalies (unexpected behaviors) were detected. Two channels had both upper and lower alarm limit thresholds set to zero and their DN values were also zero. DMD and the analysis tool gave different alarm state answers for these channels. This reflects differences in the way the CLIPS analysis tool and the DMD handle pathological cases. Alarm limits were not yet established for some channels in the CPT database.

Another example is that a certain digital bit channel had an incorrect timetag and a wrong bit value. Using the merged database tool, it was discovered that this channel was not defined in the CCL file, but a value from some old data was present and displayed. At present it is not known if this behavior is a bug since the problem can be corrected by defining the channel in the CCL database. Thus, the DMD analysis tool caught certain problems which might have gone un-noticed.

CONCLUSIONS

We believe the methods described above can be extended to a wide range of processors. If this proves to be the case for all, or even a strategic subset of processors within a system, then we can make automatic test data analysis part of regression testing.

A simple tool to reformat the subsystem inputs and output into a merged display provided an unexpected level of productivity enhancement, compared to strictly manual verification. In retrospect, the reason is clear. Merging and clustering of information saved the test engineer's time in cross-referencing four separate and voluminous printouts. Hence, significant test results analysis quality and productivity can be achieved without full automation. In addition, this tool proved to be useful during the problem definition and knowledge acquisition phase of program development.

In creating the analysis tool, we did not attempt to explicitly duplicate functionality of the software under test. Instead, we tried to emulate operations the test engineer performs when he verifies the software. That is, information collection and verification that the various computations were done correctly.

The current analysis tool could have been implemented without an expert system shell. This was not known at the outset. However, analysis capability for derived channels has not yet been achieved, due to the complexity of language parsing. Parsing is a form of expertise and a CLIPS-based inferencing approach may prove to be a simpler solution to this problem than, for example, a "C" language lex/yacc solution.

Mechanisms exist for inclusion of expertise, such as parsing, but the complexity of incorporating knowledge of this type is unknown. The scope of the present effort did not allow for a reasonable evaluation of this. Encoding test analysis knowledge in the form of rules was easier to review and maintain compared to an equivalent "C" program.

We have tried to strike a balance between production system programming and conventional programming to achieve a useful tool for the automation of test results analysis for a particular subsystem. We used CLIPS to encapsulate the analysis particulars, AWK and sort for data clustering, sorting and merging and UNIX Cshell for integration. The boundary regarding which processing tasks are best done within CLIPS and those best performed by an external function is changing, because CLIPS is changing and becoming more powerful with each release.

This work demonstrated the general usefulness of CLIPS and the ease with which expert systems or other types of production system applications can be integrated into a traditional computing environment. CLIPS applications can be combined with UNIX utilities to perform processing tasks.

ACKNOWLEDGEMENTS

The author gratefully acknowledges the following individuals for their valuable contributions to this effort: D. Klemp, R. Wells, T. Kratz, P. Harmon, H. Avant and D. Hermsen.

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

Reference herein to any specific commercial product, process or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government or the Jet Propulsion Laboratory, California Institute of Technology.

REFERENCES

Aho, A., Kernighan, B. and Weinberger, P. (1988). *The AWK Programming Language*, Addison-Wesley, Reading.

Brownston, L., Farrell, R., Kant, E. and Martin, N. (1986). *Programming Expert Systems in OPS5: An Introduction to Rule-Based Programming*, Addison-Wesley, 1986.

Giarratano, J. (1989). *CLIPS Users Guide, Version 4.3 of CLIPS*, Artificial Intelligence Section, Lyndon B. Johnson Space Center.

Sobell, M. (1985). *A Practical Guide to UNIX System V*, The Benjamin-Cummings Publishing Co.

SESSION 3 B

PRECEDING PAGE BLANK NOT FILMED

N92-16581

Acquisition, Representation and Rule Generation for Procedural Knowledge

Chris Ortiz
Software Technology Branch/PT4, NASA/Johnson Space Center, Houston, Texas 77058.
(ortiz@gothamcity.jsc.nasa.gov)

Tim Saito
Computer Sciences Corporation, 16511 Space Center Boulevard - M30, Houston, Texas 77058, U.S.A.
(tsaito@nasamail.nasa.gov)

Sachin Mithal
Computer Sciences Corporation, 16511 Space Center Boulevard - M30, Houston, Texas 77058,
(sachin@gothamcity.jsc.nasa.gov)

R. Bowen Loftin*
Department of Natural Sciences, University of Houston-Downtown, One Main Street, Room 813-N, Houston, Texas 77002, U.S.A., and NASA/Johnson Space Center
(bloftin@nasamail.nasa.gov)

Historically knowledge acquisition has proven to be one of the greatest barriers to the development of intelligent systems. Current practices generally require lengthy interactions between the expert whose knowledge is to be captured and the knowledge engineer whose responsibility is to acquire and represent the expert's knowledge in a useful form. Although much research has been devoted to the development of methodologies and computer software to aid in the capture and representation of some types of knowledge, little attention has been devoted to procedural knowledge. NASA personnel, on the other hand, frequently perform tasks that are primarily procedural in nature.

This paper describes current research into the design and continuing development of a system for the acquisition of procedural knowledge, its representation in useful forms, and proposed methods for automated CLIPS rule generation. TARGET (Task Analysis and Rule Generation Tool) is intended to permit experts, individually or collectively, to visually describe and refine procedural tasks. The system is designed to represent the acquired knowledge in the form of graphical objects with the capability for generating production rules in CLIPS. The generated rules can then be integrated into applications such as NASA's ICAT (Intelligent Computer Aided Training) architecture. The paper concludes by describing proposed methods for use in translating the graphical and intermediate knowledge representations into CLIPS rules.

Systems such as TARGET have the potential to profoundly reduce the time, difficulties, and costs of developing knowledge-based systems for the performance of procedural tasks.

INTRODUCTION

Processes and software designed to aid knowledge acquisition can be characterized by the nature of their delivery and implementation methods and styles as well as their ability to extract knowledge. Various authoring tools have evolved to solve the problems associated with the creation of a specific expert system

* Address all inquiries to R. B. Loftin, Mail Code PT4, NASA/Johnson Space Center, Houston, TX 77058, (713) 483-8070, bloftin@nasamail.nasa.gov.

PRECEDING PAGE BLANK NOT FILMED

(Boose, 1989). Historically, most knowledge-acquisition-oriented tool designs were directed toward rating or categorizing problems or knowledge. To use such tools to capture specific knowledge, the developer distinguished between types of knowledge methods/approaches. Although sharing many of the same goals, the existing methodologies are numerous, ranging from frame modeling to case-based reasoning models to repertory-grid rating structures. The various knowledge types addressed by these systems—from semantic/taxonomic to declarative to procedural—affect the design and performance decisions of researchers and implementers (Gaines, 1988). Knowledge representations, including frames, objects, rules, and decision trees, are used to capture and execute expertise. At this point, most would agree that no one tool accommodates all of the cognitive styles needed to gather the information/knowledge necessary for the creation of an expert system in one contiguous process. It is clear that viable standards have yet to be fully established and accepted.

Procedural knowledge acquisition via task analysis is a reasonable candidate for graphical representation modes. Decomposing a complex set of steps that make up a specific mission or task requires cognitive visualization and the ability to formulate and reformulate the decomposition of those steps or actions. The specific heuristic procedures that most subject matter experts (SMEs) employ share certain levels of organization and recall (de Kleer, Doyle, Steele, Jr., and Sussman, 1985). The path in which a procedure evolves starts with specific agendas and goals. The last or final action of reaching or satisfying those actual goals would end the procedure. On the other hand, any actions that would restart a process (i.e., a loop) would occur before the goal-oriented or last action. Decisions may be made during a task that direct the expert along alternative paths that may or may not be taken in other performances of the same task. In cases where the processes offer one or more options to complete a task, the process diverges into as many paths as necessary to meet the optional requirements. Each path would then contain specific values for technique evaluation or other modes of feedback. These types of complexities lend themselves to representation in a visual form.

ENVIRONMENTAL FACTORS AT NASA

As in other environments, getting and maintaining an SME's attention, time, commitment, help, and data sources at NASA is usually difficult at best. SMEs tend to differ in their communication abilities and styles, willingness to cooperate, availability, and degree of computer literacy, potentially affecting the overall success of the knowledge acquisition process (Littman, 1988). The strategy of providing the SME with a tool that can be used to document his mission(s) or task(s), on his own and within his schedule, would serve to resolve some of the difficulties associated with a knowledge engineer constantly "hovering over" an SME. However, the disadvantages of such a strategy may include the lack of positive reinforcement or external motivation (i.e., SMEs might put off documenting their task/mission unless periodically reminded or encouraged).

Other constraints affecting the type of tools delivered and used could be those associated with budget problems or deficits. Where exotic workstations might be required for more sophisticated KA tools, NASA may only have dated or under-powered PC hardware in certain areas of need. Distribution of KA tools to NASA personnel who have access to inadequate equipment to use in documenting their procedures could prove frustrating. SMEs tend to use their PCs or MacIntoshes for spreadsheets, databases, or word processing where KA is not a daily issue in their operation. The need to deliver tools that do not require SMEs to alter their current work style and environment (hardware, software, operating systems, methods, etc.) is, therefore, critical.

THE CLIPS FACTOR

In the NASA/Johnson Space Center environment, CLIPS (C-Language Integrated Production System) is widely used as an expert system development and delivery vehicle. Within the ICAT metaphor the overall procedure or task is decomposed into sets of tasks/subtasks that are termed actions. For most effective use, actions are expressed, within reason, at the lowest possible level. At any point in an ICAT training session, the expert expects the trainee to perform some action. Each action, as defined by the expert, is represented as a CLIPS fact (Figure 1) in the following pattern:

(message-sender-to-receiver   <step number>   <action type)   <argument><argument> ...)

Figure 1. CLIPS Fact for ICAT Environment

An action itself comprises at least two <argument> fields that define one single action decomposed into a hierarchical structure of two or more subactions of the form (<action> <argument>). Each <argument> may itself be an <action> at the next lower level. For example, (arg1 arg2 ... argn) can be decomposed into at least two levels: (arg1 arg2) and (arg2 . . . argn). The first pair, (arg1 arg2), is an (<action> <argument>) pair at the top level where the action arg1 has one argument, arg2. In turn, (arg2 ... argn) is another (<action> <argument>) pair at the second level where arg2 has one or more arguments, depending on the value of n. The structure for each action may be different and the number of arguments that belong with each action is variable. The expert is free to decompose the actions and arguments into hierarchies that fit his or her specific domain.

This paper details the design and implementation of a knowledge acquisition system tailored to the acquisition and representation of procedural knowledge associated with the performance of complex tasks. The primary goal of this effort has been the production of a system with an easy-to-learn and "comfortable" user interface that provides powerful mechanisms for the visual expression of procedural knowledge. The ultimate goal of this work is the expression of acquired knowledge in the form of production rules to facilitate the use of the acquired knowledge in expert systems for mission support and training.

THE TARGET (Task Analysis/Rule GEneration Tool) APPROACH

TARGET and Design Strategy

Balancing non-programmer usability, design sophistication, and hardware portability requirements, the Task Analysis/Rule GEneration Tool (TARGET) is designed to provide a knowledge acquisition environment for users of commonly-available computer systems (IBM® PCs and Apple© Macintoshes®). The forte of TARGET is the gathering of task or procedural knowledge to be expressed and analyzed graphically as well as contextually. TARGET provides users the ability to graphically decompose a task or procedure using a box-flow presentation/manipulation style within a windowed environment.

TARGET is designed to let the SMEs start documenting their job or task with minimal training in its use and no absolute need for knowledge engineer intervention. If the SME is unable to find time to work on the knowledge acquisition process alone, TARGET does allow the knowledge engineer and SME to work together in iterative sessions. It is tailored to accommodate a wide range of users, from the novice to the expert. Users can develop a discrete representation of tasks and subtasks within their domains (Payne and Green, 1986). The system then manages the information entered and represents the knowledge in a "top-down" reporting format that can then be used for rule induction and generation.

In order to support the development of intelligent computer-aided training (ICAT) systems, TARGET implements its rule representations using the rule types and structures originally developed for ICAT systems. TARGET is designed to deal with a series of tasks/subtasks that are performed procedurally or in steps. TARGET supports three action types: required, optional and flexible (as defined in the ICAT design architecture). Steps are defined as the progression from one required action to the next. Steps are represented by numerals and their values increase with the progression of the task.

TARGET User Interface

TARGET maintains a fragile balance between ease of use and design complexity/intricacy. Although it does not possess the "bells and whistles" of more sophisticated systems like Aquinas and Protege, TARGET provides enough knowledge modeling (procedural/declarative) functionality to allow the SME or knowledge engineer to build a moderately elaborate knowledge base without sacrificing the attractiveness of its user interface (Figure 2).

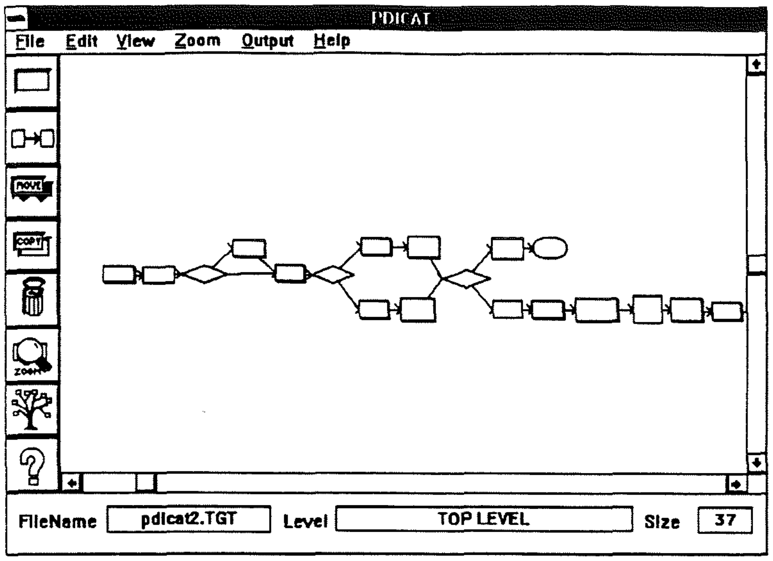

PDICAT
File Edit View Zoom Output Help
FileName pdicat2.TGT Level TOP LEVEL Size 37

Figure 2. TARGET Interface

TARGET provides a windowed environment through which decomposition can be organized and recorded. Ultimately the user, knowledge engineer or SME, is responsible for the overall quality checking of the knowledge base before its representation in or transfer to other applications. TARGET's report facilities offer some assistance in this quality checking process. Reports can be generated to provide moderately high-level feedback to the knowledge engineer and SME. TARGET produces the following reports:

- Task hierarchy: keeps a sequential/hierarchical account of tasks
- User Scratch Pad: keeps notes on conditions, states or other user-supplied details.

TARGET supports the identification and conceptualization phases of knowledge acquisition with its network approach to knowledge representation. Duties, tasks/subtasks, or steps/substeps within a process can be defined, documented, and structured to reflect these relationships to other duties, tasks/subtasks, or steps/substeps.

Given its developmental state, TARGET provides a reasonably comprehensive mechanism for generating simple representations at the very first knowledge acquisition session. The next sessions may be used to embellish what has already been elicited or to create new or modified versions of the knowledge base. The TARGET knowledge acquisition interface gives the user the freedom to generate as complex a hierarchy of knowledge as necessary. However, the disadvantage to such freedom is the ability to create a completely abstract knowledge base with relatively few standards for input. Some guiding controls from the TARGET interface could provide structure to the knowledge acquisition process and greatly enhance the ability of the user to create a "useful" knowledge base.

Task-Action Concepts (Building Blocks)

TARGET employs a free-form flow charting strategy. Users can explain procedural processes by the use of various flow chart icons manipulated in the work area. Tasks can then be linked together using directed arcs to represent procedural flow.

The task icons are separated into five major categories. The shape of the task box is an important key to determining its function. TARGET also works reasonably well on monochrome systems with the combination of shapes and colors.

The first category consists of the actions that are required to complete a process (Figure 3). The required actions are denoted by using a blue rectangular box on the screen. Likewise, optional tasks are represented using grey rectangles. The use of color representing various tasks has been reduced to just two, grey for optional tasks and blue for all others.

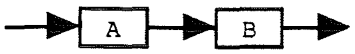


Figure 3. Required Tasks

A third task type, a hexagonal shaped box, shows where decisions are made. The connection labels reflect the choices allowed. For example, a decision box may ask if a process has been completed. This decision may branch to two other paths (Figure 4). The first arc leaving the decision may be labeled "YES" and the other labeled "NO". The arc labels represent the only possible choices allowed by the decision point. Decisions are not limited to binary operations but can have many unique arcs as necessary. The minimum number of arcs from a decision task is two.

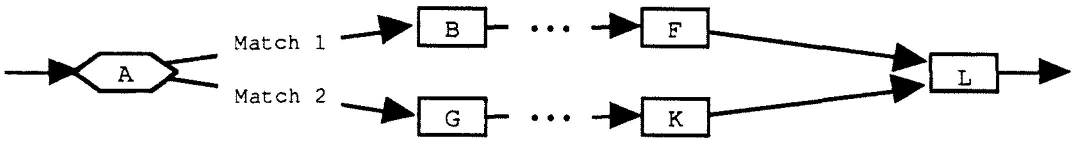


Figure 4. Decision Task

The fourth task type is a control structure. Control structures are used as a "go-to" or looping mechanism. They are ellipse-shaped to distinguish them from other tasks (Figure 5). Controls can only jump to other tasks that are on the current layer (see discussion of layers below). However, they may not jump directly to other controls. TARGET ensures that an endless loop can not exist. In addition, the control task cannot be further decomposed. TARGET will also prevent any changes to a task which a control structure is pointing to.

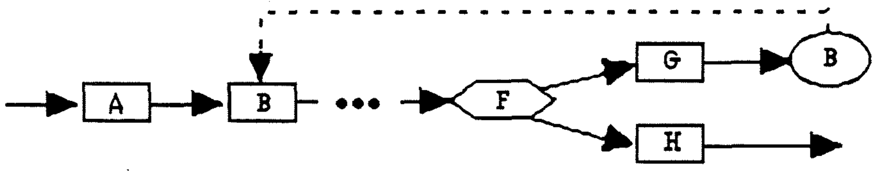


Figure 5. Control Structure

The fifth structure is a goal or an end of a process. Goal boxes are used to terminate the process or procedures. Goal tasks are denoted by using an rectangle with thick borders to distinguish them from other tasks (Figure 6). TARGET enforces certain rules regarding goals that cannot be further decomposed and may not have links originating from them.

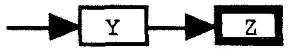


Figure. 6 Goal Structure

TARGET attempts to address the issue of parallel tasks whrere two or more sets of tasks are performed simultaneously. They are not assigned task shapes but are created whenever two or more arcs emanate from a rectangular box. Figure 7 shows a parallel task configuration. Each task chain represents a parallel process which will be executed independently until the paths converge.

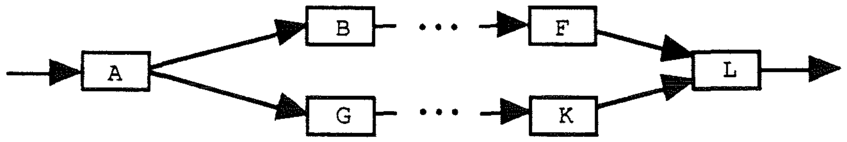
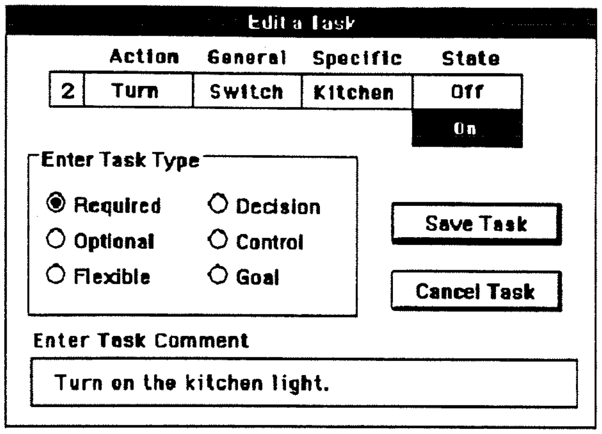


Fig. 7. Parallel Tasks

TARGET provides users with the ability to decompose various tasks into lower level tasks. Mouse manipulation makes navigating through task hierarchies fairly simple.

Example: Creating a Task

The following example will create a task which will activate a kitchen light. After selecting the create/edit icon from the toolbox and clicking on an open area in the user interface, a dialog box above will appear (Figure 8).



Edit a Task

	Action	General	Specific	State
2	Turn	Switch	Kitchen	Off
				On

Enter Task Type

[✓ Required] Decision
Optional Control
Flexible Goal

Save Task

Cancel Task

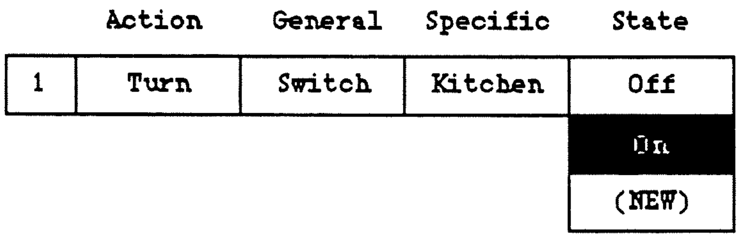
Enter Task Comment

Turn on the kitchen light.

Figure 8. Dialog Box Template

In this example, the user has entered a free form task description "Turn on the kitchen light" and has chosen "Required" as the task type. Finally information used to generate rules is entered. The rule information will assert a fact that the kitchen light has been turned on. The user may enter a number of facts that will be asserted from this dialog box. Other CLIPS functions can be used as well as user defined functions. The rule generated from this task will assert the following fact (kitchen switch turn on).

To insure that information asserted into CLIPS is consistent, a action template is provided. Entering information from the template is done from left to right. The first column represents the number of facts to be asserted. In Figure 9, the "Action" column will contain second person/present tense verbs that reflect the action within the domain. Once a verb has been selected, a "General" column is generated listing all objects associated with the selected action. Each object is then broken down further, into a list of "Specific"objects. The last column represents all valid "States" for each specific object. The user may, at anytime, insert new actions, objects or states into the network by selecting the (NEW) option.



	Action	General	Specific	State
1	Turn	Switch	Kitchen	Off
				On
				(NEW)

Figure 9. Semantic Format for Specified Action

RULE GENERATION METHODOLOGY

Knowledge Representation in CLIPS

The Rule generation component of TARGET takes the graphical description of a process and translates it into CLIPS rules. This section focuses on the graphics-to-CLIPS translation methodology using examples with CLIPS rules.

TARGET was originally designed to produce rules which could be incorporated directly into an ICAT (Intelligent Computer Aided Training) expert system architecture. In the ICAT architecture, the procedural rules interface with, and are controlled by, other CLIPS systems using a blackboard architecture. The following rule format will provide a simple control mechanism for the rules in this paper (Figure 10).

```
(defrule control_rule
                ?step <- ( next_step ?number )
=>
                (retract ?step )
                (assert (step ?number))
)
```

Figure 10. Simple Control Rule Format

The control rule acts like a traffic controller. It will receive a fact with the field next_step. The control rule will then retract this fact from the fact list and assert a new fact called step. The step fact will then trigger a TARGET rule which in turn will assert a next_step fact.

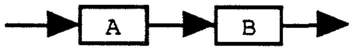
The rules, in Figure 11, produced conforms to a simple guideline. The rule will match on a control fact signifying the previous task has been executed. The rule will then retract that fact, call a series of functions needed to complete the current task, and finally assert a fact that this task has successfully completed its operation.

```
( defrule name
                ?step <- (previous task has been completed)
=>
                ( retract the previous task from the fact list )
                ( do zero or more functions (printout, assert, user defined, etc))
                ( assert a fact that this task has completed)
)
```

Figure 11. Rule Template in English

The following examples are intended to unite the TARGET graphical representation with skeletal CLIPS rules. In the following examples each task is labeled with the control facts using the same labeling approach.

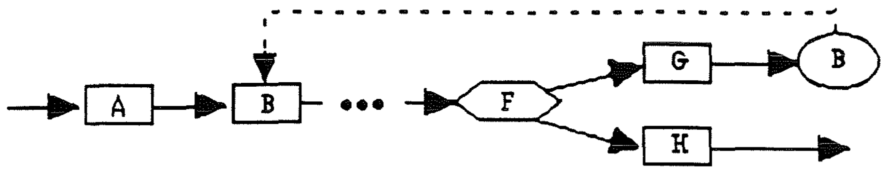
In the simplest case, all the tasks are linear. In the following example, task A must be executed before task B and so on (Figure 12). The rule generated for task B can only fire only after the previous step has completed. The control fact is then removed from the fact list. A series of functions are performed and a new control fact is asserted representing the completion of task B.



```
(defrule basic_case_B
                ?step <- (step A)
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step B))
)
```

Figure 12. Sequential Tasks

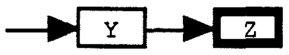
Control task rules in Figure 13 will fire only after the previous task has completed but will not process a function. They are only used to assert a control fact which will activate the task they are pointing to.



```
(defrule goto_rule
                ?step <- (step G)
=>
                (retract ?step)
                (assert next_step A)
)
```

Figure 13. Control Task Rule

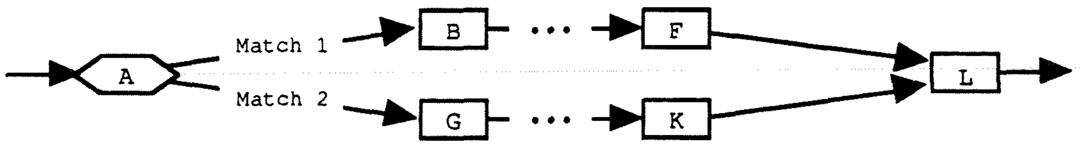
A rule derived from a goal task (Figure 14) will retract the previous control fact, process functions, but will not assert a control task, thus ending a process.



```
(defrule goal_rule_Z
                ?step <- (step Y)
=>
                (retract ?step)
                (function)
                (function)
)
```

Figure 14. Goal Task Rule

A more complex rule is generated when decisions and branching are involved. In the following example, Figure 15, decision A has two possible answers represented by Match 1 and Match 2. Depending on the outcome of the decision, one path B or G will be activated. When the two paths converge, task L must insure that either task F **or** task K has been successfully completed.



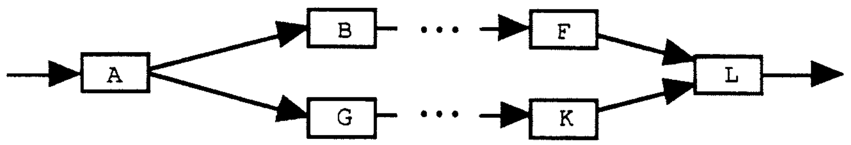
```
(defrule decision_rule_B
                ?step <- ( step A )
                (question_A match1)
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step B))
)
```

```
(defrule end_decision_rule_L
                ?step <- (step F|K)
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step L))
)
```

```
(defrule decision_rule_G
                ?step <- ( step A )
                (question_A match2 )
=>
                (retract ?step)
                (function 1)
                (function N)
                (assert (next_step G))
)
```

Figure 15. Parallel Alternative Paths

The CLIPS rules generated for parallel tasks are similar to those generated by decisions but differ in a few significant ways (Figure 16). First, matching on a decision fact is not required by the rules generated for B and G. In other words, once Task A has been completed, the two rules, B and G are both activated. Secondly, in rules B and G, the control fact from the previous step is not retracted from the fact list. If the control fact was retracted from the fact list, the first rule firing would prevent other tasks from being activated. The fact will remain on the fact list until it is removed when the parallel task chains merge. Finally, the rule associated with task L must not fire until tasks F **and** K have completed.



```
(defrule parallel_rule_B
                ( step A )
=>
                (function 1)
                (function N)
                (assert (next_step B))
)

(defrule parallel_rule_G
                ( step A )
=>
                (function 1)
                (function N)
                (assert (next_step G))
)
```

```
(defrule end_decision_rule_L
                ?step1 <- (step F)
                ?step2 <- (step K)
                ?start <- (step A)
=>
                (retract ?step1 ?step2 ?start)
                (function 1)
                (function N)
                (assert (next_step L))
)
```

Figure 16. Parallel Simultaneous Paths

DEVELOPMENT OF CISCO (CENTER INFORMATION SYSTEMS COMPUTER OPERATIONS) ICAT

TARGET is being deployed to acquire knowledge of the mainframe computer operations at JSC. The operating environment being modeled is the IBM VM (Virtual Machine) Operating System on an IBM 3090 super-mainframe CPU. TARGET tracked specific procedures for which operators were responsible. TARGET's ability to transform two-dimensional sequential events into a one-dimensional top-down report was appropriate for the type of tasks facing a mainframe systems operator. Procedures ranged from powering up and down of the CPU and its associated peripheral hardware to the initial program load and shutdown from its console monitor. After TARGET captured procedures from various sources (CIS-B Operator's Manual, SMEs, etc.), task hierarchy reports were generated and verified with the SMEs. This provided the foundation from which an CLIPS rules-based Intelligent Computer-Aided Training ICAT system was to be developed. The following is a description of the strategies taken within TARGET to generate the procedural rules for the CISCO (Center Information System Computer Operations) ICAT knowledge base.

As a procedural KA tool, TARGET models the intricate procedures which an IBM mainframe operator performs as a part of of his/her job. Such tasks as powering up/down mainframe and associated equipment, IPLs (Initial Program Loading), and system shutdowns are captured and converted to a knowledge base that will be used by the CISCO ICAT for mainframe operators at all levels.

Procedures for powering up an IBM mainframe system have been groomed for CLIPS interface. The following actions paired with the relevant translations is displayed in Table 1. Each task or discrete step consists of a specific action and its associated parameters. Through documentation of this action and its components, via the TARGET U.I., the CLIPS rule could be composed. A CLIPS rule requires several pieces of information extracted from a specific step. Figure 17 describes the transformation of a procedural step in the following manner:

Task# 1.1 Verify 208 VAC on voltmeter

Corresponding rule in ICAT format:

```
1       (defrule verify-208-VAC
2                   (step ?s&10)
3                   (environment ? AMD-plr-208-VAC present)
4       =>
5       (assert (message-E-to-I ?s require at-AMD-plr ver-plr VAC-208)
6                   (next-step 20)))
```

Description of the above example:
1 Unique name of rule.
2 Checks the step number.
3 Environment state is checked; *AMD-plr-208-VAC* is the action which contains all the relevant information. The desired state is *present.*
5 Assert *message-E-to-I* is the message from expert to the fact-list. The word *require* signals that this task is a required one. This is followed by an action and a set of parameters. Any parameter may be a combination of an action and parameters.
6 Go to step 20.

Figure 17. Single Task Step with Corresponding CLIPS Rule Content

The succession of task steps (Table 2) from the task hierarchy then becomes associated with CLIPS representations that will run within the CLIPS environment. Each action entry accumulated from the graphical interface will have its associated arguments/states (MOTOR-ON, GEN-ON, RESET-OVERVOLTAGE, GEN-OUTPUT, etc.). As a real application example, CISCO ICAT will use the CLIPS code generated from the TARGET environment to control a learning session concerning the operation and various scenarios within the mainframe computer environment.

	Task Hierarchy of <CIS-B POWER UP>	Domain Expert <Action> and <Arguments> (message-E-to-l <step> require <action> <argument> ...)	Environment State (on LHS) (environment O <variable> <state>)
1.0	3.1.1 AMDAHL Piller Power Up		
1.1	Verify 208 VAC on voltmeter	at-AMD-plr ver-plr 208-VAC	AMD-plr-208-VAC present
1.2	Press RESET OVERVOLTAGE SWITCH	at-AMD-plr prs-plr-sw RESET-OVERVOLTAGE	
1.3	Verify ON/OFF HANDLE (far left) ON (red)	at-AMD-plr ver-plr ON-OFF-HANDLE AMD-plr-ON-OFF-HANDLE ON	
1.4	Press (black) MOTOR ON BUTTON (grn START LIGHT ON now)	at-AMD-plr prs-plr-btn MOTOR-ON at-AMD-plr ver-plr START-LIGHT	AMD-plr-START-LIGHT ON
1.5	START LIGHT ON > 40 seconds? <NO>		
1.6	Press (black) GENERATOR ON BUTTON	at-AMD-plr prs-plr-btn GEN-ON	AMD-plr-START-LIGHT OFF
1.7	Verify (red) GENERATOR OUTPUT light ON	at-AMD-plr ver-plr GEN-OUTPUT	AMD-plr-GEN-OUTPUT ON
1.8	Verify (red) LOCAL SENSING light ON	at-AMD-plr ver-plr LOCAL-SENSING	AMD-plr-LOCAL-SENSING ON
1.9	Verify two GENERATOR VOLTAGE lights ON	at-AMD-plr ver-plr GEN-VOLTAGE-1 at-AMD-plr ver-plr GEN-VOLTAGE-2	AMD-plr-GEN-VOLTAGE-1 ON AMD-plr-GEN-VOLTAGE-2 ON
1.5	START LIGHT ON > 40 seconds? <YES>		
1.10	Press red MOTOR OFF BUTTON	at-AMD-plr prs-plr-btn MOTOR-OFF	AMD-plr-START-LIGHT ON
1.11	Call AMDAHL CE	com-AMD-CE repair-plr	
1.12	Goto 'Verify 208 vac on voltmeter'		

Table 1. Task Actions and their CLIPS Equivalents

Action/Argument	Action Name/Input	# of Args
at-AMD-pLr	Stand at AMDAHL pillar	1
ver-plr	Verify AMDAHL pillar component	1
prs-plr-sw	Press switch	1
prs-plr-btn	Press button	1
com-AMD-CE	Call AMDAHL CE	1
repair-plr	Repair AMDAHL pillar	0
208-VAC	208 VAC on voltmeter	0
RESET-OVERVOLTAGE	RESET OVERVOLTAGE SUITCH	0
ON-OFF-HANDLE	ON/OFF HANDLE at far left	0
MOTOR-ON	Black MOTOR ON BUTTON	0
START-LIGHT	Green START LIGHT	0
GEN-ON	Black GENERATOR ON BUTTON	0
GEN-OUTPUT	Red GENERATOR OUTPUT light	0
LOCAL-SENSING	Red LOCAL SENSING light	0
GEN-VOLTAGE-1	First white GENERATOR VOLTAGE light	0
GEN-VOLTAGE-2	Second white GENERATOR VOLTAGE light	0
MOTOR-OFF	Red MOTOR OFF BUTTON	0

Table 2. Examples of actions and arguments followed by their meaning and number of arguments.

The generated rule conforms to a general ICAT paradigm. However, translation to other knowledge base paradigms or architectures is feasible. The crucial issue is the extraction of actions and parameters from a task. This format is based on a general ICAT architecture developed by the programmers at Computer Sciences Corporation and the Software Technology Branch (PT4) at NASA/Johnson Space Center. The generated rule form is compatible with the ICAT architecture. Further work is being done for rule generation for other paradigms and architectures. The important issue is to extract actions and parameters from a task.

CONCLUSION

The CLIPS world offers TARGET a reasonable paradigm in which to produce knowledge representation from a knowledge acquisition effort. Whereas, TARGET offers the CLIPS world a mechanism in which to generate a knowledge base. As a CLIPS front-end, the system has been able to function as a knowledge engineering mediator for SMEs, programmers, managers and computer novices alike. Whether building a decision tree or a highly complex process network, TARGET provides latitude for a user to document their tasks or jobs with minimal prompting. In addition, the CLIPS code derived from the system would still provide a functional procedural model of the user's world. Overall, TARGET could significantly impact development of various ICAT systems as well as other intelligent systems. For any procedural knowledge acquisition task, it can enhance the ability of the expert to visualize and organize a task or process. Procedural visualization of this type will become more popular as more tools with organizational diagnosis capabilities evolve (Akscyn, McCracken, and Yoder, 1988).

As computer hardware power evolves, more latitude in presentation methods will be available. Visual conception and communication of abstract information will become more common. The strategic fusion of graphical display (bit-map, meta-graphic, etc.) and graphical input device (mouse, light-pen, trackball, etc.) technologies will facilitate visual as well as textual representation of knowledge (Messinger, Rowe, and Henry, 1991). Drawing tools already allow the user to produce and manipulate complex graphics. The role of these tools can also combine with organizational algorithms to create more intelligent diagrams, flow charts and interactive decision trees. With users becoming more adept at employing systems with pictorial modeling capabilities, the mode of procedural KA will also benefit from such advances.

As knowledge acquisition evolves as a discipline within artificial intelligence, more tools to assist in the knowledge acquisition process will also become available in useful forms. TARGET, and tools like it, will be employed within their own "niche" and will also be integrated with other methodologies in the future. Although TARGET currently models the sequence within the task hierarchy structure for rule induction, we will dedicate additional efforts to encapsulating additional knowledge into the steps within a network. In particular, we intend to address issues such as gathering artifact data, selected action rationale, and interactive verification and validation of rules.

REFERENCES

Akscyn, R. M., McCracken, D. L. & Yoder, E. A. (1988). "KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations", *Communications of the ACM*, July, 31 (7), 820-834.

Boose, J. H. (1989). A survey of knowledge acquisition techniques and tools. *Knowledge Acquisition*, March, 1 (1), 3-37.

de Kleer, J., Doyle, J., Steele, G. L., Jr. & Sussman, G. J. (1985). AMORD: Explicit Control of Reasoning. in *Readings in Knowledge Representation*, Brachman, R. J. & Levesque, H. J., Eds., Los Altos, CA: Morgan-Kaufmann Publishers, Inc., 345-355.

Gaines, B. R. (1988). An overview of knowledge-acquisition and transfer. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R & Boose, J. H., Eds., *Knowledge-Based Systems, Vol.1*, New York: Academic Press, 3-22.

Littman, D. C. (1988). Modelling human expertise in knowledge engineering: some preliminary observations. in *Knowledge Acquisition for Knowledge-Based Systems*, Gaines, B. R. & Boose, J. H., Eds., *Knowledge-Based Systems*, Vol.1, New York: Academic Press, 93-104.

Loftin, R. B., Wang, L., Baffes, L. & Hua, G. (1988). An Intelligent Training System for Space Shuttle Flight Controllers. *Proceedings of the 1988 Goddard Conference on Space Applications of Artificial Intelligence*, held May 24, 1988, at NASA/Goddard Space Flight Center, Greenbelt, Md, 3-10.

Loftin, R. B., Wang, L., Baffes, P. & Hua, L. (1989). An Intelligent System for Training Space Shuttle Flight Controllers in Satellite Deployment Procedures. *Machine-Mediated Learning*, 3, 43-47.

Messinger, E. B., Rowe, L. A. & Henry, R. R. (1991). A divide-and-conquer algorithm for the layout of large directed graphs. *IEEE Transactions on Systems, Man, and Cybernetics*, 21 (1), 1-11.

Payne, S., & Green, T. (1986). Task-action grammars: a model of the mental representation of task languages. *Human-Computer Interaction*, 2, 93-133.

N92-16582

PROJECTS IN AN EXPERT SYSTEM CLASS

George M. Whitson

Computer Science Department
The University of Texas at Tyler
Tyler, Texas 75701

Abstract. Many universities now teach courses in expert systems. In these courses students study the architecture of an expert system, knowledge acquisition techniques, methods of implementing expert systems and verification and validation techniques. A major component of any such course is a class project consisting of the design and implementation of an expert system. This paper discusses a number of techniques that we have used at The University of Texas at Tyler to develop meaningful projects that could be completed in a semester course.

HISTORY

As expert systems theory became a major subset of Artificial Intelligence, many universities introduced courses on his subject. Initially, there were few texts in the field and little software that was available for classroom use. Today there are many texts and inexpensive software systems to support classroom instruction. A problem in the teaching of expert systems, since such courses have been offered, is designing classroom projects that are both meaningful and that can be done in a one semester period. This paper discusses a number of projects that I have assigned over the past few years. The type of project assigned depends on the emphasis of the expert systems course, as we will illustrate in the next two sections. For those courses based on a high level shell, like CLIPS, we feel we have developed a very good method for having class projects that are both meaningful and do-able.

TYPES OF EXPERT SYSTEMS COURSES

Although the expert system field is quite young there have been a number of different types of expert systems courses. Until I started using CLIPS two years ago we tried a lot of different approaches to teaching expert systems. At our university I have taught the course with each of the following emphasis:

1. In our first expert systems course, taught in 1984, we emphasized artificial intelligence and discussed all aspects of the theory of expert systems in terms of artificial intelligence. For example, we did not look at fact

representation until we had done a complete coverage of knowledge representation.

2. In our second approach to teaching expert systems we gave a quick introduction to artificial intelligence, a solid coverage of all of the different implementation languages that were in use, including LISP, PROLOG and GOLDWORKS.

3. In our third major revision of the expert systems course we gave a brief introduction to artificial intelligence, an introduction to logic programming and spent the remainder of the course looking at techniques for implementing expert systems in PROLOG.

4. In our fourth approach to teaching expert systems we expanded the definition of an expert system to include many systems that were not rule-based. In particular, we spent a good deal of time looking at artificial neural systems and how they were related to rule-based expert systems.

5. In our final and current approach to teaching an expert systems course we quickly introduce the ideas of a rule-based expert system, get the students started on a major project and then lecture on other important topics, like fuzzy logic and artificial neural systems.

EXPERT SYSTEM PROJECTS

In each of the different types of expert systems courses we used different types of projects. It is interesting to consider how these projects have changed as the course content has changed. A sketch of the projects we gave in each of the above types of courses is:

1. When we stressed artificial intelligence we alway gavs a project to develop an expert system shell in LISP. The sketch given in [Winston, 1989] the approach we usually used. The projects were interesting from the systems point of view, but, as one can well imagine, the students spent all their time developing the shell and little time using it.

2. When we stressed a comparison of the different languages and shells used to develop expert systems, we found that we would always take a simple project so that we could implement it in several shells. Each student did the same problem in two different ways. All projects were presented to the class and then a test covering all approaches was given. As one could easily guess, we spent most of our time learning the syntax of the different tools.

3. When we based our expert systems course on PROLOG we were able to develop a fairly complex project, but fond that much explanation of logic programming was needed. Some of the specific techniques of PROLOG, especially unification, required a lot of explaining as well. While we still use PROLOG occasionally as a tool, we still find that we cannot spend as much time as we want on the real topics of expert systems, like knowledge acquisition, because of the need to explain programming language details.

4. When we emphasized artificial neural systems in our

expert systems course, the project consisted of developing an artificial neural system and an expert system to solve the same problem. While it was interesting to compare the two techniques, we realized at the end of the project that the rule-based system was not as complex as desired. This resulted from our desire to be sure we had selected a problem that had a solution in both technologies.
5. When we based our expert systems course on a shell for the project we found that with a little work we were able to develop a fairly complicated project in a single semester. While all of our previous methods of teaching the course were successful, we feel that the current approach is ideal for today in that the students spend all of their time developing and testing a real expert system rather than doing other things.

PROJECTS IN A CLIPS BASED COURSE

There are many techniques used to develop a project in an expert systems course. The one we most often hear presented at meetings is to simply require each student to develop a project on their own. The instructor provides guidance and controls the quality of the project, but the student selects the topic and finds the domain expert. While this is fine for students who are domain experts, we don't think it works very well for those who are not. We have developed a method for selecting projects that we feel is very good. This method has resulted in our developing several nice prototypes as a part of the class and has been enthusiastically received by our students. We now list some of the key components of our method of doing projects in an expert system course.

1. We define a high level tool that everyone will use to do their project. For the past two years we have used CLIPS and plan to continue to use CLIPS in the future. CLIPS is easy to use and has all the features we really want in our expert system shell. In addition, students can do much of their development work at home. We have also used GOLDWORKS and EXSYS, but prefer CLIPS.
2. I select a single project for the entire class and line up some domain experts who have an interest in developing an expert system. So far we have found domain experts who would participate in our project in several ways. In one case we found that we had four personal computer repair people in a class, in another we found several Biologists we wanted to see a diagnostic expert system for plant disease developed, and in another we found a local manager who needed to have his knowledge built into a system.
3. The class is divided into teams of about five students each. This division is dictated by our access to experts. Each team selects people to acquire the knowledge, code the system, test the system and develop the system documentation. All students are responsible for all phases of the development, but selected students are leaders in certain parts (like knowledge acquisition.)

4. The class project is described early and students begin developing the system shortly after the first week of class. This means that the expert systems theory about topics like knowledge acquisition and system verification are often covered in the lecture after students have practiced some of the ideas in their own system.
5. A major emphasis of our course is on knowledge acquisition via the traditional techniques of observation, interviewing and becoming a pseudo-expert. We spend several weeks each semester on knowledge acquisition and cover several automated techniques as well as the traditional methods.
6. Each team develops a complete prototype system. The systems are usually similar since I work with each team and present an overall suggested system design at the start of the course. But each project always has its own unique flavor. Members of the team do both verification and validation of the system as well as developing a user's manual. These prototypes have often been developed into full systems after the course is finished. At the end of the course each team demonstrates their system to the class and all systems are discussed and evaluated by the class as a whole.

SUMMARY

As one teaches an expert systems course with different emphases, projects take on different forms. Some projects develop expert system tools, some survey the current tools and some concentrate on the art of building real expert systems. After experimenting with many different types of course and projects, I have come to the conclusion that it is best to teach an expert systems course that concentrates on developing a real system. For such a course I have found that a single class project implemented by teams is best.

REFERENCES

Giarantano and Riley (1989). *Expert Systems: Principles and Programming,* PWS-KENT, Boston.
Hayes-Roth, Waterman and Lenat (1983). *Building Expert Systems,* Addison-Wesley, Reading, Mass.
Malpas, John (1987). *PROLOG: A Relational Language and its Applications,* Prentice-Hall, Englewood Cliffs, New Jersey.
Waterman, Donald (1985). A Guide to Expert Systems, Addison-Wesley, Reading, Mass.
Winston and Horn (1989). *LISP, 3rd Edition,* Addison-Wesley, Reading, Mass.

N92-16583

USING CLIPS AS THE CORNERSTONE OF A GRADUATE EXPERT SYSTEMS COURSE

Kwok-bun Yue

University of Houston - Clear Lake
2700 Bay Area Boulevard, Houston, TX 77058

Abstract. This short article describes the effective use of CLIPS as the cornerstone in a graduate expert systems course. The course included about 8 to 9 hours of in-depth lecturing in CLIPS, as well as a broad coverage of major topics and techniques in expert systems. As part of the requirement of the course, students solved two small yet non-trivial problems in CLIPS before went on to develop a toy expert system in CLIPS in an incremental manner as the term project. Furthermore, students were required to evaluate CLIPS programs by their classmates. An anonymous questionnaire at the end of the semester revealed that the students responded very favorably about the course, especially their experience with CLIPS.

INTRODUCTION

This article describes the experience of teaching and using CLIPS as the expert system shell language in a graduate expert systems course in the Spring semester of 1990. The department of computer science at the University of Houston - Clear Lake offered two graduate courses in the artificial intelligence area. The first course, Artificial Intelligence, is a survey course of general searching techniques and knowledge representation issues, as well as various application areas such as learning, natural language processing, vision, etc. The second course, Expert Systems, is intended to include an in-depth coverage of expert systems theory and programming techniques. Currently, the department also offers a graduate course in Artificial Neural Systems.

There were several papers in the literature describing the teaching of expert systems courses (Bahill and Ferrell 1986, Brown 1987, Warman and Modesitt 1989, Wolf and Rozanski 1990). (Bahill and Ferrell 1986) discussed the advantages of using an expert system shell and emphasized the importance of a student term project. However, the course did not cover many expert system techniques such as those in the areas of uncertainty handling and knowledge acquisition. (Brown 1987) emphasized the evaluations of expert systems and were more suitable for students majoring in business rather than computer science. Both (Warman and Modesitt 1989) and (Wolf and Rozanski 1990) centered their courses on group projects and were quite practically oriented.

When we developed our course, we felt that it is more suitable to have a relatively complete coverage of major topics in expert systems, especially since the technology has matured significantly in recent years. On the other hand, we would also like to teach an expert system shell language quickly and in-depth so that the students can develop their expert system term projects as soon as possible and with confidence.

CLIPS was used because it is relatively simple and can be mastered in relatively short time. CLIPS is also free and a copy of it (version 4.0) is enclosed in our textbook, (Giarratano and Riley 1989). Furthermore, most of our students are working in NASA related companies that use CLIPS extensively.

The first week of the course was devoted to the introduction and general theory of expert systems. To enable the student to develop skills in CLIPS as early as possible, we covered CLIPS in the second to fourth weeks of the course for a total of about 8 to 9 hours. This encompassed Chapters 7 to 12 in (Giarratano and Riley, 1989). Various assignments, as described in the next section, were assigned to the students to enhance their CLIPS programming skills.

Since CLIPS is a forward chaining expert system language, we spent time to cover languages in other paradigms to complement CLIPS. For example, an hour was used to lecture on VP-Expert and M1 to illustrate backward chaining shell languages. Framed-based systems and object-oriented based systems were also discussed. The exception is logic-based systems, which were assumed to be well covered in the prerequisite graduate Artificial Intelligence course.

From the fifth week on, various important topics in expert systems theory were elaborated. This included languages and tools, evaluation techniques, verification and validation, and explanation facilities. We spent especially generous amount of time in knowledge acquisition, where more than 75% of the materials in (McGraw and Harbison-Briggs 1989) were covered, and uncertainty handling, where techniques as advanced as Dempster-Shafer theory and fuzzy logic were elaborated. The last class was used for student presentations of selected term projects.

HOMEWORK ASSIGNMENTS

Believing that active participation is the best way of learning, a heavy dose of assignments was included in our courses. A brief description of the assignments is shown in Table 1 in the next page. Our assignments have the following characteristics.

(a) Nearly all assignments are related to CLIPS. It is hoped that this concentration on a single language will allow the students to develop true expertise in the language.
(b) Two somewhat traditional yet non-trivial problems were included to enhance student programming skills in CLIPS.
(c) Evaluation of other classmate programs were emphasized. This forced the students to read programs and learn by comparison and contrasting.
(d) The final term project was built in an incremental manner which better imitates how expert systems are constructed.

Each assignment is described in more details now. The purpose of the first assignment is to let the students to compare the flavor of a rule-based programming language (CLIPS) to a more traditional language like Lisp.

Homework #1: Implement the CLIPS program for the block world in the textbook (pp. 417-418) in LISP or PROLOG. Test your program with the test cases in the textbook. Briefly discuss the relative merits and difficulties of your LISP implementation.

Homework #2: Extend the CLIPS program for the block world problem in Homework #1 to handle multiple move-goals. Test your new CLIPS program with sufficient number of cases and hand in both the program and the output.

Homework #3: Design and write a CLIPS program that serves as an interpreter of nondeterministic finite state machines. Again, test it with sufficient number of cases. Describe briefly the difficulties that will encounter if you were using LISP or PASCAL.

Homework #4: Interview an expert in any problem of your interest. Hand in:

(1) a brief description of the problem,

(2) a brief listing of the reasons for selecting the problem, and

(3) about 10 non-trivial rules that the expert may use and that may form the core of a small expert system (be sure to start in a high enough level of abstraction.)

Homework #5: Implement the rules you obtained in Homework #4 in CLIPS. Be sure to include enough information so that other people can use your toy expert system. Hand in three copies.

Homework #6: Evaluate two toy expert systems by your classmates.

Homework #7:

(1) Describe and evaluate an expert system shell.

(2) Hand in a research paper in expert systems, together with your review.

Project: Develop and refine your toy expert system into a more powerful one in CLIPS with at least 40 non-trivial rules. Write a report about your expert system. You may need to give a presentation of its implementation. Hand in two copies.

Final Examination (Take home):

(1) Evaluate one of your classmates' expert systems.

(2) Answer several questions.

Table 1. Homework assignments and project.

The second and third assignments aimed to strengthen students CLIPS programming skills. Whereas there are ample of exercises in (Giarratano and Riley 1989), they are relatively easy and are much simpler than the complexity of an actual expert systems. Of course, students can develop CLIPS programming skills while developing their term projects. However, there are many tasks, such as problem definition, knowledge acquisition and representation, in developing an expert system to distract the students from concentrating on practicing CLIPS. It is thus desirable to find some problems that are clearly defined, have relatively short solutions and are non-trivial.

The second and third assignments fitted these requirements well. In the second assignments, students were asked to modify the elegant solution (only 4 rules) in (Giarratano and Riley 1989) for planning in the block world of Winograd to handle the satisfactions of multiple goals (such as to move block A on top of block B and B on top of C). We were amazed that more than 5 different approaches were collected and though the solution has only about 10 rules, no student produced a correct solution which is also optimal (minimal number of movements of blocks). Same thing can be said of the third assignment, which implements an interpreter of nondeterministic finite state machines. Although the solution has only six rules, the rules are quite intricate and no student got everything right (most students overlook a potential infinite loop problem).

The remaining assignments were essentially the incremental development of a toy expert system and the evaluations of other classmate works. One major difference between our term project and the projects described in other literatures is that the students were requested to turn in prototypes (Homework #5) of their toy expert systems. This is used to emphasize the incremental nature of expert systems development. Another difference is the requirement to evaluate other classmates' expert systems. This reinforced the importance of the ability to evaluate expert systems as well as forcing students to read CLIPS programs. In general, we think that reading programs is an invaluable experience for gaining expertise in computer languages and it is not emphasized enough in many computer science curricula.

As in other expert systems courses, a wide variety of student expert systems were collected. The students reported no difficulty in implementing their expert systems in CLIPS and their CLIPS code are in general competent. On the other hand, some students had problems in identifying suitable expert system applications and finding experts. Two projects turned out not to be suitable for expert systems technology and it was obvious that at least two projects were not based on a real expert. The identification of suitable applications and experts will be of utmost importance in the future.

DISCUSSION AND CONCLUSION

Student evaluations at the end of the semester revealed that the course was very well received by the students, indicating that it is feasible to use CLIPS effectively as the cornerstone of an expert systems course.

An independent in-depth anonymous questionnaire were also filled by the students at the end of the semester. The student responses on the usefulness of lectures on various topics may range from 7 (most useful) to 1 (least useful). The average student response on the usefulness on the overall lectures is 6.1 whereas the average response to the lectures on more theoretical topics (such as verification and validation and uncertainty handling) ranged from 5.3 to 5.9. As a contrast, the average response of the usefulness of lectures on CLIPS is 6.7. This is significantly better than that of other topics.

There may be many reasons for the relatively low enthusiasm for more theoretical topics. For example, the theory on uncertainty handling requires solid mathematics background. Topics like verification and validation of expert systems are still immature and are thus less organized and structured. Time limitation dictated that theory of knowledge acquisition can only be discussed in lectures but not practiced in the classroom, making it much less interesting and useful.

On the other hand, the responses positively indicated that the students were very interested in learning an expert system shell, CLIPS in our case, thoroughly so that they can practice the theory and develop their expert systems immediately. We feel that CLIPS is especially a good choice since it is concise, efficient, practical, free and can be mastered in relatively short time.

The student fondness of CLIPS was also shown in their responses on the usefulness of the assignments. Of all assignments, the response for the term project was the best with an average of 6.8. The two CLIPS assignments (homework assignments #2 and #3) draw an average response of 6.4 each. In contrast, the average responses of other assignments ranged from 4.9 (evaluation of an expert system shell) to 6 (development of the prototypes of the toy expert systems).

In conclusion, we believe that CLIPS can be used effectively as the cornerstone of a relatively complete graduate expert systems course, especially now that CLIPS version 5.0 provides object-oriented features to complement its rule-based programming paradigm. Well designed CLIPS problems are very helpful to the students for developing CLIPS programming expertise. A collection of such small yet illustrative problems should be very beneficial in teaching CLIPS in the future.

REFERENCES

Bahill, A. and Ferrell W. (1986). Teaching an Introductory Course in Expert Systems, ***IEEE Expert***, Vol. 1, No. 4, pp.59-63.

Brown, D. (1987). A Graduate-Level Expert Systems Course, ***AI Magazine***, Vol. 6, No. 3, pp.33-39.

Giarratano, J. and Riley, G. (1989). ***Expert System - Principles and Programming***, PWS-Kent, Boston, Massachusetts.

McGraw, K. and Harbison-Briggs, K. (1989). ***Knowledge Acquisition - Principle and Guidelines***, Prentice-Hall, Englewood Cliffs, New Jersey.

Wolf, W. and Rozanski E. (1990). Expert Systems: An Applied Course, ***SIGCSE Bulletin***, Vol. 23, No. 4, pp.23-24.

Warman, D. and Modesitt, K. (1989). Learning in an Introductory Expert Systems Course, ***IEEE Expert***, Vol. 4, No.1, pp.45-49.

SESSION 4 A

PRECEDING PAGE BLANK NOT FILMED

N92-16584

CRN5EXP - EXPERT SYSTEM FOR STATISTICAL QUALITY CONTROL

Mariana Hentea, Senior Consultant
Interactive Business Systems, Oak Brook, Illinois

ABSTRACT. The purpose of the Expert System CRN5EXP is to help the user to check the quality of the coils at two very important mills: Hot Rolling and Cold Rolling in a steel plant. The system interprets the statistical quality control charts, diagnoses and predicts the quality of the steel. Measurements of process control variables are recorded in database (ADABAS) and sample statistics such as the mean and the range are computed and plotted on a control chart. The chart is analyzed through patterns using CLIPS and forward chaining technique to reach a conclusion about the causes of defects and to take management measures for the improvement of the quality control techniques.

The Expert System combines the certainty factors associated with the process control variables to predict the quality of the steel. The paper presents the approach to extract data from database, the reason to combine certainty factors, the architecture and the use of the Expert System. However, the interpretation of control charts patterns requires the human expert's knowledge and lends to Expert Systems rules.

The conclusions reached with this system help the management and the quality engineers to eliminate the special causes of the process control variable variations and to correct about 85% of the problems from these mills.

EXPERT SYSTEM OVERVIEW

The purpose of the Expert System CRN5EXP is to help the user to track the quality of the coils at two very important mills: hot rolling (HSM) and cold rolling (CRN5). The user needs this system to find out why the coils at CRN5 had gauge variation and what is the predicted quality of the coils produced at HSM.

Hot Rolling is an upstream process and Cold Rolling is a downstream process. The quality of the coils is measured primarily by gauge, every coil produced at CRN5 should have the gauge ordered by customer. If a coil doesn't have the dimensions required by the customer then it is rejected. At CRN5, the gauge variation is a function of the mill set-up, such as: work rolls, diameters, tons of coils rolled between roll changes, frequency of roll changes, maintenance, hardness of work roll surface, roll forces, tension, speed, motor power. At HSM, the gauge variation is a function of the finishing and coiling temperatures.

The implemented software is a complex of NATURAL programs which search database files (ADABAS) for the coils which have gauge variations. The program contains routines which calculate the control charts using statistical method FORD. Control charts

PRECEDING PAGE BLANK NOT FILMED

give a good indication of whether any problems are likely to be correctable locally or will require a management action. The database files (ADABAS), on mainframe IBM, store process control variables specific to each process: hot rolling and cold rolling.

The coils which are not in statistical control for the gauge at cold rolling and hot rolling are downloaded from database files to PC files (ASCII) which are the input data for the Expert System. The present Expert System contains rules to check the variables specific to each mill. The user may select the coils produced during a period of time by entering the starting and ending dates.

SYSTEM FUNCTIONS:

CHECK THE COILS AT CRN5

CHECK THE COILS AT HSM

The user from cold mill is interested to find the causes of the gauge variations for the coils. The knowledge about mill set-up and actual process control variables help the management to eliminate special causes of gauge variations and to correct about 85% of process control problems.

The Expert System reads the data from the input file, checks the values of the varibles and provides a detailed report of the running mill. At CRN5, the process control variables, such as: tons of coils rolled between roll changes, work roll numbers, diameters, surface, frequency of roll changes, roll forces, tension, speed, and motor power are compared with the computer predicted values. The Expert System analyzes the data and issues conclusions regarding these variables. The present System is able to reach more than twenty conclusions. At HSM, the software checks the coiling and finishing temperature values.

The Expert System implements two functions: checks the coils produced at CRN5 and the coils at HSM. If the user selects the option to check the coils produced at CRN5, the Expert System analyzes the process control variables recorded at CRN5. If the actual variables are within the standards allowed for this mill, then the Expert System checks the variables recorded at HSM. If the actual variables, finishing and coiling temperatures, are within the standards allowed for the HSM, then the Expert System advices the user to search process control variables from another mill (pickle), which may had caused the gauge variations (not implemented yet). If the actual variables from CRN5 are not within the standards, then the Expert System issues different conclusions and advices managerial actions. If the actual variables from CRN5 are within the standards, then the Expert System checks the actual variables recorded at HSM. If the actual temperatures are not within the standards, then the Expert System indicates the quality of the produced coil at HSM. At user request, the Expert System provides an explanation facility which gives a detailed report about the values of actual temperatures and the way they influenced the quality of the produced coil. The finishing and coiling temperatures influence the quality of the coil which is stated as normal, soft or hard. If the user selects the option to check the coils produced at HSM, then the Expert System checks the variables recorded at HSM and predicts the quality of the coil. The Expert System helps the user from CRN5 to determine the quality of the incomming coils and based on that to compute the values for the model which are the standards for CRN5. The Expert System is used in both mills to improve the quality of the produced coils.

EXPERT SYSTEM KNOWLEDGE

The present system implements the knowledge in rules based on the information acquired from the experts (metallurgist, chemist, quality and process control engineers). The metallurgist and quality engineers consider that the values of roll forces, tension, speed, and motor power are determined by the the coil hardness. If the roll forces are high, above the predicted values, then it is an indication that incomming coil is hard. If the roll forces are low, below the predicted values, then it is an indication that the incomming coil is soft.

The program checks first all the variables specific to CRN5. If all the variables are in statistical control, then the program checks the variables from the upstream process. The program implements rules to set-up the certainty factors for different temperature ranges, based on the specialist's experience. For example, if the finishing temperature is below or above AIMS temperature (standard value based on the grade of the steel) more than 30 degrees Fahrenheit, then the coil is not certainly normal: it is hard or soft with a certainty factor. The system displays to the user the term "probability". If the finishing temperature is above the AIMS, then the coil is considered soft. If the finishing temperature is below of the AIMS temperature, then the coil is considered hard.

If the coiling temperature is not within the upper and lower limits (standard values based on the grade of the steel) then the coil is not certainly normal. If the coiling temperature is above the upper limit than the coil is soft. If the coiling temperature is below the lower limit then the coil is considered hard. Even though the temperatures are within limits, the certainty factor that a coil is normal is reduced by the certainty factors of each temperature value. The certainty factors are combined using MYCIN formulas or expert's experience in some cases. It depends of the type of effect each temperature has on the quality of the hot rolled coil. The coiling is a variable measured before finishing temperatures, so its variation causes variations to the finishing temperatures.

Checking the coils at HSM may conclude that the temperatures were normal. The analysis of the coils with gauge variation prove that there are all kinds of causes for that: process control variables outside of the ranges allowed in the mill, maintenance problems, wrong computer model.

USE OF THE SYSTEM

The Expert System implements two functions displayed on the main menu. The user can select one option and the program loads the facts in memory. The input data is read from the ASCII files (the variables specific to CRN5 and HSM are stored in separate files). The user may check a coil produced at CRN5 or at HSM. The Expert System implements also an explanation facility, the user can check why the hardness of the coil is normal, hard, or soft. The user is informed about the values of certainty factors for the coiling and finishing temperature which influence the certainty factor for the hardness of the rolled coil at HSM. By knowing the hardness of the coil expressed in certainty factor, the process control engineers can take managerial actions to improve the mill set-up.

EXPERT SYSTEM ARCHITECTURE

The system contains eighty rules. The facts for option one are loaded from the file containing the process control variables specific to CRN5. The facts for option two are loaded from the file containing the process control variables specific to HSM.

The system reaches more than twenty conclusions regarding the quality of the coil and the mill set-up.

The explanation facility uses a text file to store all the knowledge which supports a conclusion for every option. The certainty factors were used to combine facts about coiling and finishing temperatures to predict the quality of the coil (normal, hard, or soft).

EXPERT SYSTEM FOR STATISTICAL QUALITY CONTROL

The present Expert System interprets the statistical quality control charts to monitor, diagnose, and predict the possible quality of the coil.

In many companies, it is a frustrating struggle to train and educate the personnel on the proper use of control charts. Therefore, it is reasonable to provide a uniform interpretation of the control charts and to establish a state of control during manufacturing process. The Expert System significantly improves the quality control techniques.

REFERENCES

Hayes-Roth, F., D. A. Waterman and D. B. Lenat (1983). Building Expert Systems. Addison-Wesley, Redding.

Giarratano, J. C. and G. Riley (1989). Expert Systems : principles and programming. PWS-Kent, Boston.

N92-16585

DISTRIBUTED SEMANTIC NETWORKS AND CLIPS

James Snyder and Tony Rodriguez

CAD Research Unit, Design Institute
California Polytechnic State University, San Luis Obispo

Abstract. Semantic networks of frames are commonly used as a method of organizing and reasoning in many types of problems. In most of these applications the semantic network exists as a single entity in a single process environment. Advances in workstation hardware provide support for more sophisticated applications involving multiple processes, interacting in a distributed environment. In these applications the semantic network may well be distributed over several concurrently executing tasks.

This paper describes the design and implementation of a frame-based, distributed semantic network in which frames are accessed both through CLIPS expert systems and procedural C++ language programs. The application area is a knowledge-based, cooperative decision making model utilizing both rule-based and procedural experts.

INTRODUCTION

Currently, the CAD Research Unit is developing an Intelligent Computer Aided Design System (ICADS). The purpose of ICADS is to provide an intelligent environment for cooperative computer-based problem solving under the explicit control of the user using architectural design as a test environment. ICADS allows a group of distributed, intelligent agents to converse about a problem larger than any single agent's domain of expertise or knowledge, and provides advice and suggestions to the user as to the state and compliance of the current problem solution.

From a high-level point of view, ICADS is composed of three major pieces: a blackboard, a group of procedural and CLIPS-based expert systems referred to as Intelligent Design Tools (IDTs), and a semantic network of frames. The ICADS components run as distributed processes in a computer network.

To facilitate distributed expert system execution, a communication framework was developed which allows CLIPS expert systems to assert facts to another expert system under the control of a blackboard (Taylor 1990, Taylor and Myers 1990). In addition to controlling the IDT communication, the blackboard has a conflict resolver which arbitrates between IDTs. The conflict resolver establishes the system accepted values by evaluating suggestions made by IDTs and is written in CLIPS. The current working model of ICADS has six CLIPS based expert-systems. Each expert is controlled by the blackboard using the communication framework described above and are in the following architectural domains: cost, access, lighting, thermal, acoustics, and structure (Pohl 1989).

To allow C++ programs to participate in the current problem solution, they have to communicate with the blackboard and have access to the semantic network of frames at the same level of representation as the CLIPS experts. The remainder of this paper will describe the implementation of the distributed semantic network in the CLIPS and C++ environments and illustrate sample uses of the C++ implementation used in conjunction with CLIPS-based experts.

THE CLIPS FRAME REPRESENTATION

Within the CLIPS environment, the frame-based representation used in ICADS is implemented as a set of CLIPS facts. A frame is a collection of information about a class or object. The information is represented in CLIPS with a frame header fact and any number of slot facts. Slots can define a particular value of the class or identify a relation to another class. In terms of a node-link data structure, the frame is a node and a relation is a link (Barr and Feigenbaum 1981). It is important to note that this representation does not require any modifications to CLIPS--it uses only CLIPS facts (Assal and Myers 1990). Unlike procedural paradigms, it is not necessary to locate an entire frame when a piece of information is needed. Only the pertinent slots are necessary and are accessed directly through CLIPS pattern matching. A frame is represented by a set of facts that have one or more common fields to connect them together. Each fact has a keyword in the first field to indicate the type of information it represents. The keywords are: FRAME, RELATION, VALUE. The second field has the class name which is used to connect all the instances of this class or establish a relation with another class.

CLIPS Frame Definition

Figure 1 illustrates the CLIPS fact format of a frame header. The *class* field defines the class to which the frame belongs. The *instance* field uniquely identifies the frame. Having the frame header in a CLIPS left-hand-side pattern is not always necessary, but it is useful in performing operations on the whole frame; displaying and deleting are example operations.

```
(FRAME <class> <instance>)
```

Figure 1 - Frame Fact Format

CLIPS Value Slot Definition

Figure 2 shows the CLIPS fact format of a value slot. This slot provides the values for particular attributes of a frame. It is important to note that these values are multifield values and can contain mixed types of data. Note also that the instance of the frame in contained within the value slot fact. This allows for direct pattern matching of the frame attribute values.

```
(VALUE <class> <attribute> <instance> <values>)
```

Figure 2 - Value Slot Fact Format

CLIPS Relation Slot Definition

Figure 3 illustrates the CLIPS fact format of a relation slot. This relation slot represents a **has-a** relationship. The *class1* and *instance1* fields indicate the owning class. The *class2* and *instance2* fields represent the frame instance which is pointed to.

```
(RELATION <class1> <class2> <instance1> <instance2>)
```

Figure 3 - Relation Slot Fact Format

An Example CLIPS Frame

An example architectural object is a room or space. Figure 4 contains an example space frame with several values and relations. The relations in the example indicate that the LOBBY space has four walls. If a wall of a particular space is to be referenced by a rule, a pattern similar to the example would be used.

```
(FRAME space 15)
(VALUE space name 15 LOBBY)
(VALUE space perimeter 15 108)
(RELATION space wall 15 1)
(RELATION space wall 15 2)
(RELATION space wall 15 3)
(RELATION space wall 15 4)
```

Figure 4 - An Example Architectural Frame

An Example Rule Using a CLIPS Frame

Suppose the building code states that the area of all bathrooms must be greater than or equal to twenty square feet. The rule in Figure 5 checks to see if the area of a bathroom is less than twenty square feet. If the area of the space is too small an error message is printed. This simple example illustrates how values slots can be used. Relation slots are used in a simpler manner; the type of relation is included in the left-hand-side pattern and can be used to reason about groups of objects at the same time. For example, a rule could specify a pattern which references all the walls which belong to the BATHROOM space.

```
(defrule building-code-check
  (FRAME space ?id)
  (VALUE space name ?id BATHROOM)
  (VALUE space area ?id ?x&:(< ?x 20))
=>
  (printout t "The bathroom is too small" crlf)
)
```

Figure 5 - An Example Rule Using a Frame

C++ IMPLEMENTATION OF FRAMES

The purpose of the frame classes is to provide an object-oriented representation for the frame facts that are used by the CLIPS IDTs. In order to allow C++ programs to work with the current semantic network that is represented in CLIPS, a representation paralleling the CLIPS frame data structures was implemented using C++ objects. Using the class method interface, frames can be built explicitly and then added to the network; the user is responsible for creating any necessary objects to insert into the frame--relations and values are examples. After the frame has been built correctly it is then entered into the semantic network using net class methods. The following sections explain the structure of the C++ classes used to represent a semantic network of frames.

The Net Class

The net class is basically a container class that represents the entire semantic network. It is composed of frame objects. The net class provides several methods for the addition, removal, and modification of frames, as well as, query methods that allow questions to be asked about the network. Figure 6 shows the class structure of the net class. The *frames* data member is a dictionary of frame objects. A dictionary is an associative array where a name is mapped to an actual instance of an object. In this case, the frame instance name is mapped to a frame instance C++ object.

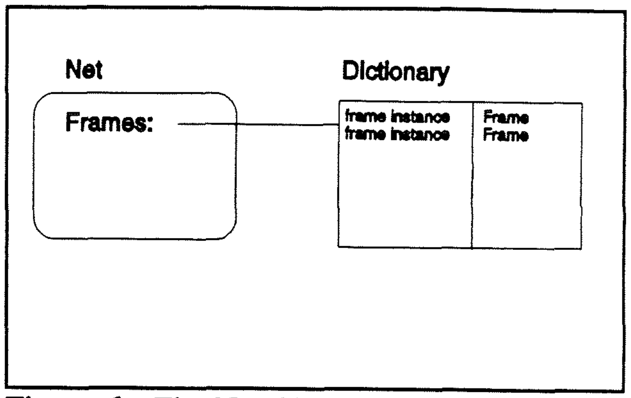


Figure 6 - The Net Class Structure

The Frame Class

The frame class is the central component of the semantic network. A frame is uniquely identified by its name and instance number and contains value and relation slots illustrated by Figure 7. The frame class has several methods that permit the addition, and deletion of value and relation slots, in addition to methods that return information about the frame itself.

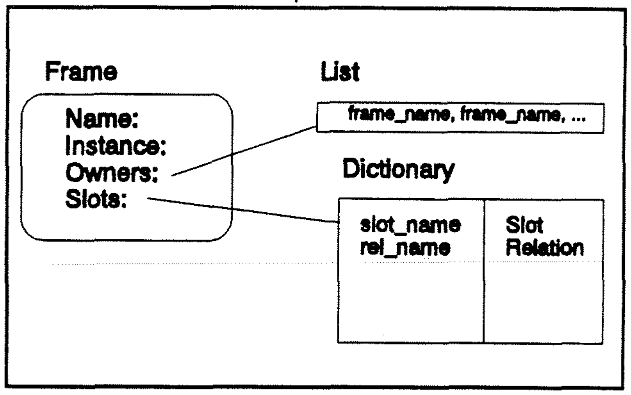


Figure 7 - The Frame Class Structure

The Slot Class

The slot class is an abstract class and is used to derive new classes. This provides a standard interface for all the slot classes. With this class the network can always make certain assumptions about the methods it can call in any derived slot class. As Figure 8 shows there are no actual data members present in this class--this is why it is abstract.

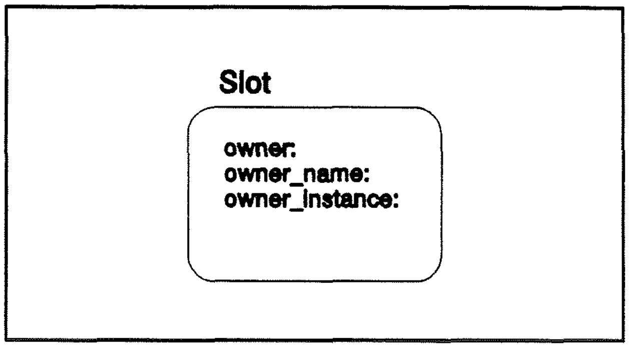


Figure 8 - The Slot Class Structure

The Value Class

The value class is the C++ representation of the CLIPS value fact and is derived from the slot abstract class. This class holds multifield values in the form of strings, integers, and floating points and provides methods for the manipulation of these multifield items. The structure represented in Figure 9 shows the slot class abstract members as well as some additional members--this is an example of inheritance in C++.

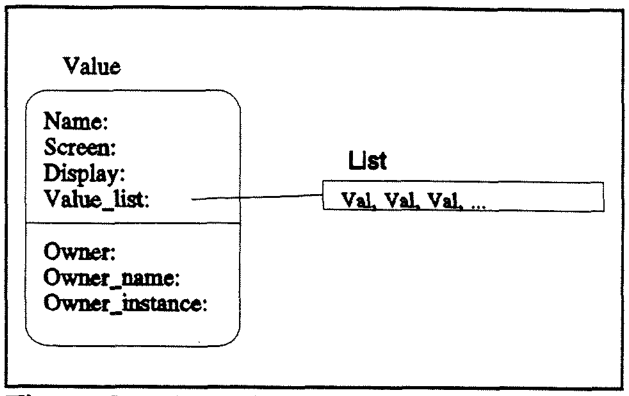


Figure 9 - The Value Class Structure

The Relation Class

The relation class, like the value class, is also a direct representation of a CLIPS fact--the relation fact; it is also derived from the slot class. This relation represents a "has-a" relationship, and when this object is inserted as a slot in a frame object, it defines the relationship between two frames. In addition, when a relation is created, the frame pointed to by the relation is notified that a relationship has been established. This allows a frame to know what frames have relations pointing to it. Figure 10 shows the structure of the relation class.

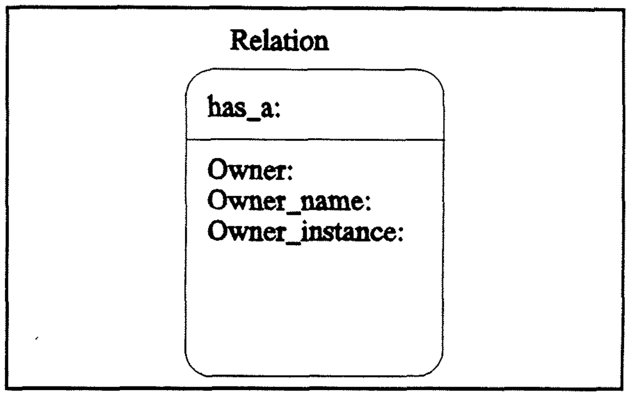


Figure 10 - The Relation Class Structure

The Val Class

The val class is used to represent all the types of data that could be used in a value class (i.e. strings, integers, floating points). Instances of this class are used by the value class to store values for value facts; the value class has a list of val objects as one of its data items. Figure 11 depicts the structure of the val class.

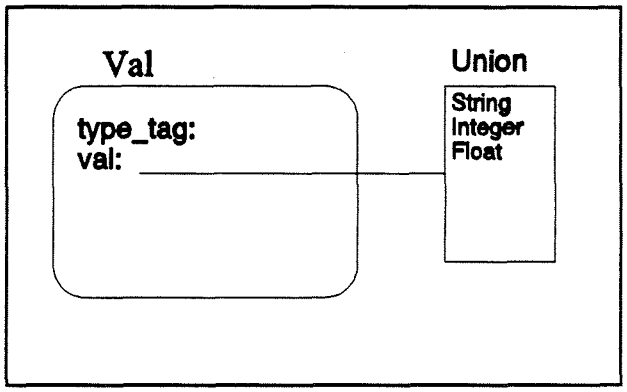


Figure 11 - The Val Class Structure

THE FRAME PARSER

An alternative to using class methods for constructing a network is the frame parser. The frame parser was designed to accept an input language that specifies CLIPS facts in the same format as the CLIPS representation uses and is listed in Appendix A. Using the frame parser relieves the user from having to create new frame class instances, reducing the complexity of the coding effort. This language does not provide queries about the network structure; the user must rely on the class methods.

In addition to the fact information, an action is prepended to the fact to indicate what action is to be taken with the fact. There are three supported actions: ADD, MODIFY, and DELETE. For example, the keyword ADD is used to add a new frame to the semantic network.

EXAMPLE USES OF THE C++ FRAME CLASSES

Several applications have been written which use the C++ frames in a procedural environment. The applications are suited to an environment such as C++ much more than an expert system shell like CLIPS; these types of programs are not easily expressed using CLIPS.

The Design Interface

One of the primary objectives of the Design Interface was to view changes in the semantic network as they occurred; the user should be able to specify the desired slots to view, and the Design Interface would automatically update the display as values change.

As facts come from the blackboard, they are parsed using the frame parser, described previously, and are stored in the semantic network of the Design Interface. In addition, a module in the Design Interface tries to match the incoming fact with a user-specified set of slots. If there is a match, the values are then displayed or updated, which ever case is appropriate.

The Pre-Design Module

Within the ICADS model the Pre-Design Module (PDM) is used to construct a building design starting with collections of objects that represent the spaces that are to be included in the final building structure as denoted by the Project Design Object Frames (PDO). The PDOs are loaded into the PDM via the frame parser which was described above.

Once the PDO frames are loaded into the system the PDM can now make queries about these frames. The spaces frames (PDO frames that represent spaces) are displayed as circles on the screen that the user can select and position as desired. Criteria are then specified by the user to assist the PDM in choosing a central space for the layout of the structure.

After the layout is agreed upon by user the building layout is sent to a CAD system as draw commands for display. A geometry interpreter notices these draw commands and sends messages, that are basically equivalent to the grammar specified by the frame parser, to a blackboard. Once these statements are received by the blackboard and a representation of the semantic network is built in CLIPS, the same messages are sent to any IDTs that require them.

This technique allows IDTs to be written in either C++ or CLIPS. In the case of CLIPS the messages sent from the blackboard are placed in the fact list and the CLIPS rules will fire accordingly. If C++ is used the messages are handled by the frame parser which will build the same semantic network structure using the frame classes. These frame classes can then be used by functions written in C++.

CONCLUSIONS

Executing expert systems in a distributed environment has allowed for an increased level of complexity to be introduced. By having multiple representations, both procedural and expert system, available to programmers, the appropriate strengths of each paradigm can be used for a particular application.

The experiences of the ICADS project have proven both methods of representation to be useful. The sample applications discussed previously would not be as easily implemented in the CLIPS environment. Although, the introduction of the object paradigm to CLIPS 5.0 may remove the need to implement the C++ frames under certain circumstances. However, CLIPS 5.0 will never completely replace the C++ frame representation; the C++ representation allows existing applications to interface into the semantic network in an intuitive and direct manner.

REFERENCES

Assal, H. and L. Myers (1990). *An Implementation of a Framebased Representation in CLIPS.* First CLIPS Conference Proceedings, Houston, Texas. pp. 570-580.

Barr, A. and E. Feigenbaum (1981). *Frames and Scripts.* The Handbook of Artificial Intelligence. Vol. I., William Kaufmann, Stanford, CA.

Coyne, R. C., M. A. Rosenman, A. D. Radford, M. Balachandran and J.S. Gero (1990). *Knowledge-Based Design Systems.* Addison-Wesley, Reading.

Giarratano J., G. Riley (1989). *Expert Systems: Principles and Programming.* PWS-Kent, Boston.

Myers, L. and J. Pohl (1991). *Computer-Based Intelligent Design Assistance: Concepts and Strategies.* First International Conference on Artificial Intelligence in Design. Edinburgh, Scotland, U.K.

NASA (1989). *CLIPS Reference Manual: Version 4.3 of CLIPS.* Artificial Intelligence Section, Lyndon B. Johnson Space Center. Houston, TX.

Pohl, J., L. Myers, A. Chapman, L. Chirica, J. Snyder, H. Assal, J. Taylor, C. Johnson, D. Johnson (1990). *Knowledge-Based CAAD and the CLIPS Expert System Shell.* Technical Report, CADRU-04-90, CAD Research Unit, Design Institute, Cal Poly, San Luis Obispo, CA.

Pohl, J., L. Myers, A. Chapman, J. Snyder, H. Chauvet, J. Cotton, C. Johnson, D. Johnson (1991). *ICADS Working Model Version 2 and Future Directions.* Technical Report, CADRU-05-91, CAD Research Unit, Design Institute, Cal Poly, San Luis Obispo, CA.

Pohl, J., L. Myers, A. Chapman, J. Cotton (1989). *ICADS: Working Model Version 1.* Technical Report, CADRU-03-89, CAD Research Unit, Design Institute, Cal Poly, San Luis Obispo, CA.

Taylor, J. (1990). *A Framework for Multiple Cooperating Agents in an Intelligent Computer-Aided Design Environment.* (Masters Thesis). School of Architecture and Environmental Design, Cal Poly, San Luis Obispo, CA.

Taylor, J. and L. Myers (1990). *Executing CLIPS Expert Systems in a Distributed Environment. First CLIPS Conference Proceedings*, Houston, Texas. pp. 686-695.

APPENDIX A - Frame Parser Grammar

```
fact:
        YADD addactions
|       YMOD modactions
|       YDEL delactions
;

addactions:
        YFRAME frame_part
|       YVALUE value_part
|       YRELATION relation_part
;

modactions:
        YVALUE value_part
;

delactions:
        YFRAME frame_part
|       YVALUE value_part
|       YRELATION relation_part
;

frame_part:
        class instance
;

value_part:
        class attribute instance value_list
;

relation_part:
        class class instance instance
;

value_list:
        value
|       value_list value
;

value:
        YSTR
|       YINT
|       YREAL
;

class:
        YSTR
;

instance:
        YINT
;

attribute:
        YSTR
;
```

N92-16586

OBJECT-ORIENTED KNOWLEDGE REPRESENTATION FOR EXPERT SYSTEMS

Stephen L. Scott

Hughes Information Technology Company
Washington Engineering Laboratory

Abstract: Object-oriented techniques have generated considerable interest in the AI community in recent years. This paper discusses an approach for representing expert system knowledge using classes, objects, and message passing. The implementation is in version 4.3 of NASA's CLIPS, an expert system tool that does not provide direct support for object-oriented design. The method uses programmer-imposed conventions and keywords to structure facts, and rules to provide object-oriented capabilities.

1. INTRODUCTION

A typical expert system consists of a rules, facts, and an inference engine. Although many types of problems can be addressed using this knowledge representation model, others may require features afforded by logical, network, or frame based models (Mylopoulos and Levesque, 83). Recent interest in object-oriented design has suggested an object-based view of knowledge representation, combining elements of several of these mechanisms (Leung and Wong, 90).

Object-oriented design elements, such as classes, objects, and messages, can be implemented in an expert system that does not directly support these capabilities by using facts containing object-oriented keywords and using rules to manipulate these facts. A fact with an object-oriented keyword will be referred to as a "structured fact". The use of facts that contain keywords in certain fields bears similarity to the use of an IS-A or A-KIND-OF link in a semantic network. The object-oriented keyword technique is used by (Assal and Meyer, 1990) in the implementation of a frame-based knowledge representation mechanism. Before proceeding further with the details of this approach, it may be useful to clarify the terminology, adapted from (Meyer, 88), that will be used in the remainder of this paper.

A *class* is a structure defined prior to run-time that identifies data and procedural characteristics of a program entity. Classes can be implemented in an expert system using a set of structured facts that describe features that characterize the class. An *object* is a run-time instance of a class. An object can be represented in an expert system by a structured fact that uniquely identifies the object. Additional structured facts can be used to store the current feature values of the object. A *feature* is a data characteristic associated with a class. Features can be of any data type, such as integer, real, character, or string. A *method* is a procedural characteristic associated with a class. In an expert system, methods can be implemented using rules. A *message* is a structured fact that contains either a request for the current value of an object's feature, or a request to initiate a method associated with an object. The use of messages is one of the distinguishing characteristics of the object-oriented methodology.

2. IMPLEMENTING CLASS AND OBJECT STRUCTURES

2.1 Class Definition Using Facts

A class is defined by asserting one or more facts of the form

```
(CLASS          <class-name>        <parent-class>)
```

This declaration indicates that <class-name> is a subclass of <parent-class>. By convention, in the first CLASS declaration for a class, the identifiers used for <class-name> and <parent-class> should be the same. This is required by the syntax of the rule used to instantiate objects from class declarations. Note that multiple inheritance is accomplished by allowing multiple class declarations, as in the following:

```
(CLASS          <class-name> <class-name>)
(CLASS          <class-name> <parent-class-1>)
(CLASS          <class-name> <parent-class-2>)
.
.
.
(CLASS          <class-name> <parent-class-N>)
```

A class feature is specified as such

```
(HAS-FEATURE    <class-name> <feature-name>
                             [default-feature-value(s)] )
```

where [default-feature-value(s)] consist of one or more data items that comprise the default values of this feature.

A class method is specified using a fact such as

```
(HAS-METHOD     <class-name> <method-name>)
```

The actual implementation of the method is not specified here, nor is any parameter information given at this time.

The following example, employing the CLIPS deffacts fact structuring construct, illustrates a typical class declaration. The length and width fields have been assigned arbitrary default values.

```
(deffacts       rectangle-class
        (CLASS          rectangle       rectangle)
        (HAS-FEATURE    rectangle       length          2)
        (HAS-FEATURE    rectangle       width           5)
        (HAS-FEATURE    rectangle       area            10)
        (HAS-METHOD     rectangle       rectangle-area))
```

2.2 Object Instantiation Using Facts

An object is created by declaring the existence of the object and its features. The instantiation is accomplished by asserting a fact of the following form

```
(INSTANCE       <object-id>     <class-name>)
```

where <object-id> is a unique identifier associated with this object, and <class-name> is the name of a parent class.

The features of the object are automatically asserted with appropriate default values by a rule in the expert system that tests for the presence of a newly instantiated object. The rule asserts facts that make the connection between this particular object and its features. For each feature, a fact of the following form is asserted.

```
(HAS-FEATURE            <object-id>    <feature>
        [default-value(s)])
```

Similarly, methods associated with an object can be declared by asserting facts such as:

```
(HAS-METHOD             <object-id>    <method-name>)
```

The following example, taken from the rectangle class presented earlier, illustrates a typical object instantiation.

```
(INSTANCE        a-box        rectangle)
(HAS-FEATURE     a-box        length          2)
(HAS-FEATURE     a-box        width           5)
(HAS-FEATURE     a-box        area           10)
(HAS-METHOD      a-box        rectangle-area)
```

3. MANAGING OBJECTS

3.1 Top Level Object Management Facts

While these conventions provide a way to structure classes and objects, it is also useful to provide some "system functions" to manage objects. Some typical operations might include getting and setting object feature values, and creating and destroying objects. To get an object's values, a fact of the following form is asserted.

```
(GET        <object-id>    <feature>)
```

An object's feature is set using a fact such as

```
(SET        <object-id>    <feature>    <value>)
```

These facts are used to trigger the GET and SET rules (see 3.2) that bring about the appropriate functionality by reading or updating the feature of the object.

To create an object, a fact of this form is asserted

```
(CREATE     <object-id>    <class>)
```

An object is destroyed by asserting a fact of the following form

```
(DESTROY    <object-id>)
```

These facts trigger the CREATE-INSTANCE and DESTROY-INSTANCE rules that instantiate or remove the object and its associated features.

It is also useful to provide other functions to return current information about a given object. Listing the features of an object may be accomplished by asserting the following fact

```
(SHOW-FEATURES <object-id>)
```

Similarly, listing the parent class or classes of an object may be accomplished by asserting a fact such as this.

```
(SHOW-CLASS        <object-id>)
```

These facts cause the SHOW-FEATURES and SHOW-CLASS rules to display the desired information.

It should be noted that these are top level functions for the benefit of the programmer. Analogues to the top level GET and SET functions are provided at the object level by the messages RETURN-VALUE, REQUEST, and APPLY-METHOD, discussed in further detail below.

3.2 Top Level Object Management Rules

The fact structuring conventions described so far do not provide the functionality needed, although they can be used to trigger rules that do. In the following discussion of the implementation of these rules, it is necessary to introduce the syntax of the CLIPS expert system. A complete presentation of the CLIPS language may be found in (Giarratano, 89) and in (Giarratano and Riley, 89).

The GET rule is used to obtain the current value of a feature of a particular object. It is implemented as follows.

```
(defrule        GET
        (GET                ?object-id      ?feature)
        (INSTANCE           ?object-id      ?class)
        (HAS-FEATURE        ?object-id      ?feature        $?values)
=>
        (printout t     ?object-id " has feature " ?feature)
        (printout t     " with value " $?values crlf))
```

The SET rule is similar, however, it contains additional code to manage the retraction of the old HAS-FEATURE and the assertion of a new HAS-FEATURE fact.

```
(defrule        SET
        (INSTANCE  ?object-id       ?class)
        ?x <- (SET     ?object-id       ?feature        $?new-values)
        ?y <- (HAS-FEATURE          ?object-id      ?feature        $?values)
=>
        (retract ?x ?y)
        (assert (HAS-FEATURE ?object-id ?feature $?new-values)))
```

Object instantiation is accomplished using a two-step process. First, CREATE-INSTANCE recursively traverses the inheritance chain, asserting facts that declare this object to be an instance of each of its ancestor classes. In the second phase, CREATE-FEATURES asserts facts to declare this object's features, and CREATE-METHODS asserts facts to declare this object's methods.

```
(defrule        CREATE-INSTANCE
        (CREATE             ?object-id      ?class)
        (CLASS              ?class          ?parent-class)
=>
        (assert         (INSTANCE ?object-id ?parent-class)))
```

```
(defrule        CREATE-FEATURES
        (INSTANCE        ?object-id      ?class)
        (HAS-FEATURE     ?class
                ?feature         $?default-values)
=>
        (assert
                (HAS-FEATURE      ?object-id      ?feature $?default-values)))

(defrule        CREATE-METHODS
        (INSTANCE        ?object-id      ?class)
        (HAS-METHOD      ?class          ?method-name)
=>
        (assert (HAS-METHOD       ?object-id      ?method-name)))
```

Object deletion is accomplished similarly. Instances of an object and any features and methods of the object must be removed. DESTROY-INSTANCE handles the former, and DESTROY-FEATURES and DESTROY-METHODS the latter.

```
(defrule        DESTROY-INSTANCE
        (DESTROY                ?object-id)
        ?x <- (INSTANCE     ?object-id ?class)
=>
        (retract ?x))

(defrule        DESTROY-FEATURES
        (DESTROY                ?object-id)
        ?x <- (HAS-FEATURE      ?object-id ?feature $?values)
=>
        (retract ?x))

(defrule        DESTROY-METHODS
        (DESTROY                ?object-id)
        ?x <- (HAS-METHOD       ?object-id ?method-name)
=>
        (retract ?x))
```

The SHOW-FEATURES rule displays all the features of a particular object. It is implemented as follows:

```
(defrule        SHOW-FEATURES
        (SHOW-FEATURES ?object-id)
        (INSTANCE        ?object-id      ?class)
        (HAS-FEATURE     ?object-id      ?feature $?values)
=>
        (printout t ?feature " with value " $?values crlf))
```

The SHOW-CLASS rule, listing the parent class or classes of an object, is similar.

```
(defrule        SHOW-CLASS
        (SHOW-CLASS      ?object-id)
        (INSTANCE        ?object-id      ?class)
=>
        (printout t      ?object " is instance of class " ?class crlf))
```

4. MESSAGE PASSING

A message can be either a request for data or the initiation of a procedural action. Both types of messages can be implemented using structured facts and rules.

4.1 Feature Extraction Using Facts

In 3.2, the GET rule was used to print the value of a feature. It is also useful to extract an object's feature value and make the information available for use by other objects. Such a request for data can be made by asserting a fact of the following form:

```
(REQUEST    <calling-object-id>
            <target-object-id>      <feature>)
```

The <calling-object-id> identifies the object that initiated the request, the <target-object-id> shows which object is being queried, and <feature> indicates the feature of interest in the target object. The REQUEST fact is detected by a general "request manager" (see 4.3) rule that removes the REQUEST fact, polls the target object for the requested value, and creates a reply to the message by asserting a fact of the following form:

```
(RETURN-VALUE   <calling-object-id>   <feature>     <value>)
```

4.2 Method Invocation Using Facts

A request to invoke a method is accomplished by asserting a fact such as

```
(APPLY-METHOD   <calling-object-id>   <target-object-id>
            <method> [optional parameters])
```

Again, the <calling-object-id> identifies the object that requested the invocation, the <target-object-id> shows which object is being queried, and <method> indicates the method of interest in the target object. In some method invocations, it may be necessary to pass one or more parameters. Unlike the generalized REQUEST operation above, where a single rule can handle all requests by all objects, each method requires a separate rule. The rule performs the necessary computation, and creates a reply to the message by asserting a fact of the following form:

```
(RETURN-VALUE   <calling-object-id>   <feature>     <value>)
```

4.3 A Rule to Implement Feature Extraction

The rule that manages REQUEST messages is as follows:

```
(defrule        REQUEST-MANAGER
        ?x <- (REQUEST      ?caller-id
                            ?target-id      ?feature)
        (HAS-FEATURE        ?target-id      ?feature        $?value)
        (INSTANCE           ?target-id      ?target-class)
=>
        (retract ?x)
        (assert
                (RETURN-VALUE ?caller-id        ?feature        $?value)))
```

The rule requires the following: a REQUEST fact must exist, the requested feature must be declared as a feature of the target object, and the target object must have been previously instantiated. The (retract ?x) statement removes the REQUEST fact.

4.4 Rules to Implement Method Invocation

Each method defined in a class must be accompanied by an appropriate rule to implement the method. A uniform naming convention, such as the concatenation of the class name, a hyphen, and the method name, can be very useful. The following rule is an example of a method to compute the area of a rectangle.

```
(defrule        rectangle-area
?x <- (APPLY-METHOD       ?caller-id       ?target-id
                rectangle-area)
        (HAS-FEATURE      ?target-id       length  ?length)
        (HAS-FEATURE      ?target-id       width   ?width)
        (INSTANCE         ?target-id       rectangle)
=>
        (retract ?x)
        (assert
        (RETURN-VALUE     ?caller-id       area    =(* ?length ?width))))
```

This rule requires the following: a request to apply the "area" method must exist, the target object must have a length and width feature, and the target object must have been previously instantiated. The RETURN-VALUE fact contains the desired arithmetic result, where the =(* ?length ?width) statement is used to calculate the result of the expression and store the value as a field of a fact. The (retract ?x) statement removes the APPLY-METHOD fact.

4.5 Message Cleanup

In the feature extraction rule (4.3) and the method invocation rule (4.4), statements were explicitly included to remove the message fact that caused the rule to fire. It is useful to consider a general purpose message cleanup method, similar to the "garbage collection" operation performed by most symbolic computation implementations. A low priority rule can be used to remove all messages and interim results after each message processing cycle. This ensures that old messages or return value facts are not accidentally reused or misused at a later time.

In order to simplify the implementation of the garbage collection rule, a fact containing each object-oriented command keyword is used. As command keywords are expected to occur in the first field of a structured fact, any fact containing a command keyword in its first field at the end of a message processing cycle can be assumed to be eligible for garbage collection.

A simple garbage collection rule can be implemented as follows. The list of commands is stored in the COMMAND-LIST fact, and the garbage collection rule tests to ensure that the selected fact contains a command.

```
(deffacts COMMAND-LIST
        (COMMAND-LIST
                RETURN-VALUE
                CREATE          DESTROY
                SET             GET
                SHOW-ATTRIBUTE          SHOW-CLASS))
```

```
(defrule        garbage-collection
        (salience -5)
        (COMMAND-LIST $?cmd-set)
        ?x <- (?cmd&:(member ?cmd $?cmd-set) $?cmd-tail)
=>
        (retract ?x))
```

5. USING OBJECT-ORIENTED KNOWLEDGE REPRESENTATION

The techniques described in this problem were used as the basis for an object-oriented knowledge base that was incorporated in the software prototype of a satellite metatdata information system. This domain is typified by very large quantities of data, suggesting a fundamental need for intelligent query and browse features. More thorough treatments of satellite information systems can be had in (Corey and Carnahan, 90) and (Roeloffs and Campbell, 90).

The project constraints dictated that the expert system component had to be portable, extensible, flexible, and robust. In addition, it had to be developed rapidly, and interface readily with existing C and C++ software on UNIX[1] platforms. The NASA CLIPS tool was virtually ideal for this task, although it lacked support for object-oriented modelling.

One of the lessons learned from this experience was that an object-oriented knowledge base can be fairly easy to integrate with traditional software. The object-oriented approach allowed the existing C programs to be minimally affected. The expert system functioned transparently to the C code, providing services through the use of the assert() and run() function calls.

The expert system software architecture was based on a "state-machine" model, with control states similar in some respects to the "read evaluate print garbage-collect" cycle of a LISP interpreter. The expert system, once initialized, entered a "read" state, where it waited for requests for information. Messages, in the form of facts asserted into the knowledge base by procedural C/C++ code, would either supply information about the constraints of the current query, or cause the system to enter an "evaluation" state.

In the "evaluation" state, the expert system applied rules to find information meeting the criteria dictated by messages. The derived information, typically tokens in the knowledge base that matched the left-hand-side (LHS) of a rule, would be processed on the right-hand-side (RHS) of the rule as parameters to a user defined C function that appended the information to a globally available linked list structure.

After a given request was satisfied, the expert system entered a "garbage-collection" state. All messages, commands, and intermediate facts were eliminated, and the system then returned to the initial "read" state to wait for additional messages.

The application program that invoked the expert system handled the "print" phase. Information was taken from the linked list structure generated by the expert system and used by the C program.

This method made the interfaces between the C/C++ code and the expert system very straightforward. The C programs made use of the expert system by including the <clips.h>

[1] UNIX is a trademark of AT&T Bell Laboratories.

headers, loading and initializing selected rule and fact files, declaring a global linked list of strings to hold query results, and proceeding with the application processing.

When information was needed from the expert system, a character string containing a syntactically valid data request (typically in the form of a constraint or a command) was assembled and "asserted" into the expert system. When all requests were in place, a call to the run(-1) function was made, activating the expert system rules. When the expert system had finished, the results of the query were available in a linked list. The C program could proceed at this point with its processing, which typically involved presenting the information onto an X Windows[2] dialog box.

6. CRITICAL ASSESSMENT AND PERFORMANCE ISSUES

The use of structured facts and rules as suggested in this work is no different than the use of other fact structuring conventions, such as the Object Attribute Value (OAV) triple or the IS-A or AKO (A-Kind-Of) links in semantic networks. The approach offers more functionality than the OAV model, since procedural and data elements can be associated with an object.

The method may offer certain performance advantages over more sophisticated systems, since expert systems are optimized to process rules and facts, rather than objects and messages. The method is reasonably portable, although rules may need to be recoded in order to accommodate the particular syntax of the tool being used.

Perhaps the greatest advantage of this method is its high degree of flexibility. While a number of expert system shells incorporate object oriented extensions, few if any allow the user to completely redefine the syntax of the object manipulation language at will.

A detrimental performance aspect of this method is that using facts to implement inheritance can cause a great many facts to be asserted when objects are instantiated. If deep inheritance chains are modeled, multiple instantiations of an object far down the chain may begin to pose memory and speed problems on the expert system inference engine.

Other, more subtle problems also exist. The method does not account for feature inconsistencies, hence it is possible for an object to inherit the same feature name from different parent classes if the default values are different. This will result in an inconsistency, since the object will manifest two features with the same name yet different values. Other object-oriented concerns, such as selective inheritance, or precedence of local features or methods over inherited features or methods are not addressed.

7. CONCLUSIONS

This paper has introduced a method for defining classes, objects, and messages in an expert system. The method can be implemented using conventional hardware and very straightforward expert system tools, and does not require a sophisticated run-time object or message manager. It requires some programmer-imposed conventions on facts and rules, and calls for the use of structured facts containing object-oriented keywords. Promising results have been obtained by incorporating an expert system using object-oriented knowledge representation with traditional C code.

[2] The X Window System is a trademark of the Massachusetts Institute of Technology.

8. REFERENCES

(Assal and Myers, 90) Assal, Hisham, and Leonard Myers. "An Implementation of a Frame-based Representation in CLIPS." First CLIPS Conference Proceedings, NASA Conference Publication 10049, Volume II, pp. 570-580.

(Corey and Carnahan, 90) Corey, Stephen M., and Richard S. Carnahan. "Knowledge Structure Representation and Automated Updates in Intelligent Information Management Systems." 1990 Goddard Conference on Space Applications of Artificial Intelligence, NASA Conference Publication 3068, May 1990, pp. 271-282.

(Giarratano and Riley, 1989) Giarratano, Joseph C., and Gary Riley. Expert Systems: Principles and Programming. Boston, PWS-Kent Publishing, 1989.

(Giarratano, 1989) Giarratano, Joseph C. CLIPS User's Guide, Artificial Intelligence Section, Lyndon B. Johnson Space Center, NASA. 1989.

(Leung and Wong, 90) Leung, K. S., and M. H. Wong. "An Expert System Shell Using Structured Knowledge." IEEE Computer. March 1990. p38-47.

(Meyer, 88). Meyer, Bertram. Object Oriented Software Construction. New York, Prentice Hall, Inc., 1988.

(Mylopoulos and Levesque, 83) Mylopoulos, John and Hector Levesque. "An Overview of Knowledge Representation." in Brodie, M.L., Mylopoulos, J., and Schmidt, J.V.,(eds.) On Conceptual Modelling: Perspectives from Artificial Intelligence, Databases, and Programming Languages, Springer-Verlag, New York, 1983.

(Roeloffs and Campbell, 90) Roeloffs, Larry H., and William J. Campbell. "Using Expert Systems to Implement a Semantic Data Model of a Large Mass Storage System." 1990 Goddard Conference on Space Applications of Artificial Intelligence, NASA Conference Publication 3068, May 1990, pp. 253-270.

SESSION 4 B

PRECEDING PAGE BLANK NOT FILMED

N92-16587

LINKFINDER: AN EXPERT SYSTEM THAT CONSTRUCTS PHYLOGENIC TREES

James Inglehart and Peter C. Nelson

Department of Electrical Engineering and Computer Science (M/C 154)
The University of Illinois at Chicago
Chicago, Illinois 60680 e-mail: inglehar@uicbert.eecs.uic.edu

Abstract. An expert system has been developed using CLIPS that automates the process of constructing DNA sequence-based phylogenies—trees or lineages that indicate evolutionary relationships. LinkFinder takes as input homologous DNA sequences from distinct individual organisms. It measures variations between the sequences, selects appropriate proportionality constants, and estimates (if possible) the time that has passed since each pair of organisms diverged from a common ancestor. It then designs and outputs a phylogenic map summarizing these results.

LinkFinder can find genetic relationships between different species, and between individuals of the same species, including humans. It was designed to take advantage of the vast amount of sequence data being produced by the celebrated Genome Project, and should be of great value to evolution theorists who wish to utilize this data, but who have no formal training in molecular genetics.

The mathematical basis of LinkFinder's DNA sequence analysis is remarkably simple. Evolutionary theory holds that distinct organisms carrying a common gene inherited that gene from a common ancestor. Homologous genes vary from individual to individual and species to species, and the amount of variation is now believed to be directly proportional to the time that has passed since divergence from a common ancestor. The proportionality constant must be determined experimentally; it varies considerably with the types of organisms and DNA molecules under study. Given an appropriate constant, and the variation between two DNA sequences, a simple linear equation gives the divergence time.

INTRODUCTION

Plants and animals have long been classified according to the similarities and differences in their form and structure. When the concept of evolution was first proposed, biologists naturally used these morphological features to establish phylogenies—evolutionary trees. Proposed lineages were modeled as paths (from root to leaf) through the tree, and closely related species were shown as parallel branches emanating from a common ancestor node.

Constructing a tree based on morphology has always been highly subjective, more of an art than a science. Most classifications are based on anatomy, but microscopic species, such as bacteria, are more commonly distinguished by chemical analyses. Morphology is good for classifying evolutionary relationships at certain scales, but it indicates neither the large-scale structure of evolution nor the fine details. Mice are clearly much closer cousins to humans than bacteria, but the details of how we diverged from mice are obscure, despite

PRECEDING PAGE BLANK NOT FILMED

numerous anatomical similarities between mice and men. And bacteria seem so alien to us (despite some common features) that the details of how we diverged from them seem impossible to determine from morphology alone.

Fortunately, sophisticated new methods of genetic analysis have arisen to challenge morphology as the prime determinant of family trees. We now understand that all life processes are ultimately controlled by DNA. This self-replicating molecule is found in every living thing, and it is the key to the structure and complexity of all life on earth. Because of recent technological advances, biologists and geneticists are now able to ascertain the atomic structure of an individual organism's DNA molecules. Since the common ancestor of all life on earth is believed to have been a single, primal molecule of DNA, evolution can be viewed as simply the development of new forms of DNA through accidental mistakes in duplication.

Because all DNA is constructed exactly the same way (only the specific base sequence differs from individual to individual), DNA analysis provides an objective basis for discerning evolutionary relationships. Closely related species should have closely similar DNA sequences. Distantly related species will have far more dissimilarities. But if all life is truly related (through some primal common ancestor) then even the most disparate life forms should share some similarities. All life shares the same chemical basis, so total DNA sequence divergence (to the point of zero resemblance between distantly related organisms) should not occur (Doolittle et al. 1986).

Some phylogenic trees based on genetic analyses are radically different from traditional morphological trees, challenging our traditional views. For example, comparative studies of the Ursidæ, or bear family, and Procyonidæ, or raccoon family, indicate that the giant panda belongs in the bear family, whereas the red panda belongs in the raccoon family (O'Brien et al. 1985). Phylogenically, then, there is really no such thing as a "panda," whereas bears and raccoons really exist.

Similar results have been obtained in primate studies. Comparative analyses of the beta-globulin genes of humans and the great apes (the chimpanzee, gorilla, and orangutan) indicate that humans are more closely related to chimpanzees than chimpanzees are to the other great apes (Miyamoto et al. 1988). This contradicts the common notion (based on the morphological similarities of apes) that humans diverged from the common ancestor of the great apes. In fact, the semantic label "ape" has now lost its phylogenic connotation, since it makes more sense to lump humans and chimpanzees together than to group chimpanzees with the other great apes (Ueda et al. 1986).

The most controversial result (cf. Latorre et al. 1986) has been the "Mother Eve hypothesis" of Rebecca Cann and her colleagues. Their studies of worldwide human mitochondrial DNA indicate that all humans alive today have a common ancestor, a woman, who lived in Africa roughly 200,000 years ago (Cann et al. 1987). Prior to this discovery, anthropologists generally assumed that the the most recent common human ancestor must have lived closer to one million years ago (Stoneking et al. 1986).

The Genetic Code

The genetic information in DNA is encoded within strings of nitrogenous bases. There are four of these: adenine (A), cytosine (C), guanine (G), and thymine (T). A DNA *sequence* can be thought of as simply a long string of these four letters in some combination.

A string of three bases codes for an amino acid; there are exactly twenty of these. Proteins are simply long strings of amino acids. Thus, for any protein, there is some corresponding base sequence that acts as a template for that protein; this template is called a *gene*. Genes code for protein production, and chromosomes are simply long strings of genes. A *genome* is the entire gene set of a single organism.

During reproduction, the genome is copied, to be passed on to future generations. Usually the copy is perfect, but sometimes an incorrect base is substituted. Or a gene may be damaged by something in its environment. If the mutation is a minor one (non-fatal) it will be copied and passed on to future generations. Over time, these inheritable mutations will lead to the development of new genotypes within a species, and ultimately to new species.

Algorithmic Tree Construction

These considerations have led to the development of phylogenic tree construction algorithms, which take as input DNA sequence data from living organisms. The input organisms are then "clustered" into larger related units on the basis of their genetic similarities. The more distantly related clusters are then iteratively clustered in turn, until a complete tree is formed. Each internal node on this tree represents a hypothetical ancestor form, a "missing link," joining separate lineages. Leaf nodes represent the input organisms living today. The root node represents the common ancestor of all the leaves on the tree.

Detailed family trees of certain organisms, such as pedigreed animals, are already known. These known trees can be used to test tree construction algorithms. Given genetic data from present-day forms, a good algorithm should reconstruct the known trees. Fruit flies are good test cases, since they have been bred in the laboratory for decades. Some human genealogies extend more than 1,000 years. But no known trees go back far enough in time to link distinct species, such as humans and chimpanzees.

Because known trees are so limited, tree construction algorithms are usually tested on made-up data. An single ancestor DNA sequence is chosen at random, and descendent generations are iteratively created by random base substitutions (Tateno 1985). A good algorithm will correctly reconstruct the entire made-up tree from the DNA sequences of its final generation.

In an accurate morphologically-based tree, the vertical height of lineages is made proportional to the passage of time. Divergence times are deduced from accepted geological time scales by examining the fossil record. Ideally we would like trees constructed by algorithm to also show how long ago each pair of lineages diverged. But finding the correct time scale is difficult, and highly organism-dependent. Bacteria, for example, can mutate much faster than humans. Because of these difficulties, the vertical height of divergent lineages in trees constructed by algorithm is usually made proportional to the "genetic distance" between divergent pairs of organisms. This "distance" is the calculated (or estimated) percentage by which the genomes of the two organisms diverge. When a tree is constructed from genetic data, some attempt is usually made to convert this "distance scale" to a time scale. This conversion usually requires expert knowledge concerning the fossil record, the types of organisms under study, and the types of genetic data being used.

LinkFinder

LinkFinder automates the process of tree construction in two ways: 1) it automatically constructs a tree from genetic data, and 2) it converts (if possible) the distance scale of the initial tree to a time scale. The topology of the final tree is entirely determined by the input data and the tree construction algorithm. But to convert the tree from a distance scale to a time scale, LinkFinder requires an expert system. Our CLIPS-based system takes into account the specific nature of the input data in choosing a conversion, considering both the fossil record and known mutation rates before making a decision.

The fossil record remains the primary source of authoritative evidence on the divergence times of major lineages, and LinkFinder's knowledge base is mostly derived from the published literature on the fossil record. Its knowledge is thus limited to areas where evidence of divergence times has already been found. Since the fossil record is incomplete, there are considerable gaps in the knowledge base, which hopefully will be filled in the future. Other techniques of estimating genetic divergence rates also exist (e.g., Nei and Tajima 1983, Ferris et al. 1983, and Stoneking et al. 1986), and some of these estimates are now being added to LinkFinder's knowledge base. Our primary reason for developing LinkFinder was to take advantage of the explosion of new genetic data being produced by the Genome Project—the ongoing worldwide effort to map and sequence the entire human genome, as well as the entire genomes of several other organisms. We hope to continue developing LinkFinder as the Genome Project progresses, adding new information to its knowledge base as it becomes available. In its present form, LinkFinder is a powerful tool for tree estimation, but it will not be a true, general purpose tree constructor until more complete data is available on rates of divergence.

HOW LINKFINDER WORKS

LinkFinder constructs a phylogenic tree from input genetic data in two distinct stages:

1. The topology of the tree is determined by the unweighted pair-group (UPG) method. At this stage, branch lengths in the tree are proportional to the calculated percent difference between the clustered genotypes.

2. A CLIPS-based expert system attempts to determine an explicit time scale for the tree. It considers known mutation rates and the fossil record (if any) of the organisms in the tree before making a decision.

Estimating Tree Topology

There is as yet no known algorithmic method of tree construction which can reproduce known phylogenic trees with unfailing accuracy. Reconstructed trees are therefore phylogenic *estimates* at best.

The UPG method utilized by LinkFinder is the simplest well-known tree construction algorithm (first proposed by Sokal and Michener in 1958), but it has stood well the test of time. Numerous, far more complicated tree construction algorithms have since been proposed (e.g., Fitch and Margoliash 1967, Farris 1972, Moore et al. 1973, and Tateno et al. 1982) but the overall performance of the UPG method still compares favorably with these (Li 1981).

Genetic data is input to LinkFinder as a two-dimensional array, with each row containing sequence data from a single operational taxonomic unit (OTU), which can be a gene, an individual, a population, a species, or a taxa of higher rank (Moore et al. 1973). Each row also contains a unique label identifying the OTU.

Another input file classifies each OTU according to kingdom, phylum, class, etc. This taxonomic information will be needed by LinkFinder's expert system.

Sequence data is coded either as a string of amino acids, or (more typically) as a string of bases. The four possible bases can be coded with the four letters A, C, G, and T, and the twenty possible amino acids can be coded using any convenient choice of twenty distinct characters. OTU labels are distinct name strings chosen to identify the OTUs under study, but in the examples below single letters will be used as labels for hypothetical OTUs.

The UPG method utilized by LinkFinder makes the following assumptions about its OTU sequence data:

1. Genetic sequences have been chosen to be *homologous* between OTUs. I.e., if the sequence is a gene coding for some protein which varies from OTU to OTU, each OTU must have inherited that gene from some common ancestor, so that each individual contemporary form represents divergence from the same ancestral form (Tateno 1985). For example, the gene for hemoglobin, found in some form in all mammals, is homologous in mammals.

2. Contemporary homologous OTU sequences are assumed to have diverged from the ancestral form because of *random* base (amino acid) substitutions in succeeding generations. (Note that changing a single base in a group of three can also change the amino acid that the group codes for.) These substitutions are assumed to be random both in the choice of base (amino acid) and with respect to position in the sequence. This assumption is well supported by our current understanding of genetic mutations (Tateno 1985).

3. The number of base substitutions in all lineages is assumed to be linear over time, i.e., all lineages are assumed to evolve at the same constant rate. This simplifying assumption is realistic in many cases, but it is not strictly true. The actual number of base substitutions in an evolutionary lineage over time is believed by many geneticists to follow a Poisson distribution whose mean is the expected number of base substitutions in that OTU over time. This implies that the actual numbers of base

substitutions in two lineages can differ considerably due to stochastic error (Tateno 1985). When the actual rates for different lineages are very different, tree estimation by the UPG method is sometimes in error. This has motivated the development of more complex tree construction algorithms. However, the UPG method is much simpler to implement, and it works well in the majority of cases (Li 1981).

Assumptions (1) and (2) combined imply that all input genetic sequences must be exactly the same length, with each containing the same number of bases (amino acids). All three assumptions must hold for the constructed tree to be considered valid.

Given an input array of homologous sequence data for n distinct OTUs, LinkFinder starts by computing the genetic distance between every pair of distinct OTUs, and loading these values into an $n \times n$ distance matrix. Genetic distance is simply the percent difference between two distinct sequences, which is calculated by direct comparison. If the pair of bases (amino acids) in the ith position of two sequences don't match, a counter is bumped, and the final count for that pair is divided by the total number of bases (amino acids) in a sequence.

The tree topology is generated from the distance matrix by the following iterative algorithm (from Li 1981):

1. Choose the smallest non-zero distance in the distance matrix. These two closest OTUs will now be clustered together into a single OTU. E.g., if d_{AB} is the shortest distance (as in Table 1), then A and B are the closest OTUs, and the new OTU will be labeled (AB).

2. Draw vertical lines from the chosen nodes A and B to their presumed common ancestor node (as in Figure 1). Make the lines proportional in length to $d_{AB}/2$.

3. Construct a new, smaller distance matrix from the old one by taking the distance between AB and any other OTU, say C, to be the arithmetic average of d_{AC} and d_{BC}—i.e., $d_{(AB)C} = (d_{AC} + d_{BC})/2$ (as in Table 2).

4. Continue the process (1, 2, and 3) until all the initial OTUs are clustered into a single binary tree. The root node of this tree will represent the common ancestor of all the initial OTUs, and the height of the tree will be proportional to the genetic distance (percent divergence) between the hypothetical ancestor and all of its descendent leaf nodes.

In the tables and figures, seven contemporary OTUs are labeled A, B, C, D, E, F, and G, and their initial distance matrix (Table 1) indicates that (AB) should be the first cluster, since the sequences of A and B differ by the smallest amount (3%). The common ancestor of A and B should thus differ 1.5% from each of its descendants, so the parent node linking A and B is placed at a height of 1.5 percentage units (Figure 1).

The second distance matrix (Table 2) gives us (EF) as a cluster with a height of 2.5 units (Figure 2). The third (Table 3) joins (AB) with D to produce the cluster ((AB)D) with an overall height of 3.75 units (Figure 3). Next we get ((EF)G), also 3.75 units high (Figure 4). Then (((AB)D)C), 4.38 units high (Figure 5). We now have only two clusters left, so our hypothetical common ancestor must lie between them. If we call this root node

Table 1. Initial distance matrix.

	A	B	C	D	E	F	G
A	0	**3**	8	7	10	9	13
B		0	9	8	11	10	14
C			0	9	12	11	15
D				0	9	8	12
E					0	5	9
F						0	6
G							0

Table 2. After one iteration.

	(AB)	C	D	E	F	G
(AB)	0	8.5	7.5	10.5	9.5	13.5
C		0	9	12	11	15
D			0	9	8	12
E				0	**5**	9
F					0	6
G						0

Table 3. After two iterations.

	(AB)	C	D	(EF)	G
(AB)	0	8.5	**7.5**	10	13.5
C		0	9	11.5	15
D			0	8.5	12
(EF)				0	7.5
G					0

Table 4. After three iterations.

	((AB)D)	C	(EF)	G
((AB)D)	0	8.75	9.25	12.75
C		0	11.5	15
(EF)			0	**7.5**
G				0

Table 5. After four iterations.

	((AB)D)	C	((EF)G)
((AB)D)	0	**8.75**	11
C		0	13.25
((EF)G)			0

Table 6. After final iteration.

	(((AB)D)C)	((EF)G)
(((AB)D)C)	0	**12.125**
((EF)G)		0

Figure 1. Initial cluster: (AB).

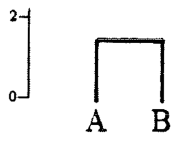


Figure 2. Resultant cluster: (EF).

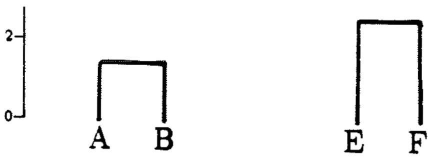


Figure 3. Resultant cluster: ((AB)D).

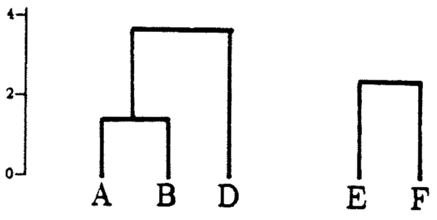


Figure 4. Resultant cluster: ((EF)G).

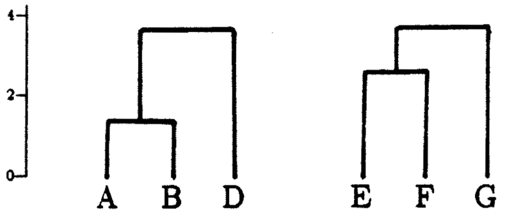


Figure 5. Resultant cluster: (((AB)D)C).

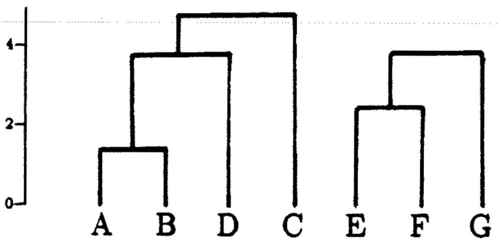
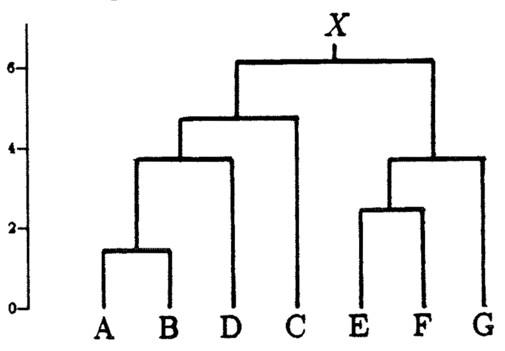


Figure 6. Resultant tree:



X, then our final tree (Figure 6) can be written in line-form as (((AB)D)C)X((EF)G). Its height of 6.06 units gives the average divergence of the seven present-day OTUs from their presumed common ancestor: about 6%.

Estimating Divergence Times

LinkFinder's method of phylogenic tree estimation is a straightforward implementation of the UPG method. It can produce a tree for any reasonable input set of sequence data. However, the vertical axis of a true phylogenic tree represents time, not percent divergence. To produce a true phylogenic tree, LinkFinder must convert the vertical axis to units of time.

Unfortunately, there is no straightforward way to do this. Divergence rates vary greatly for different organisms and different types of DNA. To make a reasonable conversion, LinkFinder must proceed on a case-by-case basis. It must consider the available experimental evidence on divergence rates for the specific data under study. It needs expert advice.

It was to solve this problem that LinkFinder was equipped with a CLIPS-based expert system. LinkFinder's knowledge base contains current data on divergence rates and times for many basic types of organisms. These data have been culled from selected books and papers on evolution, the fossil record, and genetics. Given the taxa under consideration, LinkFinder can usually make educated guesses about divergence times.

For example, it has been estimated that prokaryotic organisms (like bacteria) diverged from eukaryotic organisms (like plants and animals) roughly 1.8 billion years ago. Plants probably diverged from animals about 1 billion years ago, and animals diverged into vertebrates and invertebrates about 500 million years ago. These facts (from Doolittle et al. 1986) have been entered into LinkFinder's knowledge base, and LinkFinder can use them to obtain conversion factors. Given homologous gene sequences from two OTUs, one human, one bacterial, LinkFinder will assume that their linking ancestor existed 1.8 billion years ago. Based on this knowledge, LinkFinder will postulate a conversion factor for a divergence-scaled tree containing these two OTUs. E.g., if the pair of homologous genes (one human, one bacterial) coded for the metabolic enzyme ***triosephosphate isomerase*** (found in both bacteria and humans), LinkFinder would find a 54% divergence between the pair (Doolittle et al. 1986). This suggests a conversion factor of 33 million years per percent divergence for a tree containing these two OTUs, which is exactly what LinkFinder would propose. If the same tree contained other OTU pairs which were also present in LinkFinder's knowledge base, then other possible conversion factors would also be reported, along with the specific OTUs upon which each conversion was based. If there is good agreement between the various calculated factors, this is strong evidence in favor of an overall time-scale conversion. If the various factors do not agree, then various individual clusters within the tree should probably be assigned their own separate time scales based on the recommended conversions. In its present form, LinkFinder only recommends possible conversions. It leaves the final decision on how to scale the overall tree to the user.

LinkFinder's knowledge base is organized according to the usual taxonomic distinctions. By examining the classification file for each OTU, it can quickly position each OTU within the general biological categories in its knowledge base: prokaryote-eukaryote, plant-animal, vertebrate-invertebrate, fish-amphibian-reptile-mammal, etc. Once it has categorized each OTU, it searches for known divergence times for all OTU pairs. Each divergence time found is reported as a possible time-scale conversion factor for the percent divergence-scaled tree.

LinkFinder's knowledge base is intended to be augmented as new information becomes available. There are tens of millions of different species upon the earth, and divergence rates in general are not known for a given pair of OTUs. Even if they were, it would take considerable time to add so much information to LinkFinder's knowledge base. For now, we have concentrated on entering divergence times of the most basic taxonomic units—the kingdoms, phyla, and classes. In certain taxonomic areas (notably primates/humans) more detailed information has been entered on individual species. Our main purpose in creating LinkFinder has been to develop a prototype of the automated expert phylogenic tree constructor of the future. The great volumes of sequence data being generated by the Genome Project will be valueless without the proper analytic software tools, and detailed phylogenic analyses of these data will be needed before we can determine with any certainty the actual divergence paths taken by the myriad forms of life on earth.

REFERENCES

Cann, R.L., Stoneking, M., and Wilson, A.C. (1987). Mitochondrial DNA and human evolution. *Nature* **325**(1 Jan), 31–36.

Doolittle, R.F., Feng, D.F., Johnson, M.S., and McClure, M.A. (1986). Relationships of human protein sequences to those of other organisms. *Molecular Biology of Homo Sapiens: Cold Spring Harbor Symposium on Quantitative Biology* **51**(part 1), 447–455.

Farris, J.S. (1972). Estimating phylogenetic trees from distance matrices. *Am. Nat.* **106**, 645–668

Ferris, S.D., Sage, R.D., Prager, E.M., Ritte, U., and Wilson, A.C. (1983). Mitochondrial DNA evolution in mice. *Genetics* **105**, 681–721.

Fitch, W.M. and Margoliash, E. (1967). Construction of phylogenic trees. *Science* **155**, 279–284.

Latorre, A., Moya, A., and Ayala, F.J. (1986). Evolution of mitochondrial DNA in *Drosophilia subobscura. Proc. Natl. Acad. Sci. U.S.* **83**, 8649–8653.

Li, W.-H. (1981). Simple method for constructing phylogenic trees from distance matrices. *Proc. Natl. Acad. Sci. U.S.* **78**, 1085–1089.

Miyamoto, M.M., Koop, B.F., Slightom, J.L., Goodman, M., and Tennant, M.R. (1988). Molecular systematics of higher primates: Genealogical relations and classification. *Proc. Natl. Acad. Sci. U.S.* **85**, 7627–7631.

Moore, G.W., Goodman, M., and Barnabas, J. (1973). An iterative approach from the standpoint of the additive hypothesis to the dendogram problem posed by molecular data sets. *J. Theor. Biol.* **38**, 423–457.

Nei, M. and Tajima, F. (1983). Maximum likelihood estimation of the number of nucleotide substitutions from restriction sites data. *Genetics* **105**, 207–217.

O'Brien, S.J., Nash, W.G., Wildt, D.E., Bush, M.E., and Benveniste, R.E. (1985). A molecular solution to the riddle of the giant panda's phylogeny. *Nature* **317**(12 Sep), 140–144.

Sokal, R.R. and Michener, C.D. (1958). A statistical method for evaluating systematic relationships. *Univ. Kansas Sci. Bull.* **38**, 1409–1438.

Stoneking, M., Bhatia, K., and Wilson, A.C. (1986). Rate of sequence divergence estimated from restriction maps of mitochondrial DNAs from Papua New Guinea. *Molecular Biology of Homo Sapiens: Cold Spring Harbor Symposium on Quantitative Biology* **51**(part 1), 433–439.

Tateno, Y., Nei, M., and Tajima, F. (1982). Accuracy of estimated phylogenetic trees from molecular data. *J. Mol. Evol.* **18**, 387–404.

Tateno, Y. (1985). Theoretical aspects of molecular tree estimation. *Population Genetics and Molecular Evolution* (Ohta, T. and Aoki, K., ed.), 293–312, Japan Sci. Soc. Press, Tokyo/Springer-Verlag, Berlin.

Ueda, S., Watanabe, Y., Hayashida, H., Miyata, T., Matsuda, F. and Honjo, T. (1986). Hominoid evolution based on the structures of immunoglobulin epsilon and alpha genes. *Molecular Biology of Homo Sapiens: Cold Spring Harbor Symposium on Quantitative Biology* **51**(part 1), 429–432.

N92-16588

GENERATING TARGET SYSTEM SPECIFICATIONS FROM A DOMAIN MODEL USING CLIPS

Vijayan Sugumaran, Hassan Gomaa and Larry Kerschberg

Center for Software Systems Engineering
Department of Information and Software Systems Engineering
George Mason University, Fairfax, VA 22030

Abstract. The quest for reuse in software engineering is still being pursued and researchers are actively investigating the domain modeling approach to software construction. There are several domain modeling efforts reported in the literature and they all agree that the components that are generated from domain modeling are more conducive to reuse. Once a domain model is created, several target systems can be generated by tailoring the domain model or by evolving the domain model and then tailoring it according to the specified requirements. This paper presents the Evolutionary Domain Life Cycle (EDLC) paradigm in which a domain model is created using multiple views, namely, aggregation hierarchy, generalization/specialization hierarchies, object communication diagrams and state transition diagrams. The architecture of the Knowledge Based Requirements Elicitation Tool (KBRET) which is used to generate target system specifications is also presented. The preliminary version of KBRET is implemented in CLIPS.

1.0 INTRODUCTION

It is widely believed that there is a direct relationship between software reuse and computer software productivity. Researchers are constantly exploring new methods and concepts to improve reuse - a problem that has been solved to a greater degree of success in the computer hardware field. Several models and frameworks have been proposed and discussed in (Biggerstaff and Perlis 89); however, the problem of reuse has not yet been solved satisfactorily. Domain Analysis is a fundamental step towards reuse and is a key factor in the success of reusability. Domain analysis artifacts are more conducive to reuse because they capture the essential objects and functions that characterize the domain.

Parnas initially proposed the idea that it will be more advantageous to build a framework for a family of systems rather than build every individual system from scratch (Parnas 79). He argues that it is worth considering a family of systems when there is more to be gained by analyzing the systems collectively rather than separately, i.e. the systems have more features in common than features that distinguish them. Domain modeling addresses the problem of developing a family of systems in which the traditional system development activities like analysis, specification and design are performed at the application domain level and not at the individual system level.

(Prieto-Diaz 88) states that "In domain analysis, common characteristics from similar systems are generalized, objects and operations common to all systems within the same domain are identified and a model is defined to describe their relationships". Domain analysis also considers the variations among the current systems and must evolve to accommodate unanticipated variations as well.

Thus, a domain model is the representation of the common characteristics and variations among a family of software systems in a given application domain. A computer-based domain model generally captures the static and dynamic aspects of the application domain. The static properties include objects of the domain, attributes of those objects and relationship among them. The dynamic properties include the operations associated with objects and the messages passed between objects. The domain model may also include integrity constraints that express the rules which govern the behavior of objects in the domain.

The primary objective of the domain modeling approach to software construction is to increase reuse, i.e., reuse not only of code modules but also of domain knowledge such as domain requirements, specifications and designs. From the domain model, target systems can be generated by tailoring the domain model, or by a combination of evolving the domain model and then tailoring it. A target system is a member of the family of software systems. Thus, a target system engineer can develop the specification for a target system in terms of the domain model, specified previously by a domain analyst, and does not have to perform systems analysis every time a new target system has to be constructed.

The recent Domain Modeling Workshop held at Austin, Texas, during May 91 and a growing body of literature indicate the emphasis on domain modeling and reuse in software development. Several institutions in industry and academia are pursuing research efforts in domain analysis and domain modeling. At the Center for Software Systems Engineering at George Mason University, we are involved in a software engineering project funded partially by NASA/Goddard Space Flight Center through Computer Technology Associates. In this project, Gomaa et al. have developed a software process model called the Evolutionary Domain Life Cycle (EDLC) model that supports the evolutionary development of families of systems (Gomaa et al. 89). From the EDLC domain model, target system specifications can be generated. We are also developing a knowledge based tool called Knowledge Based Requirements Elicitation Tool (KBRET) that will tailor the domain model and generate target system specifications based on the requirements.

This paper is organized as follows: section 2 provides an overview of the EDLC methodology and the environment, section 3 briefly describes the target system specification generation process, section 4 describes the architecture/design of KBRET, section 5 details the implementation of KBRET in CLIPS and its capabilities, section 6 is summary, section 7 is acknowledgements and section 8 is references.

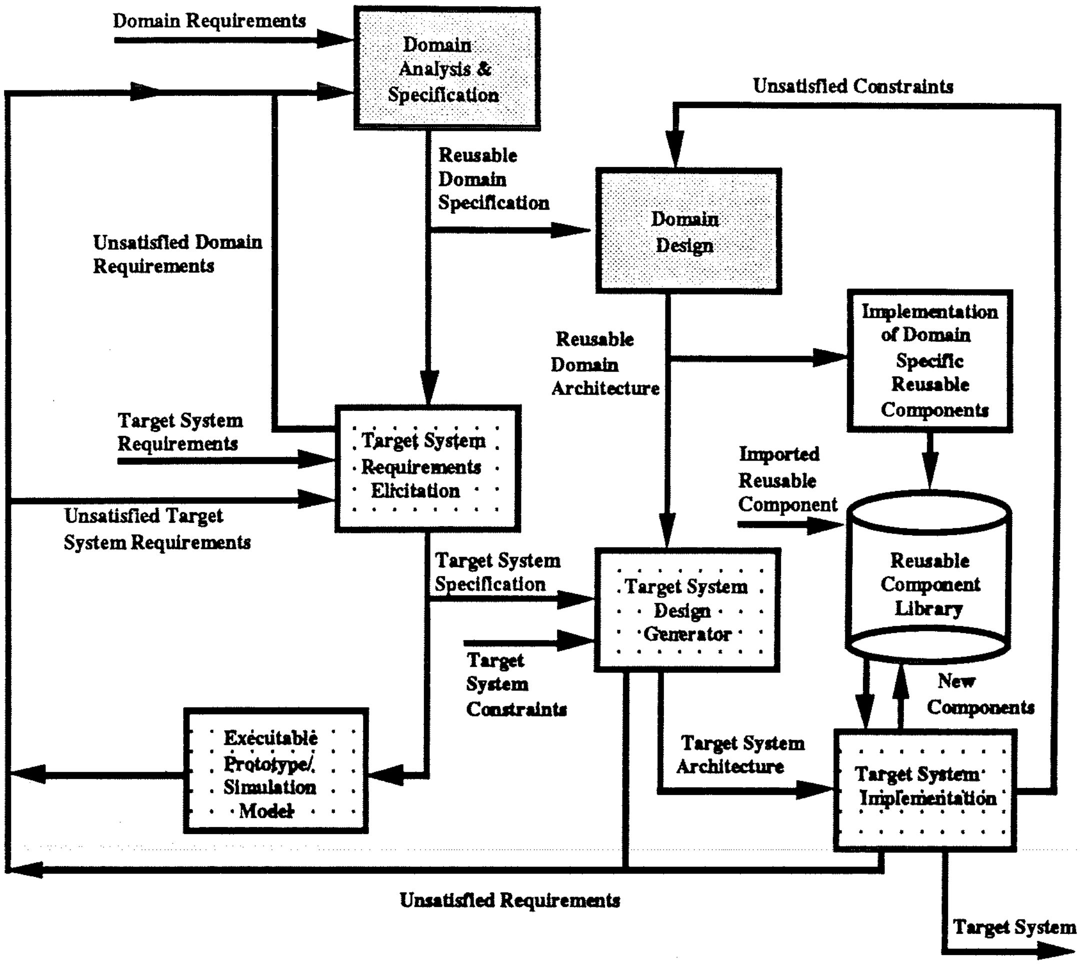
2.0 THE EVOLUTIONARY DOMAIN LIFE CYCLE MODEL

The Evolutionary Domain Life Cycle (EDLC) Model is a software life cycle model that eliminates the traditional distinction between software development and maintenance (Gomaa and Kerschberg 91a). The various activities within the EDLC paradigm are shown in Figure 1.

Software systems evolve through several iterations. Hence, systems developed using this approach need to be capable of adapting to changes in requirements during each iteration. Furthermore, because new software systems are often outgrowths of existing ones, the EDLC model takes an application domain perspective allowing the development of families of systems. A complete description of the EDLC methodology and related activities is provided in (Gomaa et al. 89). The EDLC domain model supports the following multiple views:

(a) *Aggregation Hierarchy.* The Aggregation Hierarchy is a composition hierarchy, i.e. it supports the IS-PART-OF relationship. It supports the decomposition of complex aggregate objects (subsystems) into less complex objects eventually leading to simple objects at the leaves of the hierarchy. Alternatively, associated objects may be grouped together into more complex aggregate objects. The Aggregation Hierarchy is an important abstraction concept in domain modeling, since it allows domain modelers to reason about complex aggregate objects instead of two or more simpler objects.

b) *Generalization/Specialization Hierarchies.* The Generalization/Specialization Hierarchy supports the IS-A relationship. With the generalization/specialization classification approach, similar objects are grouped into classes. These objects may have some features in common but they may also have variations between them that need to be expressed. Specialization of a class (object type) can be achieved by means of inheritance, which has been applied very effectively in object oriented programming. Meyer (Meyer 88) has convincingly shown the potential benefits of


Domain Requirements
Domain Analysis & Specification
Unsatisfied Constraints
Reusable Domain Specification
Domain Design
Unsatisfied Domain Requirements
Reusable Domain Architecture
Implementation of Domain Specific Reusable Components
Target System Requirements
Target System Requirements Elicitation
Imported Reusable Component
Unsatisfied Target System Requirements
Target System Specification
Target System Design Generator
Reusable Component Library
Target System Constraints
New Components
Executable Prototype/ Simulation Model
Target System Architecture
Target System Implementation
Unsatisfied Requirements
Target System

▨ **Domain Modeling**

⊡ **Target System Generation**

Figure 1. The Evolutionary Domain Life Cycle Model

using inheritance for reuse. Inheritance allows a class to be tailored by adding features, suppressing features or modifying features.

c) *Object Communication Diagrams*. Objects in the real world are modelled as concurrent processes (Jackson 83), which communicate with each other using messages. The object communication diagrams, which are hierarchically structured, show how objects communicate with each other.

d) *State Transition Diagrams*. As each active object is a sequential process, it may be defined by means of a finite state machine and documented using a state transition diagram (although in some cases the finite state machine might be trivial). Each object maintains its own state. An active object supports one operation for each message type that it may receive.

An example of applying the EDLC methodology to an Automobile Cruise Control Problem is given in (Gomaa 90). The domain modeling concepts are currently being applied to NASA's Payload Operations Control Center (POCC) domain at NASA Goddard Space Flight Center. To test these concepts, a domain model has been constructed which captures the similarities and variations of the POCC domain (Gomaa et al. 91b). The following section describes the proof-of-concept demonstration that we are developing for NASA/Goddard.

2.1 EDLC Domain Modeling Environment

A proof-of-concept experiment is under way to develop software tools that support the domain analysis and specification and target system requirements elicitation phases of the EDLC. The experiment uses comercial-of-the-shelf software as well as specially developed software. We are using Software Through Pictures (StP) to represent the multiple views of the domain model, although semantically interpreting the views according to the domain modeling method. The information in the multiple views is extracted, checked for consistency, and stored in an object repository.

A knowledge based tool is used to assist with target system requirements elicitation and generation of the target system specification. The tool, implemented in CLIPS, conducts a dialog with the human target system engineer, prompting the engineer for target system specific information. The output of this tool is used to adapt the domain specification to generate the target system specification.

The EDLC domain modeling environment is depicted in Figure 2. In the EDLC paradigm, the domain analyst starts the creation of the domain model by specifying the multiple views. He/she creates the Aggregation Hierarchy, Generalization/Specialization hierarchies, Object Communication Diagrams, and State Transition Diagrams. The domain analyst is not restricted to working with one diagram at a time. A complex system must be understood by a large community of users and these multiple views provide an informal specification of the domain being constructed. Software Through Pictures (StP) is used as the multiple viewpoint graphical editor. StP provides limited form of consistency checking within each view. Additional consistency checking among different views is done by special software that we have developed. Once the graphical views are determined to be consistent, the domain information from these graphical views is extracted from StP's relational database and mapped into the object repository.

The object repository presents an object-oriented representation of the informal specification which can now be enhanced with a formal specification of domain object types, their attributes, their relationship to other object types, the operations associated with the object types, constraints among object types etc. The domain analyst enters this information using the Domain Object Editor. The domain object types are organized in terms of a "meta-schema" that governs the inter-relationships among the objects represented in the multiple viewpoints. This knowledge is used to determine consistency among view points and to evolve the object specification in a consistent manner. The object repository is implemented in Eiffel.

The domain specification stored persistently in the object repository is augmented with domain features (requirements), inter-feature dependencies and feature/object dependencies. Inter-feature dependencies capture the relationships among features. For example, a feature may require the presence of some other feature(s) as prerequisite. Another example of inter-feature dependency

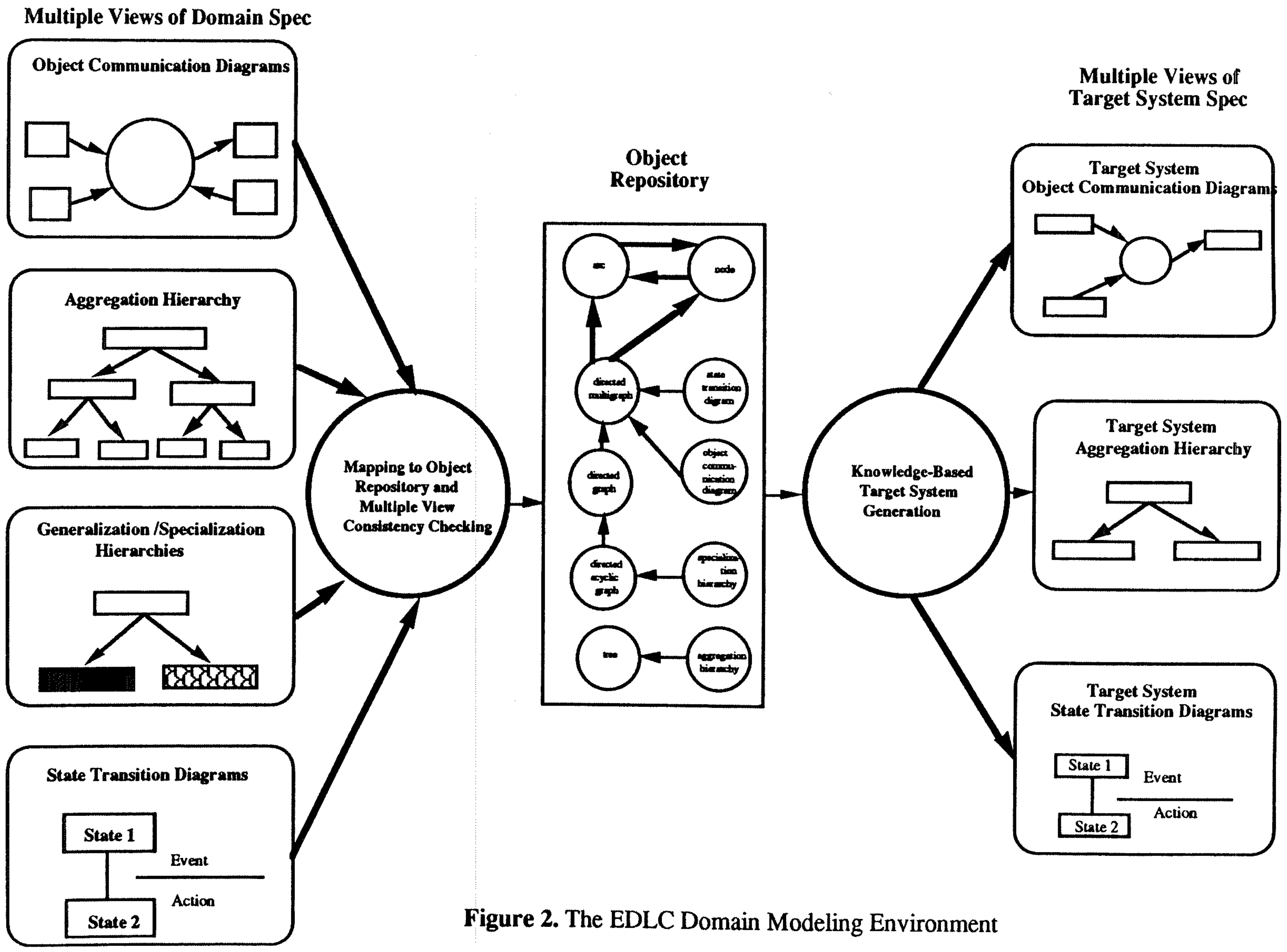

Multiple Views of Domain Spec
Object Communication Diagrams
Aggregation Hierarchy
Generalization /Specialization Hierarchies
State Transition Diagrams
State 1
State 2
Event
Action
Mapping to Object Repository and Multiple View Consistency Checking
Object Repository
arc
node
directed multigraph
state transition diagram
object communication diagram
directed graph
directed acyclic graph
specialization hierarchy
tree
aggregation hierarchy
Knowledge-Based Target System Generation
Multiple Views of Target System Spec
Target System Object Communication Diagrams
Target System Aggregation Hierarchy
Target System State Transition Diagrams
State 1
State 2
Event
Action

Figure 2. The EDLC Domain Modeling Environment

is that some features may be mutually exclusive or mutually inclusive with each other. The feature/object dependencies relate features to objects, i.e., they define the object types required to support a particular feature. The domain analyst provides this feature-related information using the Feature Object Editor and is stored in the object repository as well. The object repository interfaces with knowledge-based tools such as KBRET and provides the informal and formal specifications for reuse. Thus, the object repository provides a unique and consistent specification of the domain model, and this can be accessed by various tools. For example, we have developed a tool that retrieves the domain dependent information from the object repository and creates the domain dependent knowledge sources for KBRET. This tool maps the object repository information into CLIPS facts using the *deffacts* construct.

Once the domain modeling activity is completed, the domain specification serves as the framework for generating target systems. The process of generating target systems calls for knowledge-based tool support. This tool must not only have knowledge about the domain model, but also contain procedural knowledge about constructing target systems. A knowledge-based system called the Knowledge-Based Requirements Elicitation Tool (KBRET) is being developed using CLIPS to automate the process of generating the specifications for target systems. When a target system specification is generated, the corresponding multiple views are then generated by tailoring the domain multiple views and displayed using StP.

The paragraphs above have provided a brief overview of the EDLC methodology and the different tools that are used in creating the domain model and generating the target system specification. The following sections will describe the process of generating target system specification, the knowledge-based tool used in this process, its architecture and implementation.

3.0 TARGET SYSTEM SPECIFICATION GENERATION

The EDLC domain model captures the reusable domain features (requirements) and the dependencies among features and object types. These feature object dependencies provide a powerful indexing mechanism to retrieve reusable components from the domain model, especially object types and their informal and formal specifications. In other words, if a particular domain feature is required in the target system, the object types required to support that feature can be retrieved from the object repository. Thus, the process of generating a target system specification essentially amounts to gathering the requirements in terms of the domain features and retrieving from the domain model the corresponding components to support those features and reason about inter-feature and feature/object dependencies to ensure consistency. This process involves tailoring the domain model and creating the target system specification.

The object types in the EDLC paradigm are classified as kernel, optional, or variant. A kernel object is part of every member in the family of systems. An optional object supports a domain feature and it may or may not be included in the target system. A variant object is a specialization of a kernel or optional object and it also supports a certain domain feature. If multiple variants of the same object type are included in the target system, they have to be integrated to form one synthesized object. However, certain application domains may require the existence of multiple specializations of the same object type, as for example, in the POCC domain, multiple specialized experiments may co-exist on a mission.

Once the requirements have been gathered, the target system can be assembled by including the kernel object types, the selected optional object types and variant object types and integrating the variant object types, if necessary, with the help of the domain analyst. In general, the target system specification generation process consists of the following activities:

(a) accessing and retrieving necessary components from the various knowledge sources;
(b) displaying the information to the target system engineer;
(c) eliciting the target system requirements from the target system engineer;

(d) reason about the inter-feature and feature/object dependencies to ensure consistency;
(e) tailoring the domain specification to generate the target system (this step may or may not require variant integration); and
(f) target system consistency checking.

In order to support the above activities, the target system specification generation tool should be designed in such a way that the target system engineer could browse the domain model, interactively specify the requirements for the target system. The tool should also have the reasoning capability to ensure that a consistent specification for the target system has been generated. The following section describes the architecture of the Knowledge Based Requirements Elicitation Tool (KBRET).

4.0 KNOWLEDGE BASED REQUIREMENTS ELICITATION TOOL (KBRET)

KBRET accomplishes the task of generating the target system specification in several phases. Some of the phases that KBRET may go through are: Browsing, Target System Requirements Elicitation, Dependency Checking, Target System Generation, Variant Object Type Integration.

4.1 KBRET User Interface

The current version of KBRET uses a menu based approach for interacting with the target system engineer. From the main menu, the browsing phase or the target system requirements elicitation phase can be initiated. In the browsing phase, the target system engineer can browse the features captured in the domain model and also get explanations for those features. When the target system requirements elicitation phase is initiated, KBRET provides the domain features in a menu form and the target system engineer can select from this menu the features desired in the target system. Once the features required in the target system are selected, KBRET presents the selected features in a menu form and the target system engineer can delete some of the features selected for the target system.

Whenever a feature is selected or deleted, the dependency checking phase is invoked to ensure consistency. The target system engineer has the flexibility to select or delete features at any time during the target system requirements elicitation phase. When the requirements elicitation phase is exited, the target system construction phase can be initiated by selecting that option from the menu that KBRET provides. This menu also provides an option for specifying features that are not in the domain model. If this option is selected, KBRET suspends the session after gathering information about these new features. This information is provided to the domain analyst to enhance the domain model. When the new features are incorporated in the domain model, the target system engineer can continue the session and finish generating the target system specification. A sample session with KBRET is given in the Appendix.

To support this phased approach, KBRET utilizes various knowledge sources. These knowledge sources can be categorized as domain independent and domain dependent. This separation between the domain-independent and domain-dependent knowledge is essential for providing scale-up and maintainability of domain specifications for large domains. The various components, including the different knowledge sources of KBRET are schematically shown in Figure 3 and discussed in the following sections.

4.2 Domain Independent Knowledge Sources

The domain independent knowledge sources provide procedural and control knowledge for the various functions supported by KBRET. The *Dialog Manager* is responsible for carrying out a meaningful dialog with the target system engineer and elicit the requirements for the target system. It addresses such issues as how, and in what sequence the target system engineer should be

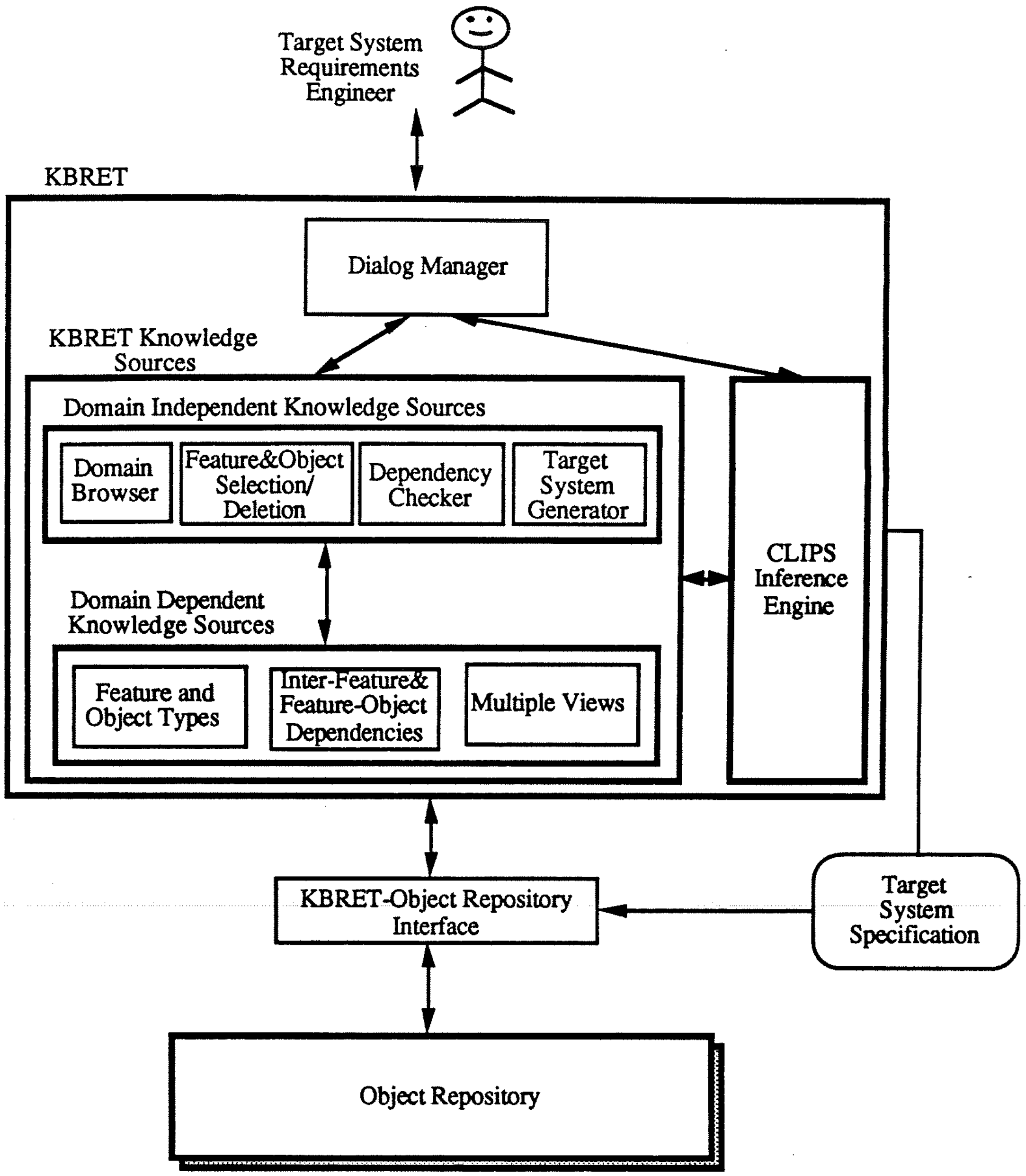

Target System Requirements Engineer
KBRET
Dialog Manager
KBRET Knowledge Sources
Domain Independent Knowledge Sources
Domain Browser
Feature&Object Selection/ Deletion
Dependency Checker
Target System Generator
Domain Dependent Knowledge Sources
Feature and Object Types
Inter-Feature& Feature-Object Dependencies
Multiple Views
CLIPS Inference Engine
KBRET-Object Repository Interface
Target System Specification
Object Repository

Figure 3. Knowledge Based Requirements Elicitation Tool (KBRET)

prompted for various features, invoking and controlling the different phases of KBRET, the user interface etc.

Before specifying the requirements for the target system, the target system engineer may wish to browse through portions of the domain model in order to gain understanding of the application domain under consideration. The *Domain Browser* knowledge source provides this facility. It provides rules for initiating and terminating the browsing facility and also the appropriate domain dependent knowledge sources to be accessed in order to facilitate the browsing of those parts of the domain model which the target system engineer wishes to explore.

The *Feature & Object Selection/Deletion* knowledge source keeps track of the selection or deletion of features for the target system and the corresponding object types. This knowledge source incorporates rules for selecting and deleting features and also invoking the appropriate rules for checking inter-feature and feature/object dependencies.

The *Dependency Checker* knowledge source cooperatively works with the Feature & Object Selection/Deletion knowledge source. When a particular feature is selected for the target system, the *Dependency Checker* enforces the inter-feature and feature/object dependencies for that feature. These dependencies are obtained from the *Inter-Feature & Feature-Object Dependencies* knowledge source which is domain dependent, as shown in Figure 3. When a feature with some prerequisite features is selected, the *Dependency Checker* ensures that those prerequisite features are included in the target system. For example, in the POCC domain, the Verifying Real Time Commands feature requires Sending Real Time Commands feature. If the Sending Real Time Commands feature is not selected and the Verifying Real Time Commands feature is desired in the target system, the Sending Real Time Commands feature will be included in the target system before selecting the Verifying Real Time Commands feature.

Similarly, before deleting a feature from the target system, dependency checking is performed to ensure that it is not required by any other target system feature. Using the example from the previous paragraph, if both Sending Real Time Commands and Verifying Real Time Commands features are selected for the target system, the Sending Real Time Commands feature cannot be deleted from the target system as long as the Verifying Real Time Commands feature is selected for the target system. Thus, the *Dependency Checker* knowledge source has rules to enforce the inter-feature and feature/object dependencies so that a consistent target system is specified.

Once the feature selection for the target system is complete, the *Target System Generator* knowledge source begins the process of assembling the target system. The domain kernel object types are automatically included in the target system. Depending upon the features selected for the target system, the corresponding variant and optional object types are included according to the feature/object dependencies. The *Target System Generator* would detect if more than one variant (specialization) of a particular kernel or optional object type were included in the target system. These multiple variant object types have to be "integrated" to produce one integrated variant object type that would support the desired features in the target system. Some domains may require the presence of multiple variants of certain objects and those variant objects should not be integrated. For example, in the POCC domain, multiple variants of observatory related objects should not be integrated.

If multiple specializations of a particular kernel or optional object have been selected and if they have to be integrated, the *Target System Generator* would access the *Multiple Views* domain dependent knowledge source and check the appropriate generalization/specialization hierarchy to see if an integrated object type for those variant object types exists as a result of previous variant integration processes. If such an integrated object type is present, then that object type is included in the target system in lieu of those variant object types to be integrated. Once all the required integrated variant object types have been included, the target system generation is complete.

If the integrated variant object type is not present in the domain model, the target system generation process is suspended and the target system engineer is notified about the need for variant integration and the variant object types to be integrated are presented to the domain analyst for integration and enhancing the domain model. Variant integration is a non trivial task and may require considerable domain knowledge. Hence, completely automating the variant integration process will be a tremendous challenge. When the integrated variant object is made available to the *Target System Generator*, the target system generation process is resumed.

4.3 Domain Dependent Knowledge Sources

As the name suggests, the domain dependent knowledge sources contain specific information about a particular application domain. They are used by the domain independent knowledge sources of KBRET in eliciting the requirements and generating the target system specification. The domain dependent knowledge sources are derived from the domain specification, which is persistently stored in the object repository. The KBRET-Object Repository Interface accesses the object repository and creates these knowledge sources using a representation that is compatible with the other knowledge sources of KBRET.

The *Features and Object Types* knowledge source contains a list of all the object types and features that have been incorporated in the domain model. For each object type, its name and properties are stored in this knowledge source. The properties of objects are: kernel, optional, variant, aggregate, agh_root and gsh_root. The CLIPS implementation of this knowledge source is essentially a list of facts - one fact for each object type and its properties and one fact for each feature. Some example facts from this knowledge source for the POCC domain are given below:

```
(Object: Command_Load_Processor kernel aggregate)
(Object: Observatory_Instrument_Telemetry_Equation_Processor optional gsh_root)
(Feature: Sending Real Time Commands)
```

The various relationships and dependencies among features and between features and object types are captured in the *Inter-Feature & Feature-Object Dependencies* knowledge source.The prerequisite relationship between two features is captured in a CLIPS fact with the key word "requires". For each feature, the object types required to support that feature are expressed as CLIPS facts using the key word "supported-by". These dependencies are enforced during feature selection or deletion by the *Dependency Checker* knowledge source. A few example dependencies from the POCC domain are given below:

```
(Verifying Real Time Commands requires Sending Real Time Commands)
(Verifying Real Time Commands supported-by Earth_Bound_Real-Time_Command Verifier)
```

The *Multiple Views* knowledge source contains the different views created using the EDLC methodology, in particular, the aggregation hierarchy and the generalization/specialization hierarchies. These hierarchies are accessed and utilized by the *Target System Generator* knowledge source when the target system is being assembled. The parent-child relationship between objects in the aggregation hierarchy is expressed as CLIPS facts using the key word "is-part-of". The supertype-subtype relation between objects in the generalization/specialization hierarchy is expressed as CLIPS facts with "is-a" key word. Sample CLIPS facts from this knowledge source are given below:

```
(is-part-of Real-Time_Command_Processor Satellite_Bound_Real-Time_Command_Processor)
(is-a POCC_Mode_Selector POCC_Mode_Selector_With_Simulation)
```

5.0 KBRET IMPLEMENTATION IN CLIPS

A prototype of the Knowledge Based Requirements Elicitation Tool has been developed using CLIPS which is an expert system shell developed by the Artificial Intelligence Section of the Mission Planning and Analysis Division at NASA/Johnson Space Center. CLIPS is written in 'C' and supports backward and forward-chaining, Rete algorithm for pattern matching, wildcards and single and multifield variables, externally defined functions in C or Ada or Fortran. Also, CLIPS can be embedded in application programs written in C, Ada or Fortran.

The basic elements of CLIPS are: fact-base, knowledge-base and inference engine (Giarratano 91). Facts are the basic form of information in a CLIPS system. Rules are fired based on the existence or non existence of facts in the fact-base. A fact is constructed of several positional fields separated by spaces or, a word followed by slots separated by parentheses. Facts may be asserted into the fact-base prior to starting execution and may be added or removed as the action of a rule firing.

A CLIPS knowledge base is represented in the form of production rules. The left hand side (LHS) of a rule is a series of relation or patterns which represent the conditions that must be satisfied for the rule to be fired. Logical operators may be used in the LHS for constraining the patterns. The right hand side (RHS) of the rule is the action to be performed as a result of the rule firing.

The inference engine examines the knowledge base to see if any rule's conditions have been met by searching the fact-base. All rules whose conditions are met are activated and placed in the agenda. The sequence in which these rules are executed is determined by the priorities assigned to those rules. The top rule in the agenda is selected and its RHS actions are executed. As a result of RHS actions, new rules may be activated or deactivated. This cycle is repeated until all rules that can fire have done so or until the rule limit is reached. The number of rule firings allowed in a cycle may be set apriori.

The domain independent knowledge sources of KBRET are implemented as CLIPS rules. The domain specific information contained in the domain dependent knowledge sources are expressed as CLIPS facts. These facts are asserted into the fact-base before the requirements elicitation process begins. When KBRET begins execution, the *Dialog Manager* initiates the dialog with the target system engineer. The rules in this knowledge source are written in such a way that the course of the dialog and the invocation of the different phases are determined by the target system requirements engineer's responses.

At the start of the dialog, KBRET prints the system banner and prompts the target system engineer if he/she wishes to browse the domain model or would like to specify the requirements for the target system, as shown in the sample dialog in the Appendix. If the response is to browse, the browsing phase is initiated. The target system engineer can explore the domain model and get explanations for the different features incorporated in the domain model. Once sufficient familiarity with the domain model has been gained, the target system requirements specification phase may be initiated.

The target system engineer is presented with the various features captured in the domain model in the form of a menu, as shown in the Appendix, and the features desired in the target system can be selected from this menu. Whenever a feature is selected for the target system, the dependency checking phase is initiated and the inter-feature and feature/object dependencies are checked and enforced. If a particular feature, say "F", requires the presence of other features, and they are not selected for the target system, the target system engineer is informed of that fact and those features are automatically included in the target system in order to support feature "F".

An example of this feature dependency checking is shown in the sample dialog in the Appendix. When the target system engineer tries to select the Verifying Real Time Commands (feature 7), KBRET comes back with a message saying that Verifying Real Time Commands requires Sending Real Time Commands feature and it will be automatically included in the target system, and requests the target system engineer's confirmation. When the target system engineer types "y" to confirm the selection, KBRET includes both the Sending Real Time Commands feature and the Verifying Real Time Commands feature in the target system and displays a message to that effect, as shown in the Appendix.

When a feature is selected for the target system, the object types that are required to support that feature are also selected in accordance with the feature/object dependencies and the CLIPS fact-base is updated to reflect that fact. The target system engineer thus, can specify the requirements for the target system and the *Feature & Object Selection/Deletion* knowledge source asserts new facts into the fact-base to record those selections. Of course, the *Dependency Checker* would ensure that the inter-feature and feature/object dependencies have not been violated.

The target system engineer can also delete features that have been selected for the target system. If a feature, say "F", is to be deleted, the *Dependency Checker* will check the fact-base to see if any of the features selected for the target system require that feature "F". If so, the deletion of feature "F" is disabled. An example this deletion dependency checking is shown in the sample dialog. When the target system engineer tries to delete the Sending Real Time Commands (feature 6) from the target system, KBRET comes back with a message saying that the Sending Real Time Commands feature is required by the Verifying Real Time Commands feature and since Verifying Real Time Commands feature is currently selected for the target system, the Sending Real Time Commands feature cannot be deleted, and the dialog continues. When a feature "F" is deleted, it may cause the deletion of some other features if those features were included in the target system solely because of the selection of feature "F" and if they are not required by any other feature selected for the target system. The deletion of a feature also triggers the deletion of object types that were included to support that feature.

If the target system engineer would like to specify a feature that has not been captured in the domain model, the requirements elicitation phase is suspended and the domain analyst is called upon to model that requirement and enhance the domain model. Then, the target system specification and generation may be resumed.

Once the requirements for the target system have been completely specified, the target system generation phase is invoked. KBRET prompts for a name for the target system that is being generated so that it could be stored in the object repository for reuse. The fact-base is examined and the features and the object types selected for the target system are gathered. KBRET then presents the list of features that have been selected for the target system. The kernel object types are included in the target system because they must be part of every member of the family of systems. The selected variant and optional object types are examined to see if variant integration is required. If variant integration is not required, then the target system specification is generated and presented to the target system engineer.

In presenting the target system specification, KBRET provides two options. The target system engineer may view only the leaf level object types or he/she can view both the aggregate and leaf level object types. If the target system engineer chooses the second option, KBRET provides the aggregation hierarchy for the target system, as shown in the Appendix. This is accomplished by pruning the domain aggregation hierarchy, i.e., deleting from the domain aggregation hierarchy the object types that have not been included in the target system. KBRET presents the target system aggregation hierarchy in an indented form, as shown in the Appendix, to reflect the various levels of the aggregation hierarchy.

If variant integration is required, the domain analyst is called upon to perform variant integration. When the integration process is completed, the target system generation phase is resumed and the target system specification is generated and presented to the target system engineer.

6.0 SUMMARY

Domain modeling field is rapidly growing and early results show that domain modeling effectively addresses some of the problems in reuse. We have given an overview of the Evolutionary Domain Life Cycle (EDLC) model and the activities within this paradigm. A domain model of the NASA/Goddard Payload Operations Control Center (POCC) domain is being developed as a proof-of-concept of our EDLC methodology. From the EDLC domain model, several target system specifications can be generated. We have discussed the target system generation process as well as the tools used in accomplishing this task. The architecture of the Knowledge Based Requirements Elicitation Tool (KBRET) and its implementation in CLIPS is also presented. KBRET interacts with the target system engineer, elicits the requirements and generates the target system specification by tailoring the domain specification.

7.0 ACKNOWLEDGEMENTS

We gratefully acknowledge the assistance of S. Bailin, R. Dutilly, J.M. Moore, and W. Truszkowski in providing us with information on the POCC. This work was sponsored by NASA Goddard Space Flight Center with assistance from Virginia Center of Innovative Technology. The Software Through Pictures CASE tool was donated to GMU by Interactive Development Environments.

8.0 REFERENCES

Biggerstaff, T.J, Perlis, A. J., (ed) (1989) *Software Reusability Concepts and Models, Volume I and II*, ACM Press Frontier Series, Addison Wesley.

Giarratano, J. C. (1991). Clips User's Guide, Version 5.0, Software Technology Branch, Lyndon B. Johnson Space Center.

Gomaa, H. (1990). A domain analysis, specification and design method for concurrent systems, George Mason University Report, September.

Gomaa, H., Fairly, R., Kerschberg, L., Sugumaran, V., O'Hara-Schettino, E., and Tavakoli, I., (1989). Sustaining engineering: life cycle support for evolutionary software development, *Research Report Prepared for NASA Goddard Space Flight Center.*

Gomaa, H., Kerschberg, L. (1991a). An evolutionary domain life cycle for domain modeling and target system generation, *Proc. of Domain Modeling Workshop*, May 13, pp. 65-71.

Gomaa, H., Kerschberg, L., Sugumaran, V., O'Hara-Schettino, E., and Tavakoli, I., (1991b). Revised domain model for the payload operations control center (pocc) domain, *Research Report Prepared for NASA Goddard Space Flight Center.*

Jackson, M. (1983). *System Development*, Prentice Hall.

Meyer, B. (1988). *Object-Oriented Software Construction*, Prentice Hall.

Parnas, D. (1979). Designing software for ease of extension and contraction, *IEEE Transactions on Software Engineering*, Vol. 5 No. 2, pp. 128-137.

Appendix. Sample Dialog with KBRET for the POCC domain.

```
***********************************************************
*                                                         *
*  KNOWLEDGE BASED REQUIREMENTS ELICITATION TOOL          *
*                       (KBRET)                           *
*                                                         *
***********************************************************

 Requirements Elicitation for POCC domain
 ***********************************
You may browse the features incorporated in the Domain Model, specify the requirements for the
Target System or quit KBRET.
 Choices                  Perform
 *******                  *******
    1             Browse the Domain Model
    2             Specify requirements for Target System
    3             Quit KBRET
Please type your choice and hit return: 1

        Domain Model Browsing Phase
        **************************
Please select one of the following choices to continue.
 Choices                  Perform
 *******                  *******
    1                 Explore the Features
    2                 Exit Browsing Phase
    3                 Quit KBRET
Please type your selection and hit return: 1

        Feature Exploration
        ****************
For the description of a feature, please type its number.
        Choices              Feature to be described
        *******              *****************
           1              Mission Type One
           2              Mission Type Two
           3              Experiment Type One
           4              Experiment Type Two
           5              Data Collection of Simulated Telemetry
           6              Sending Real Time Commands
           7              Verifying Real Time Commands
           e              Exit Browsing Phase
Please type your selection and hit return: 5

Data Collection of Simulated Telemetry:
**********************************
Simulated Telemetry Data can be collected and analyzed.

        Choices              Feature to be described
        *******              *****************
           1              Mission Type One
           2              Mission Type Two
           3              Experiment Type One
           4              Experiment Type Two
           5              Data Collection of Simulated Telemetry
           6              Sending Real Time Commands
           7              Verifying Real Time Commands
```

```
                e               Exit Browsing Phase
Please type your selection and hit return: e

Exiting the Browsing Phase........

You may browse the features incorporated in the Domain Model, or specify the requirements for the
Target System or quit KBRET.
          Choices              Perform
          *******              *******
             1              Browse the Domain Model
             2              Specify requirements for Target System
             3              Quit KBRET
Please type your choice and hit return: 2

                         Target System Requirements Elicitation Phase
                         ***************************************
Now, you will be presented with the features incorporated in the Domain Model.  If a feature is desired in the target
system, please type its number and hit return.  Please select one of the following choices to continue.
          Choices              Feature to be selected
          *******              *****************
             1              Mission Type One
             2              Mission Type Two
             3              Experiment Type One
             4              Experiment Type Two
             5              Data Collection of Simulated Telemetry
             6              Sending Real Time Commands
             7              Verifying Real Time Commands
             e              End selecting features
Please type your selection and hit return:  2
                         The Mission Type Two Feature has been selected for the Target System.

Please select one of the following choices to continue.
          Choices              Feature to be selected
          *******              *****************
             3              Experiment Type One
             4              Experiment Type Two
             5              Data Collection of Simulated Telemetry
             6              Sending Real Time Commands
             7              Verifying Real Time Commands
             e              End selecting features
Please type your selection and hit return:  3
                         The Experiment Type One Feature has been selected for the Target System.

Please select one of the following choices to continue.
          Choices              Feature to be selected
          *******              *****************
             4              Experiment Type Two
             5              Data Collection of Simulated Telemetry
             6              Sending Real Time Commands
             7              Verifying Real Time Commands
             e              End selecting features
Please type your selection and hit return:  4
                         The Experiment Type Two Feature has been selected for the Target System.

Please select one of the following choices to continue.
          Choices              Feature to be selected
          *******              *****************
             5              Data Collection of Simulated Telemetry
```

```
            6               Sending Real Time Commands
            7               Verifying Real Time Commands
            e               End selecting features
Please type your selection and hit return:  7
                    The Verifying Real Time Commands Feature requires Sending Real Time Commands Feature.
                    The Sending Real Time Commands Feature will be automatically included if the Verifying Real
                    Time Commands Feature is desired.

Please type 'y' to include or 'n' to not include the Sending Real Time Commands Feature. (y/n): y
        The Sending Real Time Commands Feature has been selected for the Target System.
        The Verifying Real Time Commands Feature has been selected for the Target System.

Please select one of the following choices to continue.
        Choices             Feature to be selected
        *******             *****************
            5               Data Collection of Simulated Telemetry
            e               End selecting features
Please type your selection and hit return:  e

Target System feature selection has been exited...

The following features are currently selected:
***********************************
        Number              Feature Name
        *******             ***********
            2               Mission Type Two
            3               Experiment Type One
            4               Experiment Type Two
            6               Sending Real Time Commands
            7               Verifying Real Time Commands

Please select one of the following choices to continue.
        Choices                 Perform
        *******                 *******
            1               Select more features for Target System
            2               Delete a feature from Target System
            3               Specify features not in the Domain Model
            4               Initiate Target System Generation Phase
            5               Quit KBRET
Please type your selection and hit return:  2

Please select one of the following choices to continue.
        Choices             Feature to be deleted
        *******             *****************
            2               Mission Type Two
            3               Experiment Type One
            4               Experiment Type Two
            6               Sending Real Time Commands
            7               Verifying Real Time Commands
            e               End deleting features
Please type your selection and hit return:  6

        Since the Sending Real Time Commands Feature is required by the Verifying Real Time Commands
        Feature and since the Verifying Real Time Commands Feature is currently desired in the Target System, the
        Sending Real Time Commands Feature may not be deleted now.
Please type (c) and hit return to continue: c

Please select one of the following choices to continue.
```

```
          Choices              Feature to be deleted
          *******              *****************
            2                 Mission Type Two
            3                 Experiment Type One
            4                 Experiment Type Two
            6                 Sending Real Time Commands
            7                 Verifying Real Time Commands
            e                 End deleting features
Please type your selection and hit return:  3
          Since Experiment Type One Feature is not required by any other target system feature, it will be deleted
          from the Target System Features.
Please type 'y' to delete or 'n' to abort the deletion of Experiment Type One Feature (y/n) : y
          The Experiment Type One Feature has been deleted from the Target System.

Please select one of the following choices to continue.
          Choices              Feature to be deleted
          *******              *****************
            2                 Mission Type Two
            4                 Experiment Type Two
            6                 Sending Real Time Commands
            7                 Verifying Real Time Commands
            e                 End deleting features
Please type your selection and hit return:  e

Target System feature deletion has been exited...

The following features are currently selected:
***********************************
          Number               Feature Name
          *******              ***********
            2                 Mission Type Two
            4                 Experiment Type Two
            6                 Sending Real Time Commands
            7                 Verifying Real Time Commands

Please select one of the following choices to continue.
          Choices                   Perform
          *******                   *******
            1                 Select more features for Target System
            2                 Delete a feature from Target System
            3                 Specify features not in the Domain Model
            4                 Initiate Target System Generation Phase
            5                 Quit KBRET
Please type your selection and hit return:  4
          Invoking the Target System Generation Phase.....

                    Target System Generation Phase:
                    ***************************
Please input a name for the Target System: EXAMPLE

                              EXAMPLE Target System Components
                              *********************************
          The following features have been selected for the EXAMPLE Target System
          **************************************************************
                              Mission Type Two Feature
                              Experiment Type Two Feature
                              Sending Real Time Commands Feature
                              Verifying Real Time Commands Feature
```

```
              Assembling the EXAMPLE Target System. Please Wait........

The Target System Object Types have been assembled.  To view those object types included in the Target System,
Please select one of the following choices:
      Choices                  Perform
      *******                  *******
         1             View Leaf Level Object Types
         2             View Aggregate and Leaf Level Object Types
Please type your selection and hit return: 2

The Aggregate and Leaf Level Objects of the Target System:
*************************************************
 Payload Operations Control Center Domain (kernel aggregate)
      Telemetry (kernel aggregate)
            Telemetry Pre-Processor (kernel)
            Spacecraft Telemetry Processor (kernel aggregate)
                  Mission Two SC Eng. Telemetry Analog Limits Checker With Eqn. Processing (variant)
                  Mission Two SC Engineering Telemetry Trend Analyzer (variant)
                  Mission Two SC Engineering Telemetry Equation Processor (variant)
                  Mission Two Discrete SC Engineering Telemetry Analyzer (variant)
                  Mission Two FDF Interface (variant)
            Observatory Telemetry Processor (kernel aggregate)
                  Experiment Two Instrument Telemetry Analog Limits Checker (variant)
                  Experiment Two Instrument Telemetry Trend Analyzer (variant)
                  Experiment Two Discrete Instrument Telemetry Analyzer (variant)
                  Experiment Two Scientific Telemetry Analyzer (variant)
            TAC Interface (kernel)
            RUPS Interface (kernel)
      Command (kernel aggregate)
            Command Load Processor (kernel aggregate)
                  Satellite Bound Command Load Processor (kernel)
                  Earth Bound Command Load Verifier (kernel)
                  Command Load Data Store (kernel)
                  OBC Image Verifier (kernel)
                  CMS Interface (kernel)
            Real Time Command Processor (optional aggregate)
                  Satellite Bound Real-Time Command Processor (optional)
                  Earth Bound Real-Time Command Verifier (optional)
                  Real-Time Command Data Store (optional)
            Satellite Bound Command Problem Resolver (optional)
      Flight Operations Analyst (kernel aggregate)
            FOA STOL Interface (kernel)
            POCC Mode Selector (kernel)
            FOA Command Processor (kernel)
            FOA NCC Processor (kernel)
            FOA Telemetry Processor (kernel)
            NCC Interface (kernel)
      History (kernel aggregate)
            Telemetry History (kernel)
            Command History (kernel)
            Flight Operations Analyst History (kernel)
            Telemetry Block History (kernel)

The EXAMPLE Target System Generation is complete.  The object types shown above have been included in it and
no variant integration is required.
```

N92-16589

The Management and Security Expert (MASE)

Mark D. Miller, Stanley J. Barr, Coranth D. Gryphon, Jeff Keegan, Catherine A. Kniker, Patrick D. Krolak[1]

University of Massachusetts at Lowell
Center for Productivity Enhancement
One University Ave
Lowell, Massachusetts 01854

Abstract

Today's computing environments are increasingly complex: they often consist of large networks of computers that include multiple vendors and operating systems. The various systems and the communications between them must be kept running at their peak performance levels to meet the demands of the user community. They must also be kept safe from malevolent intruders and damaging viruses. The challenge facing today's systems manager is an enormous one. Unfortunately, the conventional tools provided by manufacturers provide only minimal assistance, leaving the system manager with the bulk of the work.

The Management And Security Expert (MASE) can help. MASE is a distributed expert system that monitors the operating systems and applications of a network. It is capable of gleaning the information provided by the different operating systems in order to optimize hardware and software performance; recognize potential hardware and/or software failure, and either repair the problem before it becomes an emergency, or notify the systems manager of the problem; and monitor applications and known security holes for indications of an intruder or virus. MASE can eradicate much of the guess work of system management.

Introduction

The Management And Security Expert is a distributed system capable of monitoring operating system resources and network statistics across an entire network. MASE will consist of customizable expert system modules running on each host in the network, along with the addition of expert systems that provide special functions such as network security, network performance, and network audit services.

[1]Mark D. Miller, Project Manager; Stanley J. Barr, Research Assistant; Coranth D. Gryphon, Research Assistant; Jeff Keegan, Research Assistant; Catherine A. Kniker, Research Assistant; Patrick D. Krolak, Center Director

MASE is implemented using PCLIPS, or Parallel CLIPS[1], which comprises a set of extensions to NASA's CLIPS language. PCLIPS was designed by the ULowell's Center for Productivity Enhancement (CPE) for rapid prototyping of distributed applications, and has since expanded to include dynamic construct creation, archival facilities, truth maintenance, and alarm-timing mechanisms for cycling execution of constructs, and timed expiration of facts and instances.

PCLIPS is set up in a Client/Server model. Under the current model, each PCLIPS client has it's own PServer. When one expert system communicates with another, it does so by calling the *send-message* function, which connects with the PServer, and transmits a *send* message, which includes the outgoing fact. The PServer then in turn sends a PFACT out to the server of the other PCLIPS client.

Host Level Management and Security

MASE utilizies a series of diagnostic operations, or functions, written to monitor and check various system parameters and resources. At this point in the development of MASE, we have developed four types of diagnostic operations: the *check, get, action,* and*display* operations.

The *check* operations are used for checking a system resource against either a system manager defined threshold, or a previously saved system state. As the system manager configures each expert system, s/he sets the frequency in which the *check* operations are run. (Ex. 30 seconds, 6 Hours, 1 Week, 1 Month) Examples of *check* operations:

check-disk-flood, check-boottime, check-sys-clock, check-rhosts

The *get* operations are used for retrieving system values and settings. They are used when either the system manager or the CLIPS expert system is trying to solve a problem. Examples of *get* operations:

get-load, get-boottime, get-users, etc.

The *action* operations are used for fixing situations. They can be initiated by the system manager, or by the expert system. Examples of the *action* operations:

action-move-file, action-disuser-user, action-compress-file, etc.

The *display* operations are used for displaying system information in a form other than facts. This is typically done using a graphic interface. This allows the system manager to remotely monitor a situation, and take appropriate action if necessary. Examples of the *display* operations:

display-cpu-usage, display-disk-usage, display-queue

Depending upon system configuration, these operations can be performed as run-time functions, or they can be run as spawned (standalone) processes. When these operations are performed by MASE, they generate facts, which are used to drive the controlling expert system. If a particular operation is run as a run-time callable function, then the *assert* function is used to

[1]"PCLIPS: Parallel Clips", Coranth Gryphon, Mark Miller, Second Annual Clips Users Conference

add the fact to the PCLIPS fact base. When the operation is run as a spawned function, it connects to the PServer of the parent PCLIPS client, and sends a local fact (LFACT). The PServer then passes the fact along to the client.

The facts that are generated by the diagnostic operations have the following form:

(RESULT <result> <type> <func-name> <tag> 'descriptive message')

The **result** field has one of five values:

success -	This fact type is asserted when the operation completes successfully, and does not find any problems. (Used in Check operations)
warning -	This fact is asserted when the operation finds a problem with a system resource. (Used in Check operations)
error -	This fact type is asserted when the operation cannot successfully perform the task it has been assigned. (Ex. Missing configuration file - Used in all four types of operations)
outcome -	This fact type is asserted when the operation has accomplished some task. (Used in Action operations)
information -	This fact type is asserted when the operation has obtained some information. (Used in Get operations)

The **type** field has one of several values:

disk, memory, cd-rom, op-disk, network, process, op-sys, user, files, security, peripheral, zones, server, misc, unknown, "some string" - reserved for later use.

The **func-name** field is the name of the function that generated the fact.

The **tag** field is used for distinguishing between facts created by different calls to the same function. For instance, if the function is called twice, with two separate sets of parameters, the tag field allows PCLIPS to distinguish between the two calls.

The **"descriptive message"** is the rest of the fact that has the actual result information in it. For example, a warning fact generated by the check-disk-flood operation would look like this:

(RESULT warning disk check-disk-flood DEVICE "/dev/rza1" at SETTING 98 exceeding THRESH 95)

In the **"descriptive message"**, the capitalized words (DEVICE, SETTING, THRESH) are used as tags, signaling PCLIPS that the next field is information that should be extracted.

Configuring the host management expert system

Since the system manager configures each expert system, the reactions of the expert system can be tailored to the specific needs of the manager. For example, take the situation where the expert system is monitoring the amount of space available on a disk. Disks, and their associated storage space are critical resources to a running system. When a disk hits 100% capacity, users can no longer write to it. This is especially critical to users who are in editors, or are running applications that generate information that needs to be stored. During normal operational hours, the situation may not become critical, because the system manager would be able respond. However, at 3 a.m., there may not be any personnel around who can fix the problem. Because of this possibility, the following scenario typifies how MASE might be configured to solve a disk flooding problem on a Unix system. (See Fig. 1 for the user interface that the system manager would use)

Initially, the function *check-disk-flood* would be set up with a frequency of once every 30 minutes. The threshold passed to the *check* function for checking disk space on a partition might be 90%. When this threshold is exceeded, the expert system notifies the system manager of the situation, via a fact to the system manager's expert system controlled user interface. The expert system then modifies the threshold to 95%, and the frequency to 15 minutes.. If the threshold is exceeded again, the system manager is appraised of the new circumstances. Since the disk is approaching a completely flooded state, the expert system checks the time, which is 8:37 A.M. This time falls outside the range of normal system manager hours, so the expert system checks to see if the system manager is logged on and active (This can be done by querying other expert systems on the network). If the system manager is not available, the expert system takes an active role in resolving the situation. It performs the following operations:

1) Notify all the users that there is a problem, and ask them to take action themselves. (Clean up directories, compress files, etc.)
1) Check the temp directories on the disk (*check-tmp-dirs).* Purge any files older than 2 days that are in these directories.
2) Check for core files on the disk (*check-for-cores)* that are more than 1 week old. If any are found, compress them.
3) Check for any tar files on the disk (*check-for-tars),* and compress them.
4) Get a new reading of the disk space (*get-disk-space).*
5) If the percentage is below 90%, the problem has been corrected, so reset the values in the *check-disk-flood* function.
6) If the percentage is between 90% and 95%, set the parameters lower on the problem solving functions, such as using 1 day as a parameter for *check-tmp-dirs.*
7) If the percentage is higher than 95%, drastic actions may be required

a) Delete core files (*check-for-cores)* which generates a fact that contains the pathname of each core file found. These facts will in turn cause the rule that calls the function *action-delete-file* to fire.
b) Check the file system for a disk with more space.(*check-fs-usage*) If one is found, it may become necessary to move files over to it. Create a storage directory (*action-create-directory*) and move files over to it, starting with tar files. Any time a file is moved, the owner of the file must be notified of the move. (*action-notify-user*)

Notifying users becomes another issue. If mail is used, this may compound the flooding disk problem. So, the expert system might check to see if the user is logged on. If so, a message is written to the user's screen. If the user is not available, one option might be to insert a line in the .login of the user. Another option might be to keep the name of the user in the factbase, and start a check function that periodically checks to see if the user is logged in. When the user does log in, write to the screen of the user, and remove the fact from the fact base.

Currently, *action-notify-user* is set up to only to write to the screen of the user. However, that function could realistically be converted to a mini-expert system, which solves the problem of notifying the user without using mail.

The critical element of this system is that it is completely system manager configurable at startup, and modified dynamically at run-time. These operations are policy decisions, which are made on a per-site basis.

Configuring the host security expert system

Since security is vitally important on some installations, and not all that important on others, the system manager can set the level of security for the MASE system, on each node. That way it is possible to have a higher level of security on critical systems. The trade off is higher security requires higher resource utilization for MASE (more CPU time, more memory, etc.). At this point, we have three security stages defined. Stage 1 is the highest, Stage 2 is a medium level of security, and Stage 3 is a low level of security. This will change in the future, as we model national guidelines for computer security.[1][2] Here are some examples of the security levels for a BSD Unix system.

Stage 1 (High) -	Inter-MASE messages would all be encrypted. No device, including the console, would be set to *secure*. That way, no user could log onto the system directly as root. Another account would have to be used first, followed by an *su*.
Stage 2 (Medium) -	Some inter-MASE messages would be encrypted, such as registration messages. Only the console could be set to *secure*, but the system manager could override that.
Stage 3 (Low) -	No inter-MASE messages would be encrypted. (Fastest system) The system manager can set any device to *secure*.

There are two areas of interest in security at the host level. The first area is *Hole Detection*. MASE can be programmed to check for holes in the operating system, such as a world writeable password file, or world readable /dev/kmem. Since MASE is expert system driven, it is easy to add the ability to check for a new hole, should a new one arise. If a hole is detected, MASE will, depending upon it's configuration, either close the hole, or simply notify the manager. It checks for weaknesses that have been created by users, intentionally, or not. For instance, on some versions of the Unix operating system, there is a file that can be created by a user, called the .rhost file. In it, the user can list what machines and accounts can log onto the user's account, without the use of a password. In this file, the user is allowed to use a wildcard, which creates a tremendous security hole. When a wildcard is specified in the username field, it creates a hole that allows any user on the specified machine to log in on the account, without specifying a password.
Examples of some of the hole detection check functions are:

check-os-files-protection - This checks a list of os files that need specific protection settings.
check-suid-files - This checks for files with the suid bit set.
check-passwd-file - This checks for exploitable accounts (no passwords, unauthorized root uid

[1]"*A Guide to Understanding Configuration Management in Trusted Systems*", National Computer Security Center, March 1988
[2] "*A Guide to Understanding Trusted Facility Management*", National Computer Security Center, October 1989

owners, etc.)

The second area of security at the host level is *intruder detection*, which is in the preliminary stages of development. This requires MASE to monitor the areas of the operating system that are prone to attack, such as password guessing. The following scenario shows how MASE could be configured to handle a password guessing attack.

There is an attempt at logging onto the system. A valid username is provided, but the password does not match. MASE detects the failure. If a second attempt is made, MASE then attempts to determine where the login request is coming from. This information could be obtained through *intelligent* utilities at the operating system level, or network packet analysis. If three attempts fail (manager definable amount), MASE could then start putting up barriers.

1) If the attempt is being made from outside the manager's domain, it is possible, using an intelligent network router, to disable incoming traffic from the offending system, or domain. This is fairly drastic, but if the security of the system is deemed important enough by the system manager, then it is a useful response.

2) Temporarily disuser the account, until the real owner of the account identifies him or herself.

One of the advantages of MASE reacting to an intruder threat, instead of simply notifying the system manager, is that valuable time could be lost waiting for the system manager to react.

Configuring the network management expert system

At the network level , MASE can use both passive and active actions to diagnose network related problems. In order for the system manager to understand the layout of the network, it is important to generate a topological map of the network, which can be displayed for the system manager. The topological map is created through the use of several different types of network functions.

1) Pinging: This is an active function that uses network software to determine the existence of an active node on the network. Using a list of expected nodes on the network, (/etc/hosts on Unix systems) the pinging function will actively poll the various nodes on the network. If they are alive, they will respond back to the ping. This gives MASE a initial picture of what nodes are active on the network.

2) Routing Tracing: This is a second active function that allows MASE to trace what routing paths would be used between two specified machines. This enables MASE to get a clearer picture of the network topology. Using this technique, which uses UDP packets, MASE can determine where gateways and routers are located, address-wise.

3) Ethernet Monitoring: Turning a Unix machine into promiscuous mode allows MASE to monitor packets traversing the network. Using a filtering program, statistical information based upon traffic patterns can be gathered. This can be used for simple informational purposes, and it can also be used to diagnosing network problems, such as saturated subnets, malfunctioning ethernet cards, run-away daemons, etc.

4) SNMP Capabilities: The Simple Network Management Protocol (SNMP) has become the de facto standard for a network management protocol. At this point, most hardware vendors have either delivered, or promised to deliver SNMP support with their network hardware. SNMP support will allow MASE to gather network information from network machines, without having

to run MASE on each of the machines. This becomes even more important for network routers. Routers are not multi-function machines. They are designed with one purpose in mind, which is routing of network traffic. Instead of trying to get a MASE agent running on different routers, MASE will use the SNMP protocol to get the same type of network information. This way, the MASE developers do not have to port MASE to the various types of network hardware, such as routers.

Eventually, MASE will progress to the point where the expert systems will act as a network management adviser. Based upon statistical information gathered by MASE, it might suggest how a network could be reorganized, based upon traffic flow, and resource utilization. For instance, it may suggest, as a result of traffic analysis, that the insertion of a bridge at a certain point in the network will reduce the load on each side of the bridge by 40%. Or, it may suggest how to distribute a file system better, so that a particular node which is being over-utilized will share some of its load with a second node.

Configuring the network security expert system

Computer security on a network can be analyzed at various levels. The lowest level is the machine level. It could be argued that the security of a network is only as strong as the security on the weakest node on the network. This is an overstatement, but not as irrelavent as one might think.

At the network level, MASE will use it's packet monitoring capability for several security checks. An example of one follows:

In every ethernet packet, the hardware address of the ethernet card which transmits the packet is included. If it is an Internet Protocol (IP) packet, the IP address of the machine is included also. Since MASE will be monitoring network traffic, it can compile a list of all the physical ethernet addresses, and their associated IP addresses. One method of attacking a computer network is to disguise one machine so it looks like another. This is done by changing the internal networking information on the system. This type of attack would be detected by MASE, however. Changing the IP address of a machine is fairly simple, given the right privileges. However, changing the physical ethernet address of the ethernet card is not. When the disguised machine starts transmitting packets using the false IP address, MASE will detect the anomaly (conflicting physical addresses) and report the problem.

Since MASE is a distributed system, the various nodes can supply information that can lead to detection of other types of intruder attacks. For instance, if one node reports that an account was accessed improperly ten times (invalid passwords), then a problem may exist. However, if a second machine reports shortly thereafter that it has an account that is being tested (multiple password guesses), and the originating user is using the flagged account on the machine that intially reported the password attack. This should raise the possibility of an intruder attack to a much higher level.

User Assistance

One of the problems that users face on a network of computers is knowledge of, and access to, network resources. MASE will assist users in this regard. It will keep track of the various network resources, such as printers, disk drives, tape drives, application software, development software, etc. This will be maintained in a database, which the user will then be able to query against. Example:

A user has a dvi formatted file, that s/he wants to print out. A query would be made to MASE, asking what machine has a dvi to postscript utility, a postscript printer, and access in some way (User Account, Guest Account, etc).

System Manager's Interface

First, it is important that the system manager is shown what the network looks like, topologically. Second, the system manager needs to be able to access information quickly on each of the nodes. Third, the system manager needs to be able to configure MASE to meet the requirements of the particular network. This includes thresholds, corrective actions, notification levels, and frequency settings.

The MASE user interface is designed to handle each of these items. The display is set up hierarchically, enabling the manager to move up and down throughout the network, displaying items such as subnets, routers, individual nodes, node resources, and finally, the expert systems that are being executed. Each item is represented iconically, allowing the user to select an item, in order to obtain more information about the item. For instance, if the item selected by the user is a subnet, then all the nodes on the selected subnet are displayed.

One of the features of this interface is a graph-building capability. This allows the system manager to design and implement his/her own expert systems, for solving problems specific to his/her own network.

Finally, one of the problems of user interfaces for complex systems, is information overload. This is overcome in MASE, by using the manager's expert system to control the interface. Messages coming in from across the network are prioritized. For instance, a message alerting the system manager that a printer is down does not carry the same importance that "an intruder detected" message does. The controlling expert system will use the priority scheme to determine which information should be displayed before other information.

Conclusions

MASE is an on-going project that is continuously evolving. It is being funded through a RICIS contract for NASA Johson Space Center. The first phase of the project was to develop an Alpha version of MASE.. This included basic diagnostic function development, preliminary user interface development, and expert system development.

MASE goes a step further than other network packages, by providing complex rule-generation systems that allow system managers to create their own custom automated networking experts. The ability to combine existing commands and expert systems into larger, more intelligent systems is a more powerful and dynamic solution than today's rigid, stiff alternatives.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1991	3. REPORT TYPE AND DATES COVERED Conference Publication

4. TITLE AND SUBTITLE 2nd CLIPS Conference Proceedings Volume 1	5. FUNDING NUMBERS
6. AUTHOR(S) Joseph Giarratano (UHCL) and Christopher Culbert (JSC)	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) NASA Johnson Space Center Software Technology Branch/PT4 Houston, Texas 77058	8. PERFORMING ORGANIZATION REPORT NUMBER S-662
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Washington, D.C. 20546-001	10. SPONSORING / MONITORING AGENCY REPORT NUMBER CP 10085

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Unlimited/Unclassified Subject Category 61	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Papers presented at the 2nd CLIPS Conference held at the Lyndon B. Johnson Space Center (JSC) September 23, 24, and 25, 1991 are documented herin. CLIPS is an expert system tool developed by the Software Technology Branch at NASA JSC and is used at over 4000 sites by government, industry, and business. During the three days of the conference, over 40 papers were presented by experts from NASA, Department of Defense, other government agencies, universities, and industry.

14. SUBJECT TERMS CLIPS, expert sytems, knowledge-based systems, Space Shuttle, intelligent tutors, verification and validation, simulation	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited