NASA Conference Publication 10162 Vol. I

Third CLIPS Conference Proceedings

Gary Riley, Editor
Lyndon B. Johnson Space Center
Houston, Texas

Proceedings of a conference sponsored by
Lyndon B. Johnson Space Center and I-NET, Inc.
and held at the Johnson Space Center, Houston, Texas
September 12-14, 1994

1994

This publication is available from the NASA Center for AeroSpace Information,
800 Elkridge Landing Road, Linthicum Heights, MD 21090-2934, (301) 621-0390.

ABSTRACT

Expert systems are computer programs which emulate human expertise in well defined problem domains. The potential payoff from expert systems is high: valuable expertise can be be captured and preserved, repetitive and/or mundane tasks requiring human expertise can be automated, and uniformity can be applied in decision making processes. The C Language Integrated Production System (CLIPS) is an expert system building tool, developed at the Johnson Space Center, which provides a complete environment for the development and delivery of rule and/or object based expert systems. CLIPS was specifically designed to provide a low cost option for developing and deploying expert system applications across a wide range of hardware platforms. The development of CLIPS has helped to improve the ability to deliver expert system technology throughout the public and private sectors for a wide range of applications and diverse computing environments. The Third Conference on CLIPS provided a forum for CLIPS users to present and discuss papers relating to CLIPS applications, uses, and extensions.

CONTENTS

VOLUME I

VOLUME II

omit

Session 1A: Medical and Diagnostic Applications

Session Chair: Greg Madey

The Buffer Diagnostic Prototype: A Fault Isolation Application Using CLIPS

Ken Porter
Systems Engineer
MS: R1-408
Harris Space Systems Corporation
295 Barnes Blvd.
PO Box 5000
Rockledge, FL 32955

This paper describes problem domain characteristics and development experiences from using CLIPS 6.0 in a proof-of-concept troubleshooting application called the Buffer Diagnostic Prototype.

The problem domain is a large digital communications subsystem called the Real-Time Network (RTN), which was designed to upgrade the Launch Processing System used for Shuttle support at KSC. The RTN enables up to 255 computers to share 50,000 data points with millisecond response times. The RTN's extensive built-in test capability but lack of any automatic fault isolation capability presents a unique opportunity for a diagnostic expert system application.

The Buffer Diagnostic Prototype addresses RTN diagnosis with a multiple strategy approach. A novel technique called "faulty causality" employs inexact qualitative models to process test results. Experiential knowledge provides a capability to recognize symptom-fault associations. The implementation utilizes rule-based and procedural programming techniques, including a goal-directed control structure and simple text-based generic user interface that may be re-usable for other rapid prototyping applications. Although limited in scope, this project demonstrates a diagnostic approach that may be adapted to troubleshoot a broad range of equipment.

ON THE DEVELOPMENT OF AN EXPERT SYSTEM FOR WHEELCHAIR SELECTION

Gregory R. Madey [1]
Charlotte A. Bhansin [2]
Sulaiman A. Alaraini [1]
Mohamed A. Nour [1]

ABSTRACT

The prescription of wheelchairs for the Multiple Sclerosis (MS) patients involves the examination of a number of complicated factors including ambulation status, length of diagnosis, funding sources, to name a few. Consequently, only a few experts exist in this area. To aid medical therapists with the wheelchair selection decision, a prototype medical expert system (ES) was developed. This paper describes and discusses the steps of designing and developing the system, the experiences of the authors, and the lessons learned from working on this project. Wheelchair_Advisor, programmed in CLIPS, serves as a diagnosis, classification, prescription, and training tool in the MS field. Interviews, insurance letters, forms, and prototyping were used to gain knowledge regarding the wheelchair selection problem. Among the lessons learned are that evolutionary prototyping is superior to the conventional system development life-cycle (SDLC), the wheelchair selection is a good candidate for ES applications, and that ES can be applied to other similar medical subdomains.

[1] Kent State University, Kent, Ohio, 44240; (gmadey@synapse.kent.edu)
[2] Cleveland Clinic, 9500 Euclid Avenue, Cleveland, Ohio, 44195.

INTRODUCTION

The medical field was one of the first testing grounds for Expert System (ES) technology; the now classic expert system, MYCIN, has often been cited as one of the great breakthroughs in Expert Systems. MYCIN, however, is only one of a large number of expert system applications introduced over the last two decades in the medical field alone [Waterman, 1986]. Other examples include NURSExpert [Bobis and Bachand, 1992], CENTAUR, DIAGNOSER, MEDI and GUIDON [Waterman, 1986], MEDICS [Bois et al., 1989], and DiagFH [Lin and Tang, 1991] to mention only a few. However, no expert system, to our knowledge, has been developed for the wheelchair selection problem. In this paper, we report on a new application of ES in the medical field; the paper discusses the experiences of the authors with a prototype system developed, using CLIPS, to delineate a wheelchair selection for multiple sclerosis (MS) patients. Our work, therefore, contributes to the existing applications of medical expert/support systems by expanding the domain of applications to the wheelchair selection problem and demonstrating the utility of CLIPS on this problem domain. We will demonstrate that the complexity of the wheelchair selection decision makes it a prime target for an expert system application.

To prescribe a wheelchair for a patient with MS involves more than knowing the patient's disease condition and available choices of potential wheelchairs. A complex web of factors has to be untangled to reach an appropriate choice of a wheelchair. The decision is complicated by such factors as physical body measurements, age and life style, degree of neuralgic impairment, and environmental factors.

MOTIVATIONS FOR COMPUTER-AIDED WHEELCHAIR SELECTION

The motivations for using computer-aided wheelchair selection support fall into two categories: those from the therapist's standpoint and those from the patients' standpoint.

From the Therapist's Standpoint:

The use of computer-aided selection of wheelchairs benefits the therapist in several ways. The prescription of wheelchairs for the MS population is complicated by diverse presentations and symptom fluctuations as well as many other factors. The selection of a well-suited wheelchair is a function of the following variables.

1. Ambulation status: In general, a patient is classified as able to walk or not able to walk. In multiple sclerosis, some ability to walk may be limited to short distances or specific terrains. This factors into the type of wheelchair they will need.
2. Environments to be traversed: This variable includes both indoor and outdoor locations. The existence of ramps in the patient's residence and the size of the bathroom are among the environmental factors relevant to the choice of an appropriate wheelchair. The frequency of need in each environment helps determine the priority.
3. Distances to be traversed: Under this factor, both the indoor and outdoor activities are considered. Self-propelling for short distances may be feasible for some individuals who stay primarily at home, so a manual wheelchair may be appropriate, although disability level may be high. More active users may require a power wheelchair to travel long distances in the community.
4. Transport of the wheelchair: Consideration must be made for the wheelchair to disassemble into parts so as to fit in a car or to be sized to fit on public lift-equipped busses.
5. Caregiver characteristics: If a caregiver exists for the patient in question, the characteristics of the caregiver(s) are considered in the wheelchair selection. The age, number of caregivers, tolerance for equipment, their health, and the degree of support for the patient are some of the factors to be evaluated when selecting a wheelchair for the patient.
6. User characteristics: This variable includes both physical and cognitive dimensions. For the physical, body measurements of the patient are essential to the selection processes, along with qualities of posture, balance, and abnormal muscle tone. Wheelchairs come in made-to-order frame sizes and appropriate sizing is essential. Physical abilities are also evaluated: voluntary movements of the extremities and head are noted. Areas requiring support for best posture and function are documented. As for the cognitive dimension, physical and occupational therapists have to consider the extent to which the patient can safely use the devices. Some other issues to be examined by the therapists are the ability to learn the electronic system of power wheelchair, to respond quickly in dangerous situations and the ability to report discomfort or problems with fit of the wheelchair.
7. Length of diagnosis—history of disease course: This composite variable aids in determining if the MS symptoms of the individual are stable. If they seem stable, a less-modular wheelchair can be appropriate. A progressive disease course would require many modular options for future needs; as the MS symptoms change, it would be possible to modify the wheelchair to fit the needs of the patients.
8. Currently owned wheelchairs: Therapists need to consider this item early in their analysis. The current wheelchair may or may not meet some of the needs of the patient. One possibility is that the current wheelchair can be modified to meet the patient's needs. Another possibility is that the wheelchair needs to be replaced because it is inappropriate for current needs, beyond repair or desired modifications can not be performed.
9. Funding sources of past and potential wheelchairs: This factor is considered at the end of the process but it is a crucial one. Most patients are restricted in terms of the number of wheelchairs that they can purchase over time under their insurance coverage. Typically, a therapist examines and evaluates the factors that determine the needs of the patient to narrow down the choices of the available wheelchairs. Once the options are reduced, the therapist uses the funding source variable to choose among the options. The funding sources can include Medicaid, Medicare, private insurance (e.g., third party), private purchase, or charity (e.g., MS Society Equipment Loan Program). Each one of these sources has its own rules regarding the wheelchair selection problem. For example, some policies restrict the purchase of a new wheelchair to one every five years. Some will not cover a manual wheelchair if an electric wheelchair was previously

obtained. Hence, such restrictions need to be factored in when considering the selection of an appropriate wheelchair.
10. Current wheelchairs on the market: There are over 500 models, each offering sporting multiple options of sizing, weight, frame styles, footrests, armrests, cushions, and supports. A current database of technical information would greatly aid in wheelchair selection.

Note that the degree of importance placed on each one of the foregoing factors is not fixed. There is a complex interaction between variables for each patient under consideration. It can be seen from the above illustration that selecting an appropriate wheelchair would be difficult to solve algorithmically. Hence, an expert system is a good candidate for this kind of problem. It was observed by the expert involved in this pilot project that the process of selecting the wheelchair involves both forward and backward reasoning. A therapist starts with the factors that are considered to be important to the patient in question and then narrows down the options available to the patient. This process involves forward reasoning and it is estimated to be eighty percent (80%) of the overall analysis performed by the therapist. The rest of the reasoning, twenty percent (20%), is devoted to backward chaining where the therapist starts with a specific set of wheelchairs and sees if they meet the needs of the patient as well as the requirements of the funding source.

A computer-aided support system can play a significant role in helping the therapist cope with the factors mentioned above. It can guide the therapist in making the best decision about what wheelchair and features need to be prescribed, based on comparison to other successful cases. It can aid the therapist by insuring thorough evaluation. Also, it can help the therapist keep abreast of new products on the market. Such a system insures quality in the wheelchair selection process. An inappropriately prescribed wheelchair usurps coverage and prevents re-prescription of a more appropriate chair. In addition, the standardized reporting format could also be used to conduct more objective studies on wheelchair prescription.

Such a system also has value as a training tool for both novice therapists and therapy students. A tutorial in which real-life or simulated applications are demonstrated can be used for teaching and training. Furthermore, innovations in the wheelchair industry change frequently. The use of computer-based support can overcome this problem. A database of currently available wheelchairs kept and updated on a regular basis, is needed in the field of rehabilitation technology. Finally, the documentation of valuable expertise as reflected by real-life applications will be easier using a computer based system. In this context, an expert therapist is a scarce resource. Hence, years of experience involving the prescription of numerous wheelchairs can be stored in the system and used later as a reference by therapists who practice in more general areas.

From the Patient's Standpoint:

Of all patients with Multiple Sclerosis (MS), about 40 percent will lose the ability to ambulate [Poser, 1978]. Thus, wheeled mobility stands out as a primary need in this population. Because of the nature of the wheelchair selection problem, it is not unusual for the medical therapist/specialist to prescribe a seemingly appropriate wheelchair for a particular patient only to have the patient reject the wheelchair. The importance of the selection of an appropriate wheelchair for a particular patient cannot be overstated. From the MS patient's standpoint, the selection of a suitable wheelchair is critical for the following reasons:

1. Insurance: Because of funding restrictions, the patient might be restricted to a wheelchair for a minimum number of years before being eligible for another wheelchair. The MS patient wants to be sure the right chair is prescribed.
2. Cost: The prescription of an appropriate wheelchair should take the cost factor into consideration, especially if the patient is to bear that cost, for patients' resources vary. Also cost consideration is important due to funding restrictions imposed by insurers or Medicaid/Medicare programs. The costs of a wheelchair can range from several hundred to several thousands of dollars.

3. Mobility and comfort: The selection of an inappropriate wheelchair will limit already diminished mobility and deny the individual MS patient the potential for increased functional independence from an otherwise suitable wheelchair.
4. Health: An inappropriate wheelchair not only may inhibit mobility and cause discomfort, but it may worsen the patient's condition, e.g., postured deformities, pressure sores, etc.
5. Image and psychological factors: A suitably selected wheelchair might enhance the patient's personal image, and thus contribute to more community/social involvement. For example, a young MS patient might desire a sporty wheelchair to remain active and socially involved.

Because of the foregoing reasons, it is desirable to have a computer-aided wheelchair selection support system that will hopefully maximize the benefits in the selected wheelchair.

Rationale For Using An Expert System

As was discussed earlier, the selection of the wheelchair for the MS population involves the examination of presentations and symptom fluctuations. Because of the complexity of these factors, only a few therapists are available with a body of expertise to tackle the wheelchair selection decision. A computer-aided system, however, would capitalize on this expertise and make it more widely available. Hence a knowledge-based system seems appropriate, more specifically, a knowledge-based expert system. The next section discusses medical expert systems in general and develops a taxonomy for them. We then show where our prototype system fits relative to this taxonomy.

TAXONOMIC FRAMEWORK FOR MEDICAL EXPERT SYSTEMS

The wide range of intelligent (knowledge-based) medical systems today can be broadly classified using the taxonomy shown in Figure 1. This taxonomy is based on three broad dimensions: technology, domain, and application type.

A. Technology

Technology is further divided into: 1) hardware platform (e.g. PC-based, workstation-based, etc.), 2) AI method (solution), and 3) programming tools. A medical knowledge-based system can thus be classified on whether it is PC-based, mainframe-based, etc. It can also be classified on whether it is an expert system solution [Cagnoni and Livi, 1989], a neural network (ANN) [Kuhn et al., 1991], a natural language system, interactive hypermedia [Hammel, 1992], a paper-based [Ward and Reed, 1993], etc. Programming tools include AI programming languages and shells. Examples include OPS5, Lisp, Prolog, and CLIPS [Stylianou and Madey, 1992].

B. Domain

Knowledge-based systems have been applied in a variety of medical subdomains [Prasad et. al., 1989; Cagnoni and Livi, 1989; Waterman, 1989; Bobis and Bachand, 1992a; Bobis and Bachand, 1992b; Bois et al., 1989; Lin and Tang, 1991]. Example subdomains include: heart diseases, blood analysis, asthma, artificial limbs, childhood diseases, and this project on multiple sclerosis (MS). It is difficult, however, to neatly classify medical computer-aided systems on the basis of medical subdomains since many of these systems have overlapping domains.

C. Application Type

The application type dimension describes the function of the knowledge-based system for which it is developed. These applications types include diagnosis [Lin and Tang, 1991], classification [Waterman 1986], prescription/selection [Stylianou and Madey, 1992], tutoring/training [Prasad et

al., 1989], data analysis and interpretation, prognosis, and knowledge/technology transfer. Many knowledge-based systems are built to support more than one of these functions.

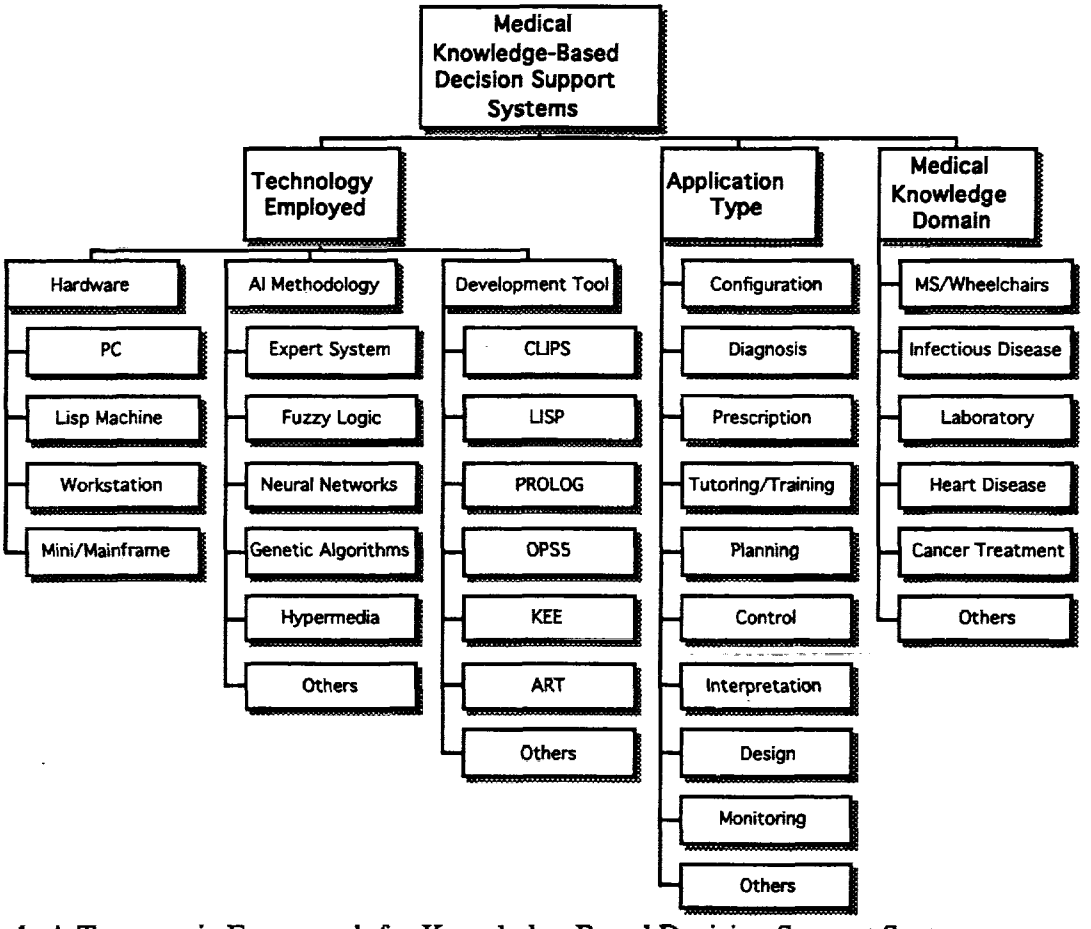


Figure 1: A Taxonomic Framework for Knowledge-Based Decision Support Systems

Review of Medical Expert Systems

Our emphasis in this paper is on medical *expert systems*, which is a subset of the computer-aided support systems in the technology dimension mentioned above. Some of the well known medical expert systems include the following [Waterman, 1986, pp.272-288]:

1. CENTAUR: The domain of this expert system is lung diseases, developed in the INTERLISP programming tool by the Stanford University. Operational functions include diagnostic interpretation of pulmonary function tests.
2. DIAGNOSER: Deals with heart diseases, develop in LISP by the University of Minnesota.
3. GUIDON: The medical domain include bacterial infections. It is developed in INTERLISP by the Stanford University.
4. MDX: Deals with liver problems, developed in LISP by the Ohio State University.
5. MED1: Deals with chest pain, developed in INTERLISP at the University of Kaiserlautern.
6. MYCIN: Best known of all medical expert systems, MYCIN's medical subdomains include bacteremia, meningitis, and Cystis infections. It was developed at Stanford University and the main operational functions include diagnosis of the causes of infections, treatment, and education.
7. NEUREX: Concerned with the nervous system, NEUREX was developed in LISP at the University of Maryland. Its functions include diagnosis and classification of the diseases of the nervous system.

8. CARAD: This expert system handles radiology; it was developed at the Free University of Brussels. Its main functions is the interpretation and classification of X-ray photographs [Bois et.al., 1989].

Our Wheelchair_Advisor stands apart from these expert systems listed above by its unique domain of wheelchair prescription for MS patients and our choice of the programming tool. This project involved the use of a PC and the expert system shell CLIPS [NASA, 1991; Giarratano and Riley, 1994; Wygant, 1989; Gonzalez and Dankel, 1993]. The functions/objectives of the Wheelchair_Advisor included diagnosis, classification, prescription, and training. Figure 2 maps these characteristics into a classification scheme to show where our prototype expert system fits relative to current computer-aided medical systems. As Figure 2 indicates, and to our best knowledge, no other expert system application has been developed in the domain of wheelchairs for MS patients.

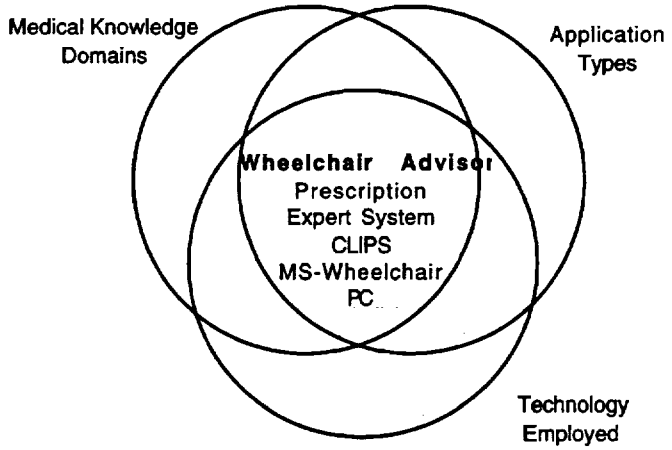


Figure 2: Classification Framework for Medical Decision Support Systems

THE WHEELCHAIR EXPERT SYSTEM PROJECT

A. The Environment

The Cleveland Clinic Foundation's Mellen Center for Multiple Sclerosis Treatment and Research was initiated in 1985 with a grant from the Mellen Foundation. The Mellen Center, the largest and most comprehensive full-time MS center in the country, is an interdisciplinary outpatient rehabilitation facility providing direct patient care, education, and basic and clinical research into the causes and management of MS. In 1993, the Mellen Center had 14,000 patient visits for its services of neurology, nursing, occupational therapy, physical therapy, psychology, and social work. Approximately 350 new patients are seen each year.

B. The Knowledge Engineering Process

The knowledge engineering process has often been described as the "knowledge engineering bottleneck" due to the difficulty and complexity of this process. To deal with the complexity of the knowledge engineering process, three basic methodologies were used to elicit knowledge from the expert: interviews, insurance documents, forms, and prototyping.

1. Interviews

Multiple interviews were conducted with the expert by three knowledge engineers (KE) all of whom, including the expert, are the authors of this paper. A typical session lasted from 3 to 5 hours.

2. Insurance Letters/Other Forms

The insurance and other prescription forms supplied the knowledge engineers with the missing links in the pieces of knowledge gained from the interviews. These forms embodied actual cases describing patient symptoms, condition, cognitive/psychological state, and the recommended wheelchair. Because of the difficulties of obtaining sufficient knowledge using interviews only, as pointed out above, the knowledge obtained from these documents was invaluable inasmuch as it complemented the expertise derived directly from the expert.

3. Prototyping

The interviews went side by side with an actual prototype developed to foster better communication between the expert and the KE's. This helped offset some of the limitations of the interviewing process. Each subsequent version of the prototype provided a chance for the expert to "endorse" the KE's interpretation of the knowledge supplied in the previous interview. At times the expert would clarify a previous answer and supply a new one; thus it became clear that the prototype helped correct errors in communication and misinterpretations.

C. The System-Building Process

The project was conducted in an interactive fashion and rapid prototyping was used to develop the system. Figure 3 shows the block diagram of the prototype system. First, the patient's needs and constraints are considered. This data can be provided on line or by using an input text file in which the data about a particular patient is stored. To accomplish this task a number of rules of the type IF/THEN are implemented. The result of this examination, which is a template of facts about the patient in question, is then used by the search module which in turns uses this information while searching the wheelchair database to find the appropriate wheelchair(s). Note that the optimizer module consists also of IF/THEN rules. As for the wheelchair database, it contains a list of wheelchairs with different features. An explanation facility where the reasoning of the system is explained to the user can be added to the system. Finally, there is a solution set module where the recommendations of the ES are included. In the next subsection, a description of CLIPS, an expert system language, is presented. Then, sample screens and dialogue are shown.

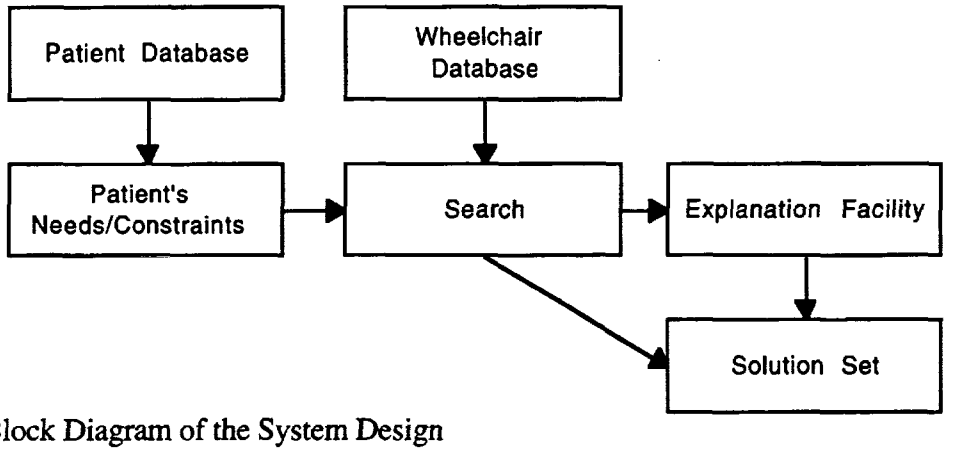


Figure 3: Block Diagram of the System Design

CLIPS

CLIPS (short for C Language Integrated Production System), developed at NASA/Johnson Space Center, has recently shown increasing usage [NASA, 1991; Giarratano and Riley, 1994; Gonzalez and Dankel, 1993; Martin & Taylor, 1992]. CLIPS is a forward-chaining rule-based language that resembles OPS5 and ART, other widely known rule-based development environments. Figure 4

shows the basic components of CLIPS, which are essential for an ES. Following this figure is a brief description of each component.

1. User Interface: The mechanism by which the user and the expert system communicate.

2. Fact-list: A global memory for data. For example, the primary symptom of an MS patient can be represented in CLIPS syntax as in Figure 5. For clarity, the reserved key words of CLIPS are printed in bold letters.

3. Knowledge-base: Contains all the rules used by the expert system. For instance, consider the following partial rule that is used by the system to list all the primary symptoms of an MS patient:

IF user has a primary symptom of cerebellar ataxia
THEN the primary symptom is cerebellar ataxia

In the CLIPS syntax, this rule and the associated dialogue can be written as shown in Figure 6.

4. Inference engine: Makes inferences by deciding which rules are satisfied by facts, prioritizes the satisfied rules, and executes the rule with the highest priority.

5. Agenda: A prioritized list created by the inference engine of instances of rules whose patterns are satisfied by facts in the fact list. The following shows the contents of the agenda at some stage:

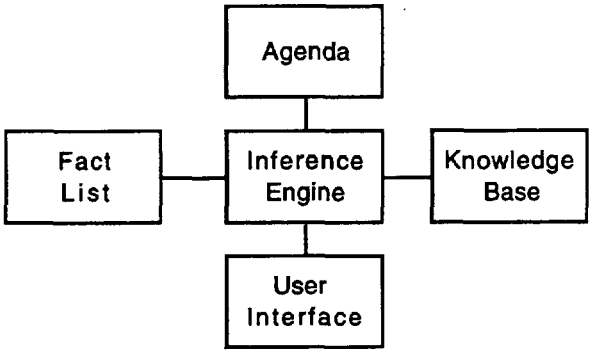


Figure 4: CLIPS Basic Components. Adapted from [Giarratano and Riley, 1994].

English:
The primary symptom of the patient is cerebellar ataxia.

CLIPS:

```
(deffacts user-data
    (ms symptoms primary cerebellar ataxia)
)
```

Figure 5: CLIPS Syntax for storing facts

n Figure 7, three instantiated rules are placed on the agenda. Each entry in the agenda is divided into three parts: Priority of the rule instance, name of the rule, and the fact-identifiers. For the first entry in the agenda, for example:

- 2 refers to the priority.
- ms-symptom-primary is the name of the rule.
- f-5 is the fact-identifier of the fact that matches the pattern of the rule. Such facts are stored as in Figure 5.

```
(defrule ms-symptoms-primary
     ?phase <- (phase ms symptom)
=>
     (retract ?phase)
     (printout t crlf "What is the primary symptom of the MS
                       patient? ")
     (bind ?answer (readline))
     (if (not (stringp ?answer))
     then (printout t crlf "Please check again!" crlf)
     (assert (phase ms symptom))
     (if (stringp ?answer)
     then (bind $?sym (str-explode ?answer))
     (assert (ms symptoms primary $?sym secondary))))
```

Figure 6: CLIPS Syntax for rules

```
Agenda
     2 ms-symptoms-primary:          f-5
     1 ms-symptoms-secondary:        f-6
     0 ms-symptoms-secondary-more    f-7,f-8
```

Figure 7: CLIPS Agenda

Sample Screens And Dialogue

The above rule, the ms-symptoms-primary rule, can be used to show a scenario of a dialogue between the end user (e.g., a physical therapist) and the expert system as follows:

```
WHAT IS THE PRIMARY SYMPTOM OF THE MS PATIENT?
cerebellar ataxia
WHAT IS THE SECONDARY SYMPTOM OF THE MS PATIENT?
weakness
```

Figure 8: A Sample screen of a dialogue in a session

Based on the new information provided by the end user, the data about the patient will be updated. Accordingly, the fact-list will include a new fact which shows the name of the primary symptom of this patient. The resulting fact is presented in Figure 5. Another impact of this new information will be to update the agenda to include the next rule to be fired, the ms-secondary-symptom rule in this case. This is possible because a new fact, f-5, which was entered by the user as an answer to an on-screen question, now satisfies this rule.

LESSONS LEARNED

There are many lessons to be learned from this project. First: the evolutionary prototyping in designing expert systems is proven to be superior to conventional system development life-cycle. Figure 9 shows the steps involved in designing a system under the traditional method.

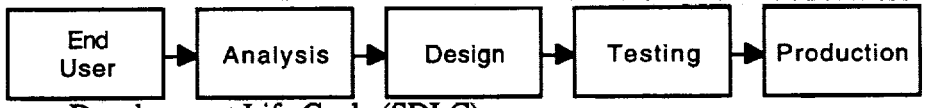


Figure 9: System Development Life Cycle (SDLC)

On the other hand, prototyping presents a more efficient way to design a system. Under this method, the end user will be aware of the costs/benefits and, most importantly, will be a part of the development team. In essence, the system will be modified a number of times until the desired system is obtained. Figure 10 shows the steps involved in this method.

Figure 10 : Evolutionary Prototyping

Second: the expert system developed in this project has shown the wheelchair selection problem to be a good candidate for ES applications. This project has also shown that there are major benefits for both the medical practitioners and the MS patients to be derived from such an application. Third, it is evident from this project that other similar medical subdomains might be good candidates for the application of the ES technology. Our project serves to expand the medical applications domain. Fourth, CLIPS was found to flexible, powerful, and intuitive development environment for this application.

CONCLUSIONS

The authors of this paper were involved in a project concerned with the actual development of a wheelchair selection expert system. A prototype expert system (Wheelchair_Advisor) was developed, using CLIPS, to prescribe wheelchairs for Multiple Sclerosis (MS) patients. This paper reports the process, the experiences of the authors, the advantages of evolutionary prototyping for expert system development, and the possibilities for new medical subdomains as candidates for expert system applications.

Our findings show that there are major advantages for using an expert system tool to aid in the analysis and selection of a wheelchair for an MS patient. Such an expert system can also be used as a training and educational tool in the medical industry.

REFERENCES

Bobis, Kenneth G. and Bachand, Phyllis M., "URSExpert: An Integrated Expert System Environment for the Bedside Nurse," in proceedings of **IEEE International Conference on Systems, Man and Cybernetics**, Chicago, 1992, pp. 1063-1068.

_____ "Care Plan Builder: An Expert System for the Bedside Nurse," in proceedings of **IEEE International Conference on Systems, Man and Cybernetics,** Chicago, 1992, pp. 1069-1074.

Bois, Ph. DU, Brans, J.P., Cantraine, F., and Mareschal, B., "MEDICS: An expert system for computer-aided diagnosis using the PROMETHEE multicriteria methods," **European Journal of Operations Research**, Vol. 39, 1989, pp. 284-292.

Cagnoni S. and Livi R., "A Knowledge-based System for Time-Qualified Diagnosis and Treatment of Hypertension," in Computer-Based Medical Systems: Proceedings of the **Second Annual IEEE Symposium**, June 26-27, 1989, Minneapolis, Minnesota. pp. 121-123.

Fieschi, M., **Artificial Intelligence in Medicine**, Chapman and Hall, London, 1990.

Giarratano, Joseph and Riley, Gary, **Expert Systems: Principles and Programming**, PWS-Kent Publishing Company, Boston, 1994.

Gonzalez, Avelino J. and Dankel, Douglas D., **The Engineering of Knowledge-Based Systems: Theory and Practice**, Prentice-Hall, Englewood Cliffs, NJ, 1993.

Hammel, Joy M., "Final Report of the AOTA/Apple Grantees," Veterans Administration, Palo Alto, CA, March 1992.

K. Kuhn, D. Roesner, T. Zammler, W. Swobodnik, P. Janowitz, J. G. Wechsler, C. Heinlein, M. Reichert, W. Doster, and H. Ditshuneit., "A Neural Network Expert System to Support Decisions in Diagnostic Imaging," in **Proceedings of IEEE 4th Symposium on Computer-Based Medical Systems,** May 12-14, 1991, Los Angeles, pp. 244-250.

Lin, W. and Tang, J.-X., "DiagFH: An Expert System for Diagnosis of Fulminant Hepatitis," in **Proceedings of IEEE 4th Symposium on Computer-Based Medical Systems**, May 12-14, 1991, Los Angeles, pp. 330-337.

Martin, Linda and Taylor, Wendy. **A Booklet of CLIPS Applications**, NASA, Johnson Space Center, Houston, TX, 1992.

Mouradian, William H., "Knowledge Acquisition in a Medical Domain," **AI Expert**, July 1990, 34-38.

NASA, Lyndon B. Johnson Space Center, **CLIPS Basic Programming Guide**, 1991., Houston, TX.

Prasad, B. , Wood H., Greer, J. and McCalla G., "A Knowledge-based System for Tutoring Bronchial Asthma Diagnosis," in **Computer-Based Medical Systems: Proceedings of the Second Annual IEEE Symposium**, June 26-27, 1989, Minneapolis, Minnesota. pp. 40-45.

Stylianou, Anthony C. and Madey, Gregory R., "An Expert System For Employee Benefits Selection: A Case Study," **Journal of Management Systems**, Vol. 4, No. 2, 1992, pp. 41-59.

Ward, Diane and Reed, Stephanie, "PowerSelect: A Prototype for Power Mobility Selection—Based Upon Human Function," in **Proceedings of the Ninth International Seating Symposium,** February 25-27, 1993, Memphis, TN, pp. 307-310.

Waterman, Donald A. **A Guide to Expert Systems**, Addison-Wesley Publishing Company, 1986.

Wygant, Robert M., "Clips—A Powerful Development and Delivery Expert System Tool" **Computers in Engineering**, Vol. 17, Nos. 1-4, 1989, pp. 546-549.

EXPERT WITNESS - A SYSTEM FOR DEVELOPING EXPERT MEDICAL TESTIMONY

Raymond Lewandowski, MD.
Center for Genetic Services
7121 S. Padre Island Dr.
Corpus Christi, Texas 78412

David Perkins and David Leasure
Department of Computing and Mathematical Sciences
Texas A&M University - Corpus Christi
6300 Ocean Dr.
Corpus Christi, Texas 78412

ABSTRACT

Expert Witness is an expert system designed to assist attorneys and medical experts in determining the merit of medical malpractice claims in the area of obstetrics. It this by substitutes the time of the medical expert with the time of a paralegal assistant guided by the expert system during the initial investigation of the medical records and patient interviews. The product of the system is a narrative transcript containing important data, immediate conclusions from the data, and overall conclusions of the case that the attorney and medical expert use to make decisions about whether and how to proceed with the case. The transcript may also contain directives for gathering additional information needed for the case.

The system is a modified heuristic classifier and is implemented using over 600 CLIPS rules together with a C-based user interface. The data abstraction and solution refinement are implemented directly using forward chaining production and matching. The use of CLIPS and C is essential to delivering a system that runs on a generic PC platform. The direct implementation in CLIPS together with locality of inference ensures that the system will scale gracefully. Two years of use has revealed no errors in the reasoning.

1. INTRODUCTION

When preparing a medical malpractice lawsuit, an attorney must identify the relevant facts and use them to decide first if the case has merit. Usually, the attorney consults a medical expert to evaluate the client's medical records and to advise the attorney. The problems for attorneys and clients is that medical experts are both expensive and relatively scarce, the problem of determining fault is tedious and time consuming, and the case load is growing.

Our approach to this problem is to make a preliminary determination of merit without investing large amounts of time from a medical expert. Using an expert system called Expert Witness, the paralegal staff can be guided in their examination of medical records and conducting of client interviews. After data collection, Expert Witness produces a transcript of reasoning that aids the attorney and medical expert in determining the validity of a case. The transcript is very similar to what the medical expert would also have produced, except that it was created with far less expense. By taking this approach, an attorney can determine the preliminary merits of a lawsuit while saving substantial amounts of money. The attorney and medical expert can take on more work. Deserving cases are more likely to be pursued because more cases can be handled overall. Fewer non-meritorious, wasteful cases need be pursued, resulting in saved expense and anguish. Overall, in two years of operation, Expert Witness has been tested in 10 legal offices on numerous cases with no complaints, and based on the success of the system, significant development is planned to greatly expand its coverage.

This paper describes the functional architecture, the implementation, the history, and the plans for expansion of Expert Witness. It begins with a functional overview of the Expert Witness in Section 2. After the functional description, some typical cases are described in Section 3. In Section 4, the implementation of the current system in CLIPS and C is described, as is the history of the project, and the future directions. Results and conclusions are given in section 5.

2. FUNCTIONAL DESCRIPTION

The current domain of expertise for Expert Witness is obstetrical malpractice. The overall context in which Expert Witness determines the extent of medical malpractice is shown in Figure 1. To determine the fault of medical personnel, Expert Witness directs a paralegal in the search for relevant medical facts from patient records and patient interviews. Such information includes the family history, the patient history, the history of the mother prior to birth, the events and medical procedures performed at birth, and subsequent tests and treatment. Expert Witness builds a case file for each client. This multiple client feature allows the paralegal to start and stop data collection corresponding to the availability of information and access to the client. When sufficient data has been collected, a narrative transcript and a fact summary is produced. The narrative transcript is similar to what the medical expert would have produced. It marks the important details, such as confirming or disconfirming evidence, presents reasoning chains based on evidence, suggests further tests, and derives conclusions regarding the viability of the case. The transcripts and the fact summaries are used by the attorney and the medical expert to make the final decision whether malpractice contributed to the client's condition, and also to determine what additional data need collected. The general philosophy embedded in Expert Witness's knowledge is to only make conservative conclusions, based on practice that is well accepted in the field.

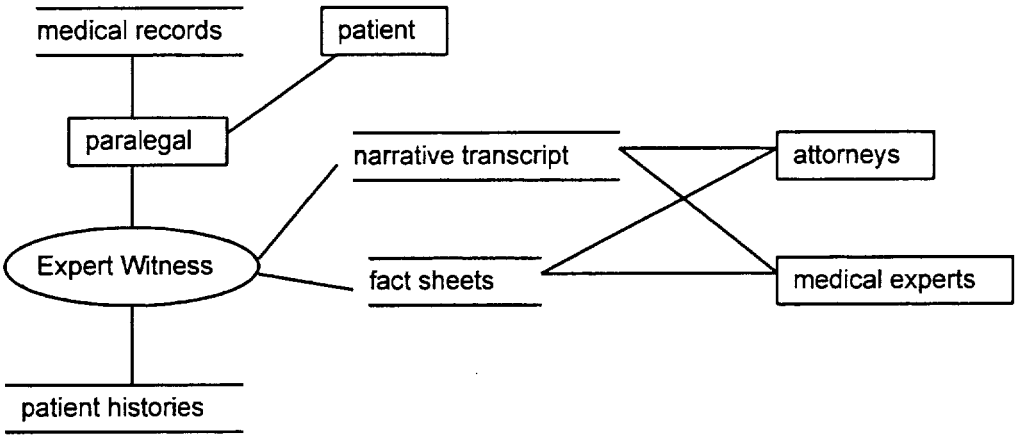


Figure 1. Context of Expert Witness

3. EXAMPLE CASES

The following cases are summaries of two actual cases handled by Expert Witness.

Case I

An infant was born with apgar scores of 8 and 9. The birth weight was six pounds. During the mothers labor, monitoring indicated that the baby was in distress. In response to the data suggesting distress, the physician treated the mother and reviewed the mother's medications. It was found that one of the medications that the mother was taking is known to create false positive findings of fetal distress. Normally, the distress patterns would have lead to a cesarean section. By reviewing the data correctly, the physician avoided an unnecessary surgery which carries added risks for the mother. The Expert Witness program analyzed the data and advised the user that the physician had acted appropriately based upon the facts presented. This analysis prevented a potentially frivolous law suite.

Case II

A child is known to be mentally retarded. The child was born with apgar scores of 2 and 5. During labor, the mother had a biophysical profile which was abnormal. After delivery, the infant developed low blood sugar and seizures. Family history revealed that the mother has a nephew by one of her sisters who is also mentally retarded. The Expert Witness program analyzed the data and advised the user that there appeared to be some improprieties on the part of the physician during the mother's labor that could have contributed to the child's present condition. It also noted, however, that there may have been a pre-existing condition which may be the main contributor to the child's problems. It suggested that further analysis is necessary. This is a case that deserved further in depth analysis by actual expert witness physicians.

4. IMPLEMENTATION, HISTORY, AND FUTURE DIRECTIONS

Expert Witness is used cyclically to build up a patient file. Within each cycle there are two stages, data collection and data inference. Data collection is done interactively as the paralegal presents known information to the system through a text based user interface written in C. Once all known information is provided, the inference phase begins, and the known data are analyzed to determine what conclusions are able to be made and what they are. When more information is needed, additional data are suggested in the transcript. The medical expert may also direct the paralegal to obtain more information. The next cycle of data collection/inference process allows direct entry of any additional information, and produces a more complete narration.

The inference part of the system is written in CLIPS 4.3.[1] Over 600 rules constitute the knowledge base. The basic architecture is an elaboration of the heuristic classification model.[2] Initial data are abstracted from quantitative values to qualitative categories. Matching, based directly on CLIPS rule matching, is used to determine the first level of solutions in the form of direct conclusions in the narrative transcript. Additional reasoning is performed to produce the next level of conclusions, based on the initial level. In contrast to some heuristic classifiers which seek to produce one conclusion and may take the first one that is satisfactory, Expert Witness makes all conclusions that it can. It uses a mix of reasoning methods, using some data to strengthen and some data to weaken conclusions. It does not use certainty factors or other approximate reasoning methods, since the qualitative representation of strength of belief using basic CLIPS was adequate for the conservative reasoning philosophy adopted for the system.

The performance of Expert Witness has been very good. The knowledge used has generally been localized, and the reasoning chains have been kept relatively short. Factoring of the rule base into a number of independent subsystems for determining the first level of conclusions has also helped. The second level conclusions are made using a rule base that is loaded after all first level conclusions have been made.

Expert Witness was built over a period of 5 months beginning in 1991. The initial knowledge engineer and expert was Dr. Ray Lewandowski, a medical consultant and clinical geneticist. The user interface was constructed by David Perkins at Texas A&M University-Corpus Christi. The system has since been used by ten attorneys and their staff. Follow-up consultations are performed with Dr. Lewandowski. Plans are underway to increase the number of users. In the several years since being introduced in the field environment, no incorrect recommendations have been made, and much time has been saved.

Based on the success of the initial system and demands of the users for broadening the scope of application, additional experts are currently being interviewed in the areas of neonatology, expanded obstetrical coverage, and hospital practices and procedures. Additional modules beyond those are in the planning stage. No significant changes to

the structure of the knowledge base are expected. Knowledge should remain localized, and the performance penalty should grow linearly with the number of systems. Each system will be incorporated so that it can function as a stand-alone or integrated component of the entire system.

5. RESULTS AND CONCLUSIONS

The system has since been used continuously since its development by ten attorneys and their staff. In the several years since being introduced in the field environment, no incorrect recommendations have been made, and much time has been saved. Based on this extended success, plans are underway to increase the number of users and the scope of the system's coverage.

A critical success factor for Expert Witness, aside from the quality of the knowledge base, has been the need for it to run on a generic hardware platform. The use of CLIPS has allowed us to keep the system small, while maintaining speed and ease of programming, both because the inference component is small and because it easily interfaced with a compact C user interface.

The second critical success factor derived from CLIPS is the suitability of the forward reasoning and matching to the application and representation of the knowledge. Although CLIPS would have allowed it, no meta-level reasoning was necessary. This simplicity allowed the knowledge base to grow to over 600 rules without greatly affecting the structural complexity of the knowledge or the cost of using it. On the face of it, the plainness of the knowledge representation as rules speaks against this system when compared to more complicated knowledge structures and control regimes, but in reality, the degree of fit between the knowledge and the inference system has allowed us to create and maintain a reasonably large knowledge base cheaply and reliably. This simplicity is crucial for us when we consider expanding the knowledge base as much as fivefold, which we intend to do.

REFERENCES

[1] CLIPS Release Notes, Version 4.3, NASA Johnson Space Flight Center, June 13, 1989.

[2] Clancey, W. J., "Heuristic Classification," *Artificial Intelligence*, **27:3**, December 1985, pp. 289-350.

Session 1B: Database and Object Oriented Programming Extensions

Session Chair: Allan Dianic

THE DESIGN AND IMPLEMENTATION OF EPL: AN EVENT PATTERN LANGUAGE FOR ACTIVE DATABASES*

G. Giuffrida and *C. Zaniolo*

Computer Science Department
The University of California
Los Angeles, California 90024
giovanni@cs.ucla.edu

Abstract

The growing demand for intelligent information systems requires closer coupling of rule-based reasoning engines, such as CLIPS, with advanced Data Base Management Systems (DBMS). For instance, several commercial DBMS now support the notion of triggers that monitor events and transactions occurring in the database and fire induced actions, which perform a variety of critical functions, including safeguarding the integrity of data, monitoring access, and recording volatile information needed by administrators, analysts and expert systems to perform assorted tasks; examples of these tasks include security enforcement, market studies, knowledge discovery and link analysis. At UCLA, we designed and implemented the Event Pattern Language (EPL) which is capable of detecting and acting upon complex patterns of events which are temporally related to each other. For instance, a plant manager should be notified when a certain pattern of overheating repeats itself over time in a chemical process; likewise, proper notification is required when a suspicious sequence of bank transactions is executed within a certain time limit.
The EPL prototype is built in CLIPS to operate on top of Sybase, a commercial relational DBMS, where actions can be triggered by events such as simple database updates, insertions and deletions. The rule-based syntax of EPL allows the sequences of goals in rules to be interpreted as sequences of temporal events; each goal can correspond to either (i) a simple event, or (ii) a (possibly negated) event/condition predicate, or (iii) a complex event defined as the disjunction and repetition of other events. Various extensions have been added to CLIPS in order to tailor the interface with Sybase and its open client/server architecture.

INTRODUCTION

A growing demand for information systems that support enterprise integration, scientific and multimedia applications has produced a need for more advanced database systems and environments. In particular, active rule-based environments are needed to support operations such as data acquisition, validation, ingestion, distribution, auditing and management —both for raw data and derivative products. The commercial DBMS world has sensed this trend and the more aggressive vendors are moving to provide useful extensions such as *triggers* and *open servers*. These extensions, however, remain limited with respect to functionality, usability and portability; thus, there remains a need for an enterprise to procure a database environment that is (1) more complete and powerful, and

* This work was done under contract with Hughes Aircraft Corporation, Los Angeles, California.

thus supports those facilities not provided by vendors, and (2) is more independent from specific vendor products and their releases and helps the enterprise to manage their IMS and cope with multiple vendors, data heterogeneity and distribution.

A particularly severe drawback of current DBMS is their inability of detecting patterns of events, where an event is any of the possible database operation allowed by the system; typically they are: *insertion*, *deletion* and *updating*. Depending on the application, sequences of events temporally related to each other, might be of interest for the user. In addition to basic database events, management of long transactions and deferred actions may be involved in such patterns. Practical examples of such meaningful patterns of events are:

- Temperature which goes down for three consecutive days;
- 2 day delayed deposit for out-of-state checks;
- 30 days of inactivity on a bank account;
- IBM shares increased value consecutively within the last week;
- Big withdrawal from a certain account followed by a deposit for the same amount on another account.

These and similar situations may require either a certain action to take place (e.g.: buy IBM shares) or a warning message to be issued (e.g.: huge transfer of money is taking place.) EPL gives the user the ability to handle such situations.

The purpose of this paper is to propose a rule-based language and system architecture for data ingestion. The first part of this paper describes the language, then the system architecture is discussed.

EPL DESCRIPTION

An EPL program consists of several named modules; modules can be compiled, and enabled independently. The head of each such module defines an **universe** of basic events of interest, which will be monitored by the EPL module. The basic events being monitored can be of the following three types:

```
insert(Rname), delete(Rname), update(Rname)
```

where *Rname* is the name of a database relation.

A module body consists of one or more rules having the basic form:

```
event-condition-list
 →
action-list
```

The left hand side of the rule describes a certain pattern of events. When such a pattern successfully matches with events taking place in the database the set of actions listed in the right hand side are executed.

For instance, assume that we have a database relation describing bank accounts whose schema is: **ACC(Accno,Balance).** We want to monitor the deposits of funds in excess of more than $100,000 into account 00201. In EPL, this can be done as follows:

```
begin AccountMonitor
    monitor update(ACC), delete(ACC), insert(ACC);

    update(ACC(X)),
    X.old_Accno = 00201,
    X.new_Accno = 00201,
    X.old_Balance - X.new_Balance > 100000
    ->
    write( "suspect withdraw at %s", X.evtime)
end AccontMonitor.
```

The lines "begin AccountMonitor" and "end AccountMonitor" delimit the module. Several rules may be defined within a module (also refereed as "monitor".) The second line in the example define the *universe* for the module, in this case any update, delete and insert on the **ACC** table will be monitored by this module. Then the rule definition comes. Basically this rule will be fired by an update on the **ACC** table, The variable *X* denotes the tuple being updated. The EPL system makes available to the programmer both the new and the old attribute values of X, these are respectively refereed by means of prefixes "new_" and "old_". An additional attribute, namely "evtime", is available. This contains the time when the specific event occurred.

In the previous rule, the event-condition list consists of one event and three conditions. The action list contains a single write statements. In general one or more actions are allowed, these actions are printing statements, execution of SQL statements, and operating system calls.

The previous rule can also be written in a form that combines an event and its qualifying conditions, that is:

```
update(ACC(X),
       X.old_Accno = 00201,
       X.new_Accno = 00201,
       X.old_Balance - X.new_balance > 100000)
->
write("suspect withdraw at %s", X.evtime)
```

In this second version, the extent of parentheses stresses the fact that conditions can be viewed as part of the qualification of an event. A basic event followed by some conditions will be called a **qualified event.**

The primitives to monitor sequences of events provided by EPL are significantly more powerful than those provided by current database systems. Thus, monitoring transfers from account 00201 to another account, say 00222, can be expressed in a EPL module as follows:

```
update(ACC(X), X.old_Accno = 00201,
       X.old_Balance - X.new_Balance > 100000)
update(ACC (Y), Y.old_Accno = 00222,
       X.old_Balance - X.new_Balance = Y.new_Balance - Y.old_Balance)
 ->
write("suspect transfer")
```

Thus, the succession of two events, one taking a sum of money larger than $100.000 out of account 00201 and the other depositing the same sum into 00222, triggers the printing of a warning message.

For this two-event rule to fire, the deposit event must *follow immediately* the withdraw event. (Using the concept of *composite events* described later, it will also be possible to specify events that need not immediately follow each other.)

The notion of *immediate following* is defined with respect to the universe of events being monitored in the module. Monitored events are arranged in a temporal sequence (a history). The notion of universe is also needed to define the negation of *b*, written *!b*, where *b* stands for a basic event pattern. An event *e1* satisfies *!b* if *e1* satisfies some basic event that is not *b*.

We now turn to the second kind of event patterns supported by EPL: **clock events**. Each clock event is viewed as occurring immediately following the event with the time-stamp closest to it. But a clock event occurring at the same time as a basic event is considered to follow that basic event. For example, say that our bank make funds available only after two days from the deposit of a check. This might be accomplished as follows:

```
insert( deposit( Y), Y.type = "check"),
clock( Y.evtime + 2D)
 ->
... action to update balance ...
write( "credit %d to account # %d", Y.amount, Y.account).
```

In this rule the "clock" event makes the rule waiting for two days after the check deposit took place.

The EPL system assumes that there is some internal representation of time, and makes available to the user a way to represent constant values expressing time. In particular, any constant number can be followed by one of the following characters: 'S', 'M', 'H', 'D', which stand, respectively, for seconds, minutes, hours and days. In the previous example the constant *2D* stands for two days. A value for time is built as the sum of functions that map days, hours, minutes and seconds to an internal representation. Thus *2D+24H+61M* will map to the same value of time as *3D+1H+1M*. Thus, EPL rules are not dependent on the internal clock representation chosen by the system. Observe that a clock can only take a constant argument —i.e., a constant, or an expression which evaluates to a constant of type time.

Patterns specifying (i) basic events, (ii) qualified events, (iii) the negations of these, and (iv) clock events are called *simple event patterns*. Simple patterns can always be decided being true or false on the basis of a single event. *Composite event patterns* are instead those that can be only satisfied by two or more events. Composite event patterns are

inductively defined as follows. Let $\{p_1, p_2, ..., p_n\}$, n>1 be event patterns (either composite or simple.) Then the following are also composite event patterns:

1. $(p_1, p_2, ..., p_n)$: a sequence consisting of p_1, immediately followed by p_2, ..., immediately followed by p_n.
2. $n{:}p_1$: a sequence of n>0 consecutive p_1's.
3. $*{:}p_1$: a sequence of zero or more consecutive p_1's.
4. $[p_1, p_2, ..., p_n]$: $(p_1, *{:}!p_2, p_2, ..., *{:}!p_n, p_n)$.
5. $\{p_1, p_2, ..., p_n\}$ denotes that at least one of the p_i must be satisfied for $(1 \le i \le n)$.

Using the composite patterns we can model complex patterns of events. For instance, we can be interested to "the first sunny day after at least three consecutive raining days." Assuming that we have a "weather" table which is updated every day (the "type" attribute contains the weather type for the specific day.) Our rule will be:

```
[ (3:insert( weather(X), X.type = 'rain')),
  insert( weather(Y), Y.type = 'sun')]
->
write("hurrah, eventually sun on %d\n", Y.evtime).
```

This rule fires even though between the three raining days and the sunny one there are days whose weather type is different from "rain". In case we are interested to "the first sunny day immediately following three or more raining days," we should rewrite the rule as:

```
(3:insert( weather(X), X.type = 'rain'),
 *:insert( weather(Z), Z.type = 'rain'),
 insert( weather(Y), Y.type = 'sun'))
->
write("hurrah, eventually sun on %d\n", Y.evtime).
```

Note the different use of the *relaxed sequence* operator "[...]" in the first case and the *immediate sequence* one "(...)" in the second rule. By combination of the available composite operators, EPL can express very complex patterns of events.

EPL ARCHITECTURE

EPL's architecture combines CLIPS with a DBMS that supports the *active* rule paradigm. The current prototype runs on Sybase [SYBASE] and Postgres [POST] — but porting to other active RDBMS [K90][MS93] is rather straightforward.

Sybase rule system presents some drawbacks like:

- Only one trigger is allowed on each table for each possible event;
- The trigger fires immediately after the data modification takes place;
- Trigger execution is set oriented, which means the triggers are executed only once regardless the number of tuples involved in the transaction;
- Only SQL statements are allowed as actions.

EPL tries to both overcome to such limitations and allow the user to model patterns of meaningful events.

Event Monitoring Mechanism

For each event which needs to be monitored by an EPL rule a trigger and a table (also referred as Event monitor Relation, ER) have to be created on Sybase. The trigger fires on the event which can be either an insert, delete or, update. This trigger will copy the modified tuple(s) into the corresponding ER.

As an example, say that we want to monitor the insertion on the **ACC** relation previously defined. As soon as the EPL monitor is loaded into the system the fact *(universe acc_mon i acc)*, will be asserted into the CLIPS working memory. This new fact triggers a CLIPS rule which creates a new Sybase relation called "ERacc_ins" having the following schema: *(int accno, int balance, int evtime)*. Moreover, a Sybase trigger is created by sending the following command from CLIPS to Sybase:

```
create trigger ETacc_ins on acc for insert as
begin
  declare @d int
  select @d = datediff( second, '16:00:00:000 12/31/1969', getdate())
  insert ERacc_ins select distinct *, @d from inserted
end
```

The event time is computed as the number of seconds since the 4pm of 12/31/1969. This is a standard way to represent time on UNIX systems. Any ER name starts with the prefix "ER" followed by both the monitored table name and the type of event. The correspondent trigger has "ET" as prefix.

As later explained, the module **EPL-Querier** performs the communication between CLIPS and Sybase.

EPL rules as finite-state machines

As previously discussed, any EPL rule is modeled by a finite state automaton which is implemented by a set of CLIPS rules. Transitions between automaton states take place when the following conditions occur:

- a new incoming event satisfies the current pattern;
- a pending clock request reaches its time;
- a predicate is satisfied.

On a transition the automaton can take one of the following actions:

- Move to the next **Acceptance** state where it will wait for the next event;
- Move to a **Fail** state. Here, the automaton instantiation, with relative information, is removed from the memory;
- Move to a **Success** state. Here, the actions specified in the right hand side of the rule are executed.

Each EPL rule is transformed to a set of CLIPS rules which implement its finite state machine. Several instantiations of the same EPL rule can be active at the same time.

Architecture overview

EPL is basically built on top of CLIPS in two ways: 1) Some new functions, implemented in C, have been added to CLIPS in order to support EPL; 2) CLIPS programs have been written to implement the EPL rule execution system. Figure 1 depicts the entire EPL system.

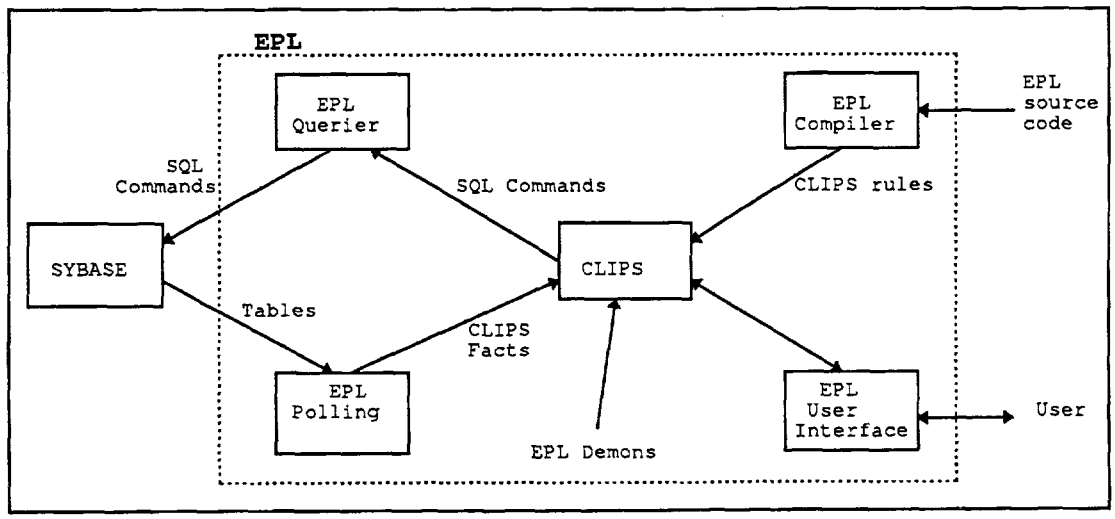


Figure 1. EPL Architecture

EPL-Compiler is a rule translator, it takes an EPL program as input and produces a set of CLIPS rules which implement the EPL program. **EPL-Polling**, at regular intervals, transfers the ERs from Sybase to the CLIPS working memory. This process requires some data type conversion in order to accommodate Sybase tuples as CLIPS facts. The **EPL-Querier** sends SQL commands to Sybase server when either (1) Sybase triggers or ERs have to be created (or removed) or when (2) an SQL command is invoked on the action side of an EPL rule. The **EPL-User-Interface** accepts commands from the user and produces output to either the screen or a file. At low level, EPL commands are executed by asserting a fact in the CLIPS working memory. Such an assertion triggers a rule which executes the desired command. This loose coupling allows an easy design of the user interface whose only task is to insert a new fact depending on the user action.

EPL-Demons is a CLIPS program which implements the EPL rule execution system. Basically this set of CLIPS rules monitors the entire EPL execution. The EPL demons, together with the CLIPS rules produced by the EPL-compiler, form the entire CLIPS program under normal execution time. The CLIPS facts on which these rules work are those periodically produced by the EPL-Polling, and those asserted by the EPL user interface as a consequence of user actions.

Conclusions

This document has described the design and the architecture of EPL, a system which provides sophisticate event pattern monitoring and triggering facilities on top of commercial active databases, such as SYBASE. EPL implementation is based on CLIPS and the design of an interface between SYBASE and CLIPS represented one of the most critical tasks in building EPL. EPL rules are translated into a set of CLIPS rules which implement the finite state machine needed to execute such EPL rules.

This paper provided an overview of the language definition and a brief description of the system implementation neglecting various implementation details for lack of space.

Future work is required to provide language extensions and interfacing with other active database systems.

Acknowledgments. This report builds upon a previous one authored by S. Lau, R. Muntz and C. Zaniolo. The authors would also like to thank Roger Barker for several discussions on EPL.

REFERENCES

[ISO] ISO-ANSI Working Draft of SQL-3.

[COLE] R. Coleman. "Pulling the Right Strings", Database Programming and Design, Sept. 1992, pp. 42-49.

[SYBASE] Sybase Inc. Sybase's Reference Manual.

[Fo82] C.L.Forgy. "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem" on Artificial Intelligence 19, 1982.

[CLIPS] "CLIPS User's Guide", Artificial Intelligence Section.

[K90] G.Koch, "Oracle: the Complete Reference," Berkeley, Osborne, McGraw-Hill, 1990.

[LMM86] J.L Lassez, M.J. Maher, K. Marriot. "Unification Revisited", Lecture Notes in Computer Science vol.306, Springer-Verlag, 1986.

[MS93] J.Melton, A.R.Simon. "Understanding the new SQL: A Complete Guide," San Mateo, California, Morgan Kaufmann Publishers, San Francisco, California, 1993.

[NT89] S.Naqvi, S.Tsur "A Logical Language for Data and Knowledge Bases," Computer Science Press, New York, 1989.

[POST] J. Rhein, G. Kemnitz, POSTGRES User Group, The POSTGRES User Manual, University of California, Berkeley.

[St87] M. Stonebraker, The POSTGRES Storage System, Proc. 1987 VLDB Conference, Brighton, England, Sept. 1987.

[St92] M. Stonebraker, The integration of rule systems and database systems IEEE Transactions on Knowledge and Data Engineering, October 1992.

CLIPS++: Embedding CLIPS into C++ *

Lance Obermeyer and Daniel P. Miranker

Tactical Simulation Division
Applied Research Laboratories
The University of Texas at Austin
Austin, TX 78713
lanceo@arlut.utexas.edu

Department of Computer Sciences
The University of Texas at Austin
Austin, TX 78712
miranker@cs.utexas.edu

Abstract

This paper describes a set of C++ extensions to the CLIPS language and their embodyment in CLIPS++. These extensions and the implementation approach of CLIPS++ provide a new level of embeddability with C and C++. These extensions are a C++ include statement and a defcontainer construct; `(include <c++-header-file.h>)` and `(defcontainer <c++-type>)`.

The include construct allows C++ functions to be embedded in both the LHS and RHS of CLIPS rules. The header file in an include construct is the same header file the programmer uses for his/her own C++ code, independent of CLIPS. The defcontainer contruct allows the inference engine to treat C++ class instances as CLIPS deftemplate facts. Consequently, existing C++ class libraries may be transparently imported into CLIPS. These C++ types may use advanced features like inheritance, virtual functions, and templates.

The implementation has been tested with several class libraries, including Rogue Wave Software's Tools.h++, GNU's libg++, and USL's C++ Standard Components. The execution speed of CLIPS++ has been determined to be 5 to 700 times the execution speed of CLIPS 6.0 (10 to 20 x typical).

1 Introduction

CLIPS++ is a reimplementation of NASA's CLIPS 6.0 [3] that has been tailored to support applications with large data and performance requirements or applications that must coexist with C++. This reimplementation has the following features

- Rules may directly access C++ objects. No need to reformat C++ objects to CLIPS representations or vica versa[1].
- Simple integration with existing C++ code.
- Compatible with C++ development tools.
- Execution time is reduced from 5 to 700 times (10 to 20x typical).
- Scalabile with respect to data and throughput requirements. See Section 3.2.
- Matching technology that eliminates the problems of volatile match time; resolving a critical problem for real-time applications.

CLIPS++ is compatible with NASA CLIPS 6.0 except that the COOL object system has been replaced with C++ objects, and only a single LEX-like conflict resolution strategy is supported. Nearly all publically available CLIPS programs available on the Internet have been compiled and correctly executed by CLIPS++.

2 Language

The CLIPS++ system includes minor language extensions that allow CLIPS rules to operate transparently on C++ object instances. These extensions comprise just two new constructs and a semantic extension to the use of deftemplate.

*This effort was funded in part by the State of Texas Advanced Technology Program, the Applied Research Laboratories Internal Research and Development Program DABT63-92-0042.

[1] CLIPS++ is a true integration with C++, not a simple wrapper like RETE++

Data Size	CLIPS 6.0	CLIPS++	Speed-Up
16	3	<1	>3.00
32	64	<1	>64.00
64	271	9 3	906.33
128	n.a	12	
256	n.a	66	

Table 1: Execution Time of Manners, (seconds), CLIPS 6.0 vs. CLIPS++

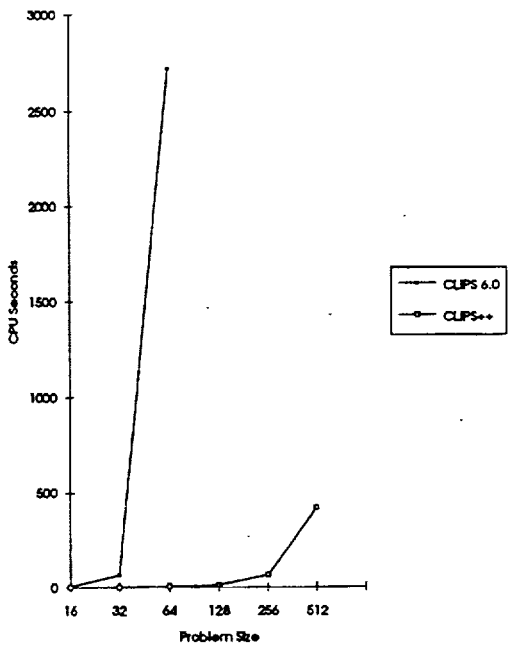


Figure 1: Relative Performance of Manners, CLIPS 6.0 vs. CLIPS++

2.1 Declarations

2.1.1 include

```
; salary.clp
(include "decls.h")
```

The include construct is equivalent to a C/C++ #include. It makes all declarations in a legal C/C++ header file visible to CLIPS++. For example, assume the C++ header file decls.h declares a type `employee_type`. That file may in turn include definitions from third party class libraries. The Rogue Wave string and date classes are used in the running example in this paper.

Consequently, listing the include statement (above) at the beginning of a CLIPS++ source file makes declarations in the file decls.h, including `employee_type`, visible to the CLIPS++ program.

```
#include<rw/cstring.h>
#include<rw/rwdate.h>


class   last_raise_type {
public:
        RWDate          date;
};

class   employee_type   {
public:
        RWCString       name;
        RWCString&      get_department(void)
                                {return dept;}
        last_raise_type last_raise;
private:
        RWCString       dept;
};
```

2.1.2 defcontainer

```
; salary.clp
(defcontainer employee (type employee_type))
```

The defcontainer construct is the primary language addition. Defcontainer is equivalent to the CLIPS deftemplate, except that it is used to declare that a C++ type can be referenced in a rule's LHS. Note that the slots of a defcontainer are precisely the slots of the C++ object defined in the include file. Thus, the arguments to defcontainer are limited to the containers name and the C++ type that is stored in that container.

Any C++ type can be used provided that the type overloads the operators == and > in the obvious way. Advanced C++ features like inheritance, multiple inheritance, virtual functions, and templates may be used. Most notably the C++ data type need not inherit from a CLIPS++ provided base class. This allows application developers great flexibility in designing class hierarchies, and reusing existing code.

2.2 Rule Syntax

There is no real change between CLIPS++ rule syntax and CLIPS rule syntax. There is a semantic extension to the meaning of the slot names for unordered facts. When the compiler recognizes that a template name is C++ container name, declared by defcontainer, rather than a template name declared by deftemplate, then the slot identifiers are allowed to be any legal C++ expression that returns a value from an object. The expressions may include the C++ dot (.) and dereference (-¿) operators. As a result LHS's can be formulated to traverse complex object structures. For example, the following is a legal CLIPS++ rule that operates on the class defined above. The compiler recognizes **employee** as the name of a container of instances of the C++ class **employee_type**. The slot defined by the accessor function **(get_department())** returns values from the instances. The slot defined by the expression **(last_raise.date)** returns a value from a nested object instance.

```
; give a raise to everyone in r&d who has not
; had a raise since the beginning of 1994
(defrule give-a-raise-to-r&d
  ?e <- (employee
           (get_department() "r&d")
           (name ?name))
           (last_raise.date ?d
                 &:(< ?d (RWDate 1 1 1994)))
  =>
...
```

In the RHS, member functions may be called for objects that are matched in the LHS. In both the RHS and LHS, arbitrary C/C++ objects may be accessed or called.

The tight integration with C++ has performance benefits as well. C/C++ functions are directly called. CLIPS introduces a level of indirection by activating C functions through a name to address mapping scheme.

3 Architecture and Environment

3.1 Development Environment

The CLIPS++ system is based on an optimizing compiler that accepts CLIPS or CLIPS++ programs as input and outputs C++. Output code is compiled by the host system's C++ compiler, and linked with a runtime library of CLIPS++ support routines.

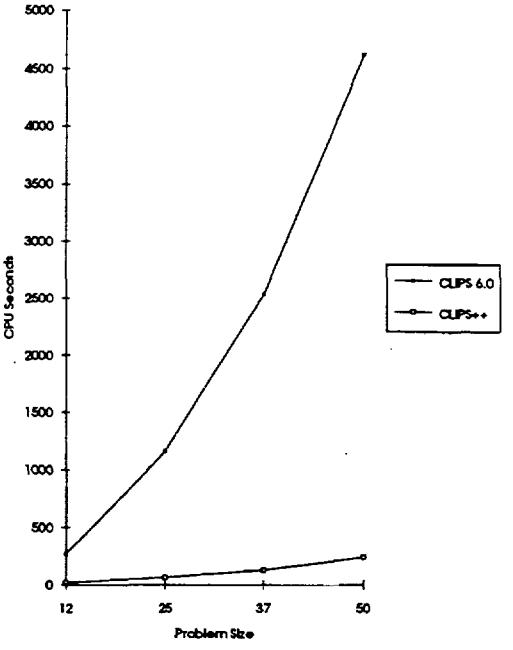


Figure 2: Relative Performance of Waltz, CLIPS 6.0 vs. CLIPS++

The system includes a debugger capable of monitoring system execution and inspecting data, and a profiler capable of guiding the user through the program optimization sequence.

3.2 Performance and Matching Technology

The CLIPS++ system employs both published and unpublished optimization techniques developed over the last 12 years[7, 6, 4, 1, 5, 2]. The CLIPS++ system features the LEAPS matching technique asymptotically better than RETE or TREAT. Consequently the performance of CLIPS++ scales with problem size

data size	CLIPS 6.0	CLIPS ++	Speed-Up
12	269	19	14.16
25	116	1 63	18.43
37	253	6 130	19.51
50	462	0 240	19.25

Table 2: Execution Time of Waltz, (seconds), CLIPS 6.0 vs. CLIPS++

as measured by data and throughput requirements[2].

Rather than computing an entire conflict set and then applying a conflict resolution strategy to determine a single rule instantiation to fire, LEAPS folds the conflict resolution strategy into the matcher such that the first instantiation that it discovers on each cycle is the same instantiation that a RETE or TREAT implementation would fire. LEAPS has been formally proven to produce the same execution sequences as the RETE match.

Real-time applications benefit substantially. Since it is much faster, and more predictable to compute only the fired instantiation, CLIPS++ eliminates much of the volatility in match times developers have come to expect from rule systems. The combination of the improved algorithm and the optimization techniques often result in provably optimal code.

3.3 Integration with C++ Class Libraries

Integration with C++ class libraries is a simple matter of including the correct header files and linking with the correct libraries. CLIPS++ can inference over class library objects if they are declared using a defcontainer construct. CLIPS++ can call library functions wherever a standard CLIPS function would be used.

In the above example, the line

```
(last_raise.date ?d &:(< ?d (RWDate 1 1
1994)))
```

uses RWDate objects, which are the date objects from Rogue Wave's Tools.h++ class library. The statement binds an object of type RWDate to the variable ?d, constructs a temporary object with the date 1/1/94, and compares the bound object with the temporary object. The comparison will automatically call the Rogue Wave supplied function

```
operator <(const RWDate& d1, const RWDate&
d2)
```

Nearly all of the complexities of C++ class libraries are hidden from the programmer.

The CLIPS++ system has been tested with several class libraries, including the Tools.h++ library from Rogue Wave Software, the C++ Standard Components from USL, and the libg++ library from GNU.

[2]Nearly all claims of scalable performance are based on increasing the size of the rule base, *not* by increasing the size of the working memory

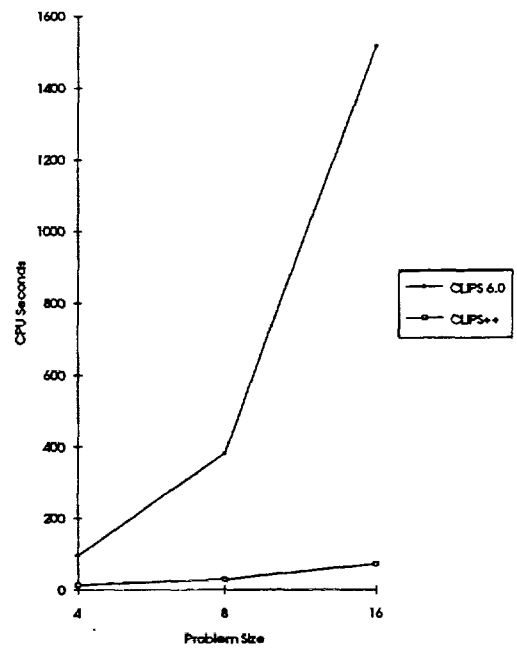


Figure 3: Relative Performance of Waltzdb, CLIPS 6.0 vs. CLIPS++

4 Benchmark Results

We detail the performance of 4 of the 5 programs in the Texas Benchmark suite, Waltz, Waltzdb, Manners and ARP[1][3] Performance results are for CLIPS++ vs. CLIPS 6.0 on standard benchmark programs. All times reported are user + system cpu seconds. The test platform was a Sun Sparcstation 2 running SunOS 4.1.3. Both the test programs and the baseline CLIPS 6.0 were compiled using GNU's gcc version 2.5.8, with highest optimization.

We clearly demonstrate both absolute improvements in speed and very substantial improvement in scalability with respect to data size size. The asymptotic improvement due to the LEAPS match reveals important speed improvement for small data set sizes in the range of 3 to 7. As data set sizes increase, increases in speed are measured by orders of magnitude (i.e. 10x, 100x even 1000x).

[3]Available by ftp from anonymous@cs.utexas.edu, connect to pub/ops5-benchmark-suite. CLIPS versions are also there.

data size	CLIPS 6.0	CLIPS++	Speed-Up
4	95	13	7.31
8	38	1 29	13.14
16	151	9 72	21.10

Table 3: Execution Time of Waltzdb, (seconds), CLIPS 6.0 vs. CLIPS++

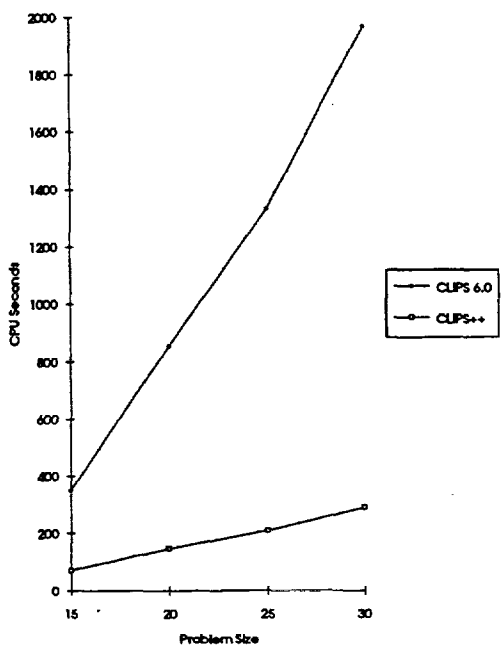


Figure 4: Relative Performance of ARP, CLIPS 6.0 vs. CLIPS++

Data size	CLIPS 6.0	CLIPS++	Speed-Up
15	349	71	4.92
20	853	146	5.84
25	1332	209	6.37
30	1967	289	6.81

Table 4: Execution Time of ARP, (seconds), CLIPS 6.0 vs. CLIPS++

5 Conclusion

The CLIPS++ system is an advanced production system that successfully integrates declarative CLIPS rules with object oriented C++ data types. This integration extends from simple user defined types to complicated class libraries from commercial vendors.

Additionally, the CLIPS++ system is based on the LEAPS algorithm, and contains many published and unpublished performance optimizations. The combination of the asymptotically superior LEAPS algorithm and the optimizations results in a production system of unprecedented performance.

References

[1] D.A. Brant, T. Grose, B. Lofaso, and D.P. Miranker. Effects of Database Size on Rule System Performance:Five Case Studies. In *Proceedings of the 17th International Conference on Very Large Data Bases (VLDB)*, 1991.

[2] D.A Brant and D.P. Miranker. Index Support for Rule Activation. In *Proceedings of the 1993 ACM SIGMOD International Conference on the Management of Data*, 1993.

[3] J.C. Giarratano. *CLIPS User's Guide, Version 6.0.* Artificial Intelligence Section, Lyndon B. Johnson Space Center, 1994.

[4] D. P. Miranker, D. Brant, B.J. Lofaso, and D. Gadbois. On the Performance of Lazy Matching in Production Systems. In *Proceedings of the 1990 National Conference on Artificial Intelligence*, pages 685–692. AAAI, July 1990.

[5] D. P. Miranker, F.H. Burke, J. J. Steele, J. Kolts, and D. R. Haug. The C++ Embeddable Rule System. *Int. Journal on Artificial Intelligence Tools*, 2(1):33–46, 1993. Also in the Proc. of the 1991 Int. Conf. on Tools for Artificial Intelligence.

[6] D. P. Miranker, B.J. Lofaso, G. Farmer, A. Chandra, and D. Brant. On a TREAT Based Production System Compiler. In *Proceedings of the 10th International Conference on Expert System, Avignon, France*, pages 617–630, June 1990.

[7] D.P. Miranker and B. J. Lofaso. The Organization and Performance of a TREAT Based Production System Compiler. *IEEE Trans. on Knowledge and Data Engineering*, pages 3–10, March 1991.

N95-19631

EXPERT SYSTEM SHELL TO REASON ON LARGE AMOUNTS OF DATA

G. Giuffrida
Computer Science Department
The University of California Los Angeles
giovanni@cs.ucla.edu

Abstract

The current DBMSs do not provide a sophisticated environment to develop rule based expert system applications. Some of the new DBMSs come along with some sort of rule mechanism, these are *active* and *deductive* database systems. However, both of these are not featured enough to support full implementation based on rules. On the other hand, current expert system shells do not provide any link with external databases. That is, all the data are kept into the system working memory. Such working memory is maintained in main memory. For some applications the reduced size of the available working memory could represent a constraint for the development. Typically these are applications which require reasoning on huge amounts of data. All these data do not fit into the computer main memory. Moreover, in some cases these data can be already available in some database systems and continuously updated while the expert system is running.
This paper proposes an architecture which employs knowledge discovering techniques to reduce the amount of data to be stored in the main memory, in this architecture a standard DBMS is coupled with a rule-based language. The data are stored into the DBMS. An interface between the two systems is responsible for inducing knowledge from the set of relations. Such induced knowledge is then transferred to the rule-based language working memory.

INTRODUCTION

Current trends in database research area are about making smarter and more friendly current DBMS. Traditionally, a DBMS is mostly a repository of information to be later retrieved by an ad-hoc query language. If some data are currently unavailable in the system, or the query is not properly issued, the system can return either an error or a non-answer. This is not really the case when a specific question is asked to a human expert, in this case the expert digs in his or her memory looking for an answer, if no proper answer is found the expert perhaps starts looking for alternative, and approximate, responses. Actually, the expert uses his or her knowledge in a sort of "cooperative way" trying to make the user as more satisfied as possible. To answer such a query the expert employs his or her knowledge in a much more productive way than performing a simple linear scan over the known set of tuples in his or her mind: **meta-knowledge** was fruitfully employed in order to formulate the answer.

In an artificial system such meta-knowledge can be known a priori (read: Tied up to the system at database design time) or induced by an already available set of relations. The purpose of *induced knowledge* [Qu79] [CLC91] is to describe the current state of the database by means of sets of rules; a rule induction algorithm is responsible to *discover* such induced knowledge. Basically, induced rules are oriented to summarize the way the different attributes are related to each other. Artificial reasoning can be built on top of this meta-knowledge.

This paper proposes an architecture which integrates two public domain systems, namely Postgres and CLIPS, into a unique environment. The proposed architecture offers a shell for developing expert systems able to work on large amounts of data. Database relations are stored on Postgres while the artificial expert is implemented on top of CLIPS. The artificial expert can specialize the rule induction algorithm in order to focus the induction only on the necessary data.

Postgres comes along with a rule system to implement active database features. By the way, implementing an expert system using only Postgres has some drawbacks. The Postgres rule mechanism has been designed for triggering over updates on the database. We always need to perform a secondary storage update in order to fire a Postgres rule. This can be too expensive in many cases when a little fact stored in main memory can perform the same task. Moreover, the rule system in Postgres does not implement all the facilities of a special purpose rule-based language such CLIPS.

On the other side CLIPS does not provide any access to an external database. This can cause serious limitations in designing expert systems reasoning on large amounts of data stored on a database.

CURRENT SYSTEMS

Latest developments in databases are oriented to enrich traditional systems with some sort of rule-based mechanism. Novel additional features are so added to traditional DBMS by means of rules. New classes of systems have appeared on the market, they are: *Active* [St92] [HCL+90] [MD92] and *Deductive* [Mi88] [RSS92] [NT89] databases. The following sections give some details about these new systems. The suitability of those as expert system shell is also investigated. Advantages and drawbacks for each system are drawn. Also, the limitations of current rule-based languages as developing shell for expert system oriented to reason on large amounts of data are discussed.

Active Database

More and more active database systems are everyday appearing on the market. The transition of such system from research stages to the market was fast. They materialized in systems like Sybase [MD92], Postgres [St87], Starburst [HCL+90], and more. However, the active feature of databases is still under investigation. Additional research still needs to be done.

The active feature extends the traditional systems with a rule-based mechanism oriented to allow implementation of *demons*. The purpose of these demons is basically to overview operations on the database and, when certain conditions are satisfied, invoke a corrective action over the database. The system can so be extended with sort of animated entities; data integrity can be enforced by means of active rules.

Typically an active rule has the following format:

```
on <event>
if <condition>
then <action>
```

This rule model is also referred as E-C-A (Event-Condition-Action) rule. Simpler rule model can utilize an E-A pattern. **event** is one of the possible operations oriented to tuple handling;

typical events are: *insert*, *delete*, *update* or, *retrieve*. The specified **condition** is a condition which should hold on the data involved in the current event in order to fire the rule. **action** is the set of actions that take place when the specified condition is satisfied.

In the following a Postgres rule is reported as example of active rule (copied from [Post]):

```
define rule r1 is
  on replace to EMP.salary
     where current.name = 'Joe'
  do replace EMP (salary = new.salary)
     where EMP.name = 'Sam'
```

Rule r1 will fire every time the salary of Joe is updated. For such update the salary of Sam will be set to the same value of the new salary of Joe. In other words, Sam will always get the same salary of Joe. Note that the inverse is not true, the same rule will not take care to update Joe's salary when Sam's salary is replaced by some new value.

Evaluation of database active feature is based upon (1) the domains of possible events which can be monitored; (2) the formalism to specify the condition; (3) the language to execute actions. Other features, like rule execution model, have also to be considered for a sound evaluation. Postgres [St87], is actually one of the most advanced active databases currently available.

Active database systems present some drawbacks as expert system shells. The most significant ones are hereafter listed:

- Firing only upon database events, no "Working memory" is available. All the data are kept in the secondary memory storage. Main memory usage might, in several cases, be preferable to a secondary memory one.
- The language to specify actions is usually restricted to the database domain. Typical rule-based languages allow on the right-hand side complex operations (print, while loop, user inputs, etc.) to be performed.
- Not well defined rule execution model is available.

Other criticisms to these systems are about the rule redundancy required in some cases. A set of sort of complementary rules might be defined in order to ensure some data integrity. For instance, let us suppose we want to make sure that each employee works for one and only one department. A set of rules needs to be defined in order to enforce this constraint: One will take care when a new employee is hired, another when the employee is transferred, another when a department is dropped, and so on. This is not really the case of an equivalent rule-based implementation where a single rule can handle all the cases. The different behavior between active DBMS and RBL is actually based on different architectures. In an active database system the rules are tied up to the database operations, so a trigger must be defined for each possible operation on the interested data. In a rule-based system, rules are fired straight from the data in the working memory. The operations which update the data are hidden to the triggered rules. This model reduces the number of required rules. Moreover, the maintenance cost for a set of rules is of course higher than the one for a single rule.

The previous considerations should convince the reader that implementing an expert system on top of an active database is not a straightforward task; they lack the power for that purpose.

Deductive Databases

Another challenging and relatively new research direction in database area is about *Deductive databases*. These systems are essentially based on a Prolog-like development language. Some of these are LDL++ [NT89], NAIL! [MUVG86] and CORAL [RSS92]. Here facts can be either stored on main memory or secondary memory storage. However, in the author's opinion, deductive databases are still not at a stable stage. Further investigation needs to be spent on those. This belief is also enforced from the lack of any commercial deductive database system on the market at the time of this writing.

Development of deductive databases raised several both theoretical and practical issues that are still looking for answers. Research is still in progress in this direction.

Some of the most significant points of deductive systems are:

- Prolog-like rule-based development environment;
- Recursive rule invocation allowed;
- Easily definable virtual relations (*views* in database jargon);
- Hybrid forward/backward chaining rule execution model. The method chosen depends on the current task to be performed. It is chosen automatically by the system itself. The user does not have any responsibility on that;
- Restricted usage of negation. Only certain classes of negation (namely *stratified*) are allowed. Basically, no negation on recursive calls can be used.
- Ability of interfacing with already available DBMSs.

Deductive databases are definitively a proper environment for developing expert reasoning on large amounts of knowledge. However, as already said, they still need some time before reaching a steady stage.

Expert System Shells

Somewhat current expert systems shells present the opposite problem of the previously discussed database systems. In this case all the facts are kept into the system working memory. No connection with any external storage system is provided. This can be a constraint difficult to be taken around when huge amount of knowledge is employed as basis for the reasoning. In this case the main memory cannot be big enough to accommodate this large number of tuples. In another possible scenario, reasoning has to be developed on data which are already stored on some database systems. Perhaps these data are still evolving while the artificial reasoning is performed. The only possible solution to this problem is represented by a periodic transfer of the new updated data from the database to the expert system shell.

Current solutions to this problem are oriented to create an ad-hoc interface between an expert system shell and a DBMS. Such solutions are often properly tailored for the application itself. They do not provide a generic solution to the problem.

KNOWLEDGE INDUCTION ALGORITHM

The process of knowledge induction [Qu79] [CL90] aims to build useful semantic data starting from the available relations in a database. In this section, an overview of the knowledge induction algorithm is given while keeping an eye open on our specific implementation.

Somehow the knowledge induction process aims to "capture" the semantics of the stored data and model it by means of *induced rules*. Different forms of rule may exist, we are mainly interested in rules which correlate two attributes between themselves. The correlation of interest for us is the definition of a domain for the first attribute which identifies a unique value for the second attribute. We are interested to the *range form* which is: "*if* $x_1 \leq X \leq x_2$ *then* $Y = y$ *with Probability P and Coverage C.*" The **probability** value expresses the *certainty degree* of the given rule while the **coverage** refers to the number of tuples used to build such rule. Additional details can be found in [Qu79], [CL90] and [CLC91]. Several different forms and attributes can be combined to express induced rules.

The set of induced rules represents part of the knowledge about the given databases. Storing these rules most probably requires less space than the corresponding table format. Induced rules have to change dynamically reflecting the database updates. Every update operation on the database has to affect the set of induced rules accordingly.

In the proposed system induced rules are stored as CLIPS facts whose form is:

```
(ik <arg1> <lowerBound> <upperBound> <arg2> <value> <prob> <cover>)
```

The first atom is to classify the set of facts describing the induced knowledge. The remaining atoms are the information about the specific induced rule. The set of all these facts forms the induced knowledge. Semantics of most of the arguments is straightforward. **prob** is the probability factor and **cover** is the number of tuples covered by the rule (look at [Ch94] for a complete description of these parameters.) For instance, the rule "*if 62000 ≤ SALARY ≤ 85000 then TYPE = 'MANAGER' with Probability=0.6, Coverage=3*" is stored in the following CLIPS fact:

```
(ik SALARY 62000 85000 TYPE MANAGER 0.6 3)
```

As an example, let us suppose to have the following table, namely *emp*, in our Postgres database:

EMP

NAME	TYPE	SALARY
John	STAFF	30000
Mary	MANAGER	62000
Eva	STAFF	32500
Bob	MANAGER	85000
Mark	MANAGER	76000
Judy	MANAGER	83000
Kia	STAFF	56000
Tom	MANAGER	50000
Phil	STAFF	54000

By applying the induction algorithm on these tuples the set of CLIPS facts will be:

```
(ik emp.SALARY 62000 85000 emp.TYPE MANAGER 0.8 4)
(ik emp.SALARY 50000 50000 emp.TYPE MANAGER 0.2 1)
(ik emp.SALARY 30000 32500 emp.TYPE STAFF 0.5 2)
(ik emp.SALARY 54000 56000 emp.TYPE STAFF 0.5 2)
```

The induced rule schema is only a first attempt which is used more as framework for this paper. A real design should take into more consideration other parameters to be included in the schema.

The induced rules need to be dynamically updated reflecting operations performed over the database. For instance, let us suppose the following new tuple is inserted into the system:

```
Betty  MANAGER  60000
```

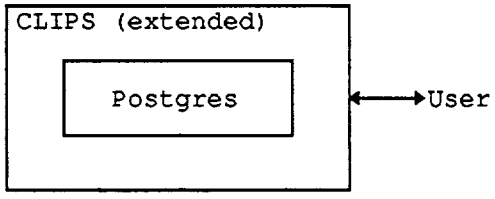
The knowledge base has to be updated to the following set of facts:

```
(ik emp.SALARY 60000 85000 emp.TYPE MANAGER 0.83 5)
(ik emp.SALARY 50000 50000 emp.TYPE MANAGER 0.16 1)
(ik emp.SALARY 30000 32500 emp.TYPE STAFF 0.5 2)
(ik emp.SALARY 54000 56000 emp.TYPE STAFF 0.5 2)
```

The new tuple affected the values in the first fact, the lower bound, probability and coverage values changed accordingly. Similar changes will take place in case of deletion or updating.

PROPOSED SYSTEM OVERVIEW

In the proposed system architecture, Postgres and CLIPS are being integrated. The interface between them represents the key point for the design. An expert system can be developed on top of CLIPS. This latter is extended with features to make available summarized data stored in Postgres. From an expert system shell's point of view the new system can be seen as in the following figure:



Postgres presence is actually hidden to the end-user. The application built on top of CLIPS takes advantage of the meta-knowledge induced from the stored database. The expert system being developed on CLIPS has to be aware of the following:

- Database schema;
- Database statistics: number of tuples, number of induced rules, rule coverage, popularity, probability, etc.

CLIPS has to be extended with dedicated constructs to inquire for such information. Depending on both the database information and the application being implemented, the meta-knowledge extraction process is properly tailored and executed. From CLIPS several actions can be taken in order to model the induction algorithm to best fit the current application.

The knowledge induction process executes in a dynamic fashion, that is, once the rule schema has been defined, any update on the monitored Postgres relations will be reflected on the induced rules. The rule induction manager then propagates these induced rules to the CLIPS working memory in the fact form previously discussed. Eventually, these facts trigger some CLIPS rules.

The interface between CLIPS and Postgres then can be defined by the following set of functions available on the CLIPS side:

- Inquire for the database schema;
- Inquire for database statistics;
- New rule induction schema definition: $R_1.X \rightarrow R_2.Y$, where R_1 and R_2 are relations while X and Y are attributes;
- Modify rule induction parameters (cut-off factor, etc.);
- Retrieve current rule induction schema;
- Remove induced rule;

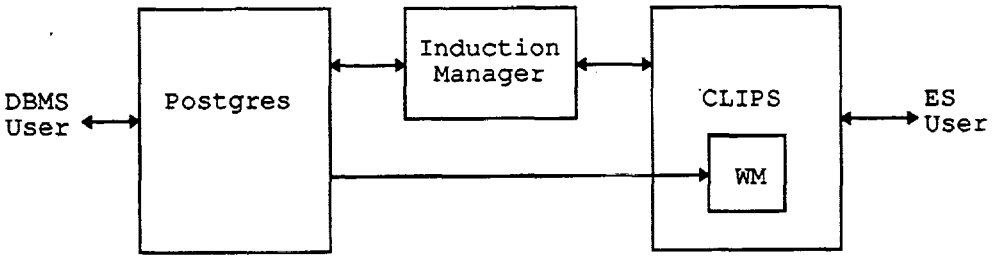
The system extends the set of built-in CLIPS functions to include these new services.

DESIGN

The system will be presented to the expert system developer as a more featured shell which includes ability to access summarized data. The key points of all design are essentially the following:

- Interface between CLIPS and Postgres;
- Induced knowledge management.

The system architecture is the following:



The *Induction Manager* is responsible for the communication between CLIPS and Postgres. CLIPS utilizes services offered by this module to tailor the induction algorithm to its needs. Responses from Postgres to CLIPS are also passed through the induction manager. The link between Postgres and WM (the CLIPS working memory) stands for a straight access from Postgres to CLIPS working memory. That is, once the rule schema has been defined, updates on Postgres will result in updates on the induced rules. These rules are then written straight to CLIPS' working memory without requiring the intervention of the induction manager. In other words, the induction manager can be seen as a set of function to be invoked from CLIPS, while

the link from Postgres to WM represents an asynchronous flow of data toward CLIPS. Once the rule induction schema has been defined, this flow of data will be dynamically maintained, that is, any update to any monitored Postgres relation can cause some rule induction instances updates, these will be promptly propagated to CLIPS through this link. As better explained in the following, actually the induction manager is scattered between Postgres and CLIPS. Some Postgres triggers and CLIPS rules (together with some external Postgres procedures) form the complete block.

The entire system works in a dynamic fashion. That is, at the same time the CLIPS application is running people can keep on working on Postgres. Postgres updates affect the set of induced rules on the CLIPS side. Such updates can trigger some CLIPS rules to execution. Under this light the entire system can be seen as an extension of the rule-based mechanism of Postgres: Some Postgres rule activations eventually trigger some CLIPS rules.

CONCLUSIONS

This paper discussed the design of a system where expert shell techniques are combined together with knowledge discovering techniques. Two public domain systems, namely CLIPS and Postgres, have been combined into a unique one. Purpose of this design is to provide an expert system shell with ability to reason over a large amount of data. Current expert system shells lack the ability of accessing data stored on external devices. On the other side database technology is not yet well refined to provide a featuring shell to develop expert systems. Proper combinations of these techniques may lead to interesting results.

The possible applications for such a system are those where reasoning on large volume of data are required. For instance, think about the complexity of reasoning on the millions of customers of a phone company or a frequent flyer program, in this case the complexity is due to the enormous amount of data to reason on.

REFERENCES

[Ch94] W.W.Chu, "Class notes CS244-Spring 1994", University of California in Los Angeles, 1994.

[CL90] W.W.Chu, R.Lee, "Semantic Query Processing Via Database Restructuring," Proceedings from the 8th International Congress of Cybernetics and Systems, 1990.

[CLC91] W.W.Chu, R.Lee, Q.Chen, "Using Type Inference and Induced Rules to Provide Intensional Answers," Proceedings of the 7th International Conference on Data Engineering, 1991.

[Fo82] C.L.Forgy, "RETE: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence 19, 1982.

[Gi89] J.C.Giarratano, "CLIPS User's guide," Artificial Intelligence Section, Lyndon B. Johnson Space Center, June 1989.

[HCL+90] L.Haas, W.Chang, G.M.Lohman, J.McPherson, P.F.Wilms, G.Lapis, B.Lindsay, H.Pirahesh, M.Carey, E.Shekita, "Starburst midflight: as the dust clears," IEEE Transactions on Knowledge and Data Engineering, March 1990.

[MD92] D.McGoveran, C.J.Date, "A Guide to SYBASE and SQL Server," Addison-Wesley Publishing Company, 1992.

[Mi88] J.Minker "Foundation of Deductive Databases and Logic Programming," Morgan-Kaufmann, Los Altos, CA, 1988.

[MUVG86] K.Morris, J.Ullman, A.Van Gelder, "Design overview of the NAIL! system," proceedings of the 3rd Int. Conference on Logic Programming, Springer-Verlag LNCS 225, New York, 1986.

[NT89] S.Naqvi, S.Tsur, "A Logical Language for Data and Knowledge Bases," Computer Science Press, 1989.

[RSS92] R.Ramakrishan, D.Srivastava, S.Sudarshan, "CORAL: A Deductive Database Programming Language," Proc. VLDB '92 Int. Conf. 1992.

[Post] J. Rhein, G. Kemnitz, POSTGRES User Group, The POSTGRES User Manual, University of California, Berkeley.

[St87] M. Stonebraker, "The POSTGRES Storage System," Proc. 1987 VLDB Conference, Brighton, England, Sept. 1987.

[St92] M. Stonebraker, "The Integration of Rule Systems and Database Systems," IEEE Transactions on Knowledge and Data Engineering, October 1992.

[Qu79] Quinlian, J.R., "Induction Over Large Data Bases," STAN-CS-79-739, Stanford University, 1979.

OMIT

Session 2A: Automation, Process Control, and Advisory Applications

Session Chair: A. Chandrasekaran

AI & Workflow Automation: The Prototype Electronic Purchase Request System

Michael M. Compton
compton@ptolemy.arc.nasa.gov
(415) 604-6776

Shawn R. Wolfe
shawn@ptolemy.arc.nasa.gov
(415) 604-4760

AI Research Branch / Recom Technologies, Inc.
NASA Ames Research Center
Moffett Field, CA 94035-1000

Abstract:

Automating "paper" workflow processes with electronic forms and email can dramatically improve the efficiency of those processes. However, applications that involve complex forms that are used for a variety of purposes or that require numerous and varied approvals often require additional software tools to ensure that 1) the electronic form is correctly and completely filled out, and 2) the form is routed to the proper individuals and organizations for approval. The Prototype Electronic Purchase Request (PEPR) system, which has been in pilot use at NASA Ames Research Center since December 1993, seamlessly links a commercial electronic forms package and a CLIPS-based knowledge system that first ensures that electronic forms are correct and complete, and then generates an "electronic routing slip" that is used to route the form to the people who must sign it. The PEPR validation module is context-sensitive, and can apply different validation rules at each step in the approval process. The PEPR system is form-independent, and has been applied to several different types of forms. The system employs a version of CLIPS that has been extended to support AppleScript, a recently-released scripting language for the Macintosh. This "scriptability" provides both a transparent, flexible interface between the two programs and a means by which a single copy of the knowledge base can be utilized by numerous remote users.

Introduction

The Procurement Division at NASA Ames Research Center processes up to twenty thousand purchase requests (PRs) every year. These PRs, which all use a common form, are used to procure virtually anything used at the Center: computers, hazardous chemicals, office equipment, scientific instruments, airplane parts, and even funding for external research projects. PRs can be submitted by any civil servant employee at the Center, and must be approved by anywhere from three to twenty different individuals and offices. The average time required to submit a PR and obtain the necessary approvers' signatures is eighteen business days. Worse yet, approximately half of the PRs that are submitted are either incorrectly filled out, lack some required additional paperwork, or are routed to the wrong group for approval and must be returned to the originator. This not only delays procurement of the requested items but also burdens the system with a significant amount of paper flowing in the "wrong direction". In addition, the paper system lacks any mechanism for tracking a submitted PR, so people who originate these purchase requests often try to track them manually by picking up the telephone and calling around until they find where the PR is in the approval process. This, along with the numerous "walk-though" PRs, contribute significantly to the delays involved in processing the requests.

In 1991, the AI Research Branch at NASA Ames undertook a "weekends and evenings" effort to see whether a knowledge-based system, in conjunction with other advanced computing tools, could help expedite the process by which purchase requests are submitted, routed, and approved. The resulting system, called the Prototype Electronic Purchase Request system (PEPR), combines a commercial electronic forms package with a knowledge-based system that both ensures that the

submitted forms are correct and complete, and generates an electronic "routing slip", based on the content of various fields on the form, that reflects the approvals that particular PR requires. The PEPR system currently operates in a Macintosh environment and takes advantage of several new collaborative features of the latest release of the Macintosh OS, including digital signatures and "OS-level" electronic mail.

The system is now being used by several different groups at Ames to process a particular class of PR, namely those that apply to the funding of external research at colleges and universities. Initial results indicate that the system can dramatically reduce the time required to originate and process PRs and their supporting paperwork by ensuring that PRs entering the system are error-free and automatically routing them to the proper individuals. The system also provides a tracking capability by which the originator of a PR can query the system about the status of a particular PR and find out exactly where it is in the approval process.

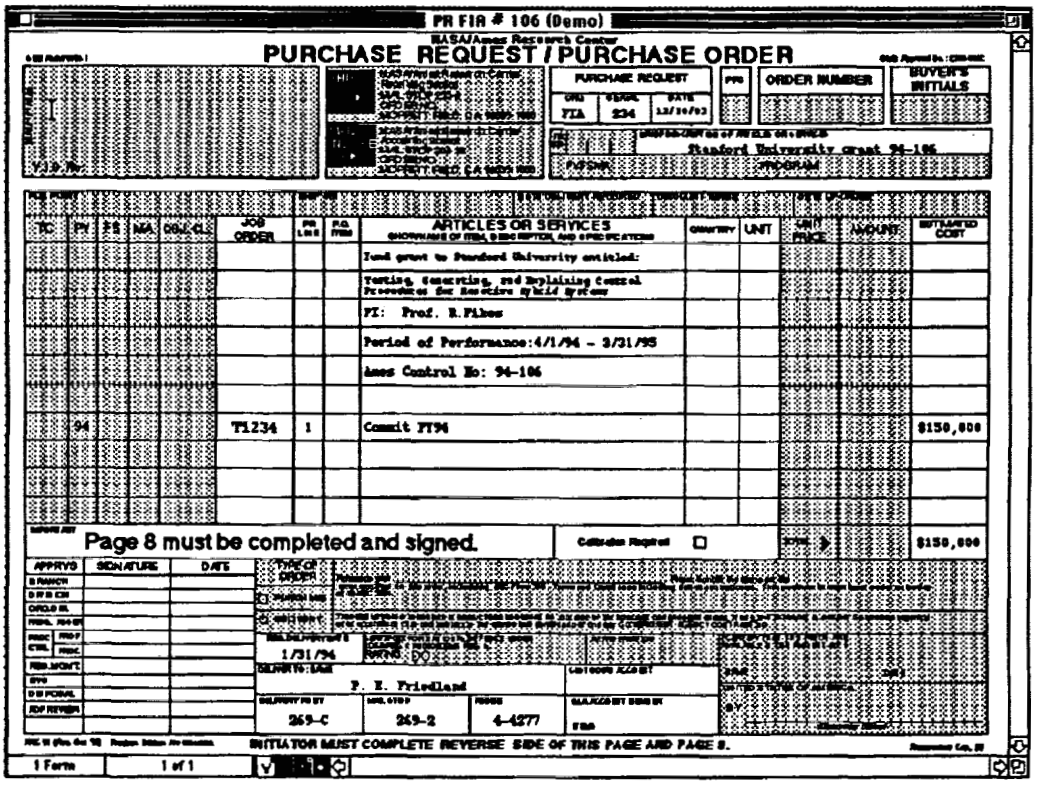


Figure 1: An Example Purchase Request

Figure 1 shows an example PR. Because of its size and complexity, we have focused on the Ames Purchase Request form for the development of the PEPR system. However, our implementation is largely form-independent, and can be applied to other forms that require "complex routing". We have also applied the PEPR system to approximately six other types of forms and are actively pursuing other potential applications of the system both inside and outside of NASA Ames.

Why AI?

The knowledge-based component of the PEPR system utilizes a fairly straightforward rule- and object-based mechanism to provide its validation and routing capabilities (although we are

investigating machine learning techniques to ease the knowledge-acquisition problem -- see the section entitled Future Plans, below). There are two main reasons that a knowledge-based approach is appropriate to the problem of validation and routing of electronic forms.

First, the knowledge required to ensure the PR's correctness and completeness is quite diverse and very widely distributed among the various groups at Ames. Different validation "rules" come into play depending on what items or services are being ordered and what offices are required to approve a particular purchase. Early on in the project it became clear that these validation rules would have to be acquired and revised incrementally. In addition, the fact that different validation rules come into play at different stages of the approval cycle meant that the validation mechanism had to be both "item-sensitive" and "context-sensitive". By adopting a rule-based approach, we were able to design and implement a general mechanism for applying validation rules of varying complexity and then add and/or refine form-specific validation rules as they were discovered.

Second, the knowledge that we required to generate a correct "approval path" for a particular PR was not well-defined and distributed among a wide variety of people. We also recognized that in order to guarantee that the system would always generate a correct set of approvers, we would need to be able to incrementally add routing knowledge as "holes" in the existing routing knowledge became apparent. The inherent separation of "inference engine" and "knowledge base" in rule-based systems offered a clear advantage over a conventional procedural approach.

Why CLIPS?

We decided to use the CLIPS shell to implement the knowledge-based portion of the PEPR system for a variety of reasons. First, the data-driven nature of forms processing seemed to suggest that a forward-chaining inference engine would be appropriate. Second, CLIPS runs in a Macintosh environment, which is the platform of choice among our targeted pilot users. Third, the availability of CLIPS source code meant that we could tailor it to our specific needs (see [2] for a more detailed description of the modifications we made to CLIPS). Several other projects within our branch had successfully applied CLIPS to a variety of problems, so there was a fair amount of local expertise in its use. Also, the fact that it is available to government projects at no cost made it particularly appealing.

Key Design Requirements

To evaluate the suitability of a knowledge-based system in an automated workflow environment, it was of course necessary to provide other components of the workflow system. As a result, certain design issues and requirements were identified early in the project. The following represent key assumptions and design desiderata that probably apply to any automated workflow system:

• **Familiar user interface:** The electronic version of the form had to look very much like the paper form with which the users were already familiar. Also, the electronic form needed to perform the rudimentary operations that users have come to expect from any automated application (printing support, automatic calculation of numeric fields, etc.)

• **Reliable data transport mechanism:** In order to get the forms from user to user, the system had to utilize an easy-to-use and reliable electronic mail system.

• **User authentication:** Once users are expected to receive and process sensitive data electronically, they must be assured that the people who sent the data are who they claim to be. Therefore, our system needed to ensure authentication of its users.

• **Data integrity assurance:** Likewise, the users needed to be sure that the data they received had not been altered, either accidentally or intentionally, while in transit.

• **Seamless integration:** We wanted the operation of the knowledge-based component of the system to be completely invisible from the user and have its output appear as data on the form.

• **Tracking capability:** Enabling a user to determine where in the process a particular form is at any particular moment, without having to bother other users, is very important to the acceptance of an automated workflow system. We wanted our users to be able to determine the status of their submitted forms automatically.

Of course, we did not want to have to develop all of the mechanisms required to meet these key needs. Thankfully, most of the requirements described above had been provided by recently-released commercial products. Therefore, our goal in the development of the PEPR system was to make use of commercially-available technology to fulfill these requirements whenever possible, and to integrate the various software components as cleanly as possible. While this approach required us to make use of pre-release versions of some software components of the system (with many of the frustrations inherent in "beta testing"), it enabled us to focus on the development and integration of the knowledge-based component and also led to mutually beneficial relationships with the vendors whose products we utilized.

System Components

Because of the workflow-enabling capabilities of the latest release of the operating system, the availability of workflow-related products, and the relative popularity of the platform at Ames, we chose to implement the first version of the PEPR system on the Apple Macintosh. The PEPR system is comprised of several commercial software tools:

• **Expert System Shell:** As described above, we selected CLIPS with which to implement the knowledge-based portion of the PEPR system.

• **Electronic Forms Package:** The Informed™ package from Shana Corporation is used to produce high-fidelity electronic forms. This package is comprised of the Informed Designer™ program, which permits a forms designer to define the layout and functionality of the form, and the Informed Manager™ program which permits filling out the form by end-users.

• **Scripting Language:** The various software modules that comprise the PEPR system share data by means of AppleScript™, a scripting language for the Macintosh that allows the programs to interact with each other and share data, even across an AppleTalk network.

• **DBMS:** The PEPR system currently utilizes 4th Dimension™, a "scriptable" data base management system from ACIUS, to hold data associated with the routing and tracking of forms as they are sent from user to user.

These applications all operate together under version 7.1.1 of the Macintosh operating system (also known as "System 7 Pro") which provides system-level capability for electronic mail, digital signatures (for user authentication and data integrity assurance) as components of Apple's PowerTalk™ software product. The PowerShare™ system, which provides the store-and-forward mail service and user catalog support, operates on a centrally-located server system and supports all client users.

Each user of the system is required only to be running System 7 Pro and the Informed Manager application. CLIPS and 4th Dimension reside on a central "server" and can be accessed remotely by all users.

Knowledge Base Structure

The PEPR knowledge base is comprised of four main modules; the Validator, the Classifier, the Approval-path Generator, and the Organization "data base". In addition, each form that the PEPR system supports has its own set of form-specific validation rules that are loaded dynamically as the form is processed.

• **Validator:** The PEPR validator is responsible for ensuring that the various fields on the form are correct and complete. The validation rules are represented as CLIPS classes, and are organized hierarchically with their own "apply" methods. Actual form-specific validation rules are represented as instances of these classes and are loaded dynamically from disk files when a particular form type is to be validated. If a validation rule is violated, the validator creates an instance of an error object with a suitable error message that eventually gets returned to a field on the form.

• **Classifier:** If the validator finds no errors on the form, the Classifier is invoked. The Classifier uses the contents of specific fields on the form to construct hypotheses about potential categories to which the specific form might belong. The Classifier loads a group of form-specific "clues" that are comprised of a text string, a field name, a classification category, and a certainty factor. These clues are evaluated in turn; if the clue's text string is found within the associated field, then membership in the given category is established with the given certainty factor. These certainty factors can be positive or negative, and are combined using the CF calculus defined by Shortliffe *et al* for the MYCIN system. If the resulting certainty associated with a certain hypothesis exceeds a threshold value, then the form is said to belong to that category.

• **Approval-path Generator:** Once all of the applicable categories for a given PR have been determined, the approval-path generator looks at specific fields on the form and determines the originating organization. It then loads the form-specific routing rules, and determines both the "management" approvals that are required (which depend upon the originating organization and, often, the total dollar amount associated with the PR) and the "special" approvals that are required (which are dependent on the classification categories to which the PR was assigned). These approvals are represented as the various organizations that must approve the form. The approval-path generator then looks up the "primary contact" for each of these organizations in the "organization data base" and inserts that person's name in the forms electronic "routing slip". (Note that by updating the "primary contact" for an organization periodically allows forms to be routed to designated alternates should the real primary person be on vacation or otherwise unavailable).

• **Organization Data Base:** This portion of the knowledge base contains CLIPS objects that correspond to the various managerial groups and hierarchies at Ames, and is used to help generate approval paths, as described above. (Of course, this module is currently a very good candidate for re-implementation in some other format as an external data base, and the PEPR team is currently negotiating with other Ames groups who maintain similar data bases for other purposes).

Development History

The PEPR system has been under development on a part-time basis for the past three years. Since then, the system has undergone various changes, both in its architecture and functionality. In Early 1991, work began in investigating the problems associated with the procurement process and the potential applicability of software tools to help address those problems. The team identified a useful sub-class of purchase requests on which to begin work, namely those PRs associated with

the funding of research at external universities (this sub-class had the advantages of being reasonably straightforward with respect to the routing required and of providing an immediate near-term benefit -- the AI Research Branch submits a substantial number of these PRs and would therefore be in a good position to evaluate the utility of such a system). In June of 1991, the forms required to support university grants were distributed to a small number of users. These forms included only the evaluation forms (not the PR), and did not utilize the knowledge-based component. Early in 1992, the electronic forms were re-implemented in Informed, which proved to be a superior forms package to the that which had been used previously. These forms (except the PR) were given to numerous users around the Center, and were well-received. The knowledge-based validator, although working, was not deployed to end-users because we lacked a mechanism to efficiently share data between the form and the knowledge system. By the fall of 1992, we had initial versions of both the validator and the approval-path generator working, but they were only usable as a demonstration because they were not well-enough integrated with the forms system to be given to end-users. This "integration" was by means of a popular keyboard-macro package that allowed the two applications to clumsily share data via a disk file. However, this approach had two serious drawbacks. First, the keyboard macro package merely simulated the manipulation of the user interface, and so the user would have been subjected to a very distracting flurry of dialog boxes and simulated mouse-clicks. Second (and more importantly), that integration required that both the forms package and the knowledge base be running on the user's machine. That was an unacceptable limitation and would have undoubtedly "turned off" more users that it would have helped. We were, however, able to give the end-users electronic versions of the grant evaluation forms, which were somewhat helpful to the more experienced users even without the knowledge-based components. In early 1993, the team signed on as pre-release users of AppleScript, and modified the CLIPS shell to be "scriptable". This not only enabled a more "seamless" and less distracting integration between the forms package and the knowledge base, but more importantly it enabled us to set up a single copy of the knowledge system on a central server and permit users to access it over the network. By the summer of 1993, we became pre-release users of the new operating system software (part of the Apple Open Collaboration Environment) that provided support for digital signatures, system-level electronic mail, and other workflow-facilitating features. With these new features came the ability not only to give real users access to the knowledge base validation and routing capability, but also the data integrity assurance that would be required to support electronic submission of the sensitive data contained on financial instruments such as a purchase request form.

In December 1993, the PEPR system "went live", and is now in daily use within the Aerospace Systems Directorate at Ames for the electronic submission, approval, and routing of purchase requests and university grant evaluation forms. The system is even being used to electronically send grant award forms to selected universities, something that had previously been done manually by the University Affairs Office at Ames.

Future Plans

The PEPR team is currently supporting the University Grant pilot testers, and is in the process of making small refinements to the system as the users report problems and suggest improvements. In the coming months, we expect to be able to expand both the user base of the system and the scope of the purchase requests to which it is applied. We are also investigating other related workflow applications, both within Ames and at other government laboratories and within industry.

The PEPR team is also working very closely with researchers at Washington State University who are applying machine learning techniques to electronic forms. Our hope is that as our data base of "correct and complete" forms grows, we will be able to utilize these techniques to automatically generate new validation and routing rules.

References

[1] Compton, M., Wolfe, S. 1993 *Intelligent Validation and Routing of Electronic Forms in a Distributed Workflow Environment.*. Proceedings of the Tenth IEEE Conference on AI and Applications.
[2] Compton, M., Wolfe, S. 1994 *CLIPS, Apple Events, and AppleScript: Integrating CLIPS with Commercial Software.* Proceedings of the Third Conference on CLIPS.
[3] Compton, M., Stewart, H., Tabibzadeh, S., Hastings, B. 1992 *Intelligent purchase request system,* NASA Ames Research Center Tech. Rep. FIA-92-07
[4] Shortliffe, E. H. 1976 *Computer-based medical consultations: MYCIN.* New York: American Elsevier.
[5] Hermens, L.A., Schlimmer, J.C. 1993 *Applying machine learning to electronic form filling.* Proceedings of the SPIE Applications of AI: Machine Vision and Robotics

58-81
34068
P-9

N95-19633

A KNOWLEDGE-BASED SYSTEM FOR CONTROLLING AUTOMOBILE TRAFFIC

Alexander Maravas* and Robert F. Stengel**

Department of Mechanical and Aerospace Engineering
Princeton University
Princeton, NJ 08544

Abstract

Transportation network capacity variations arising from accidents, roadway maintenance activity, and special events, as well as fluctuations in commuters' travel demands complicate traffic management. Artificial intelligence concepts and expert systems can be useful in framing policies for incident detection, congestion anticipation, and optimal traffic management. This paper examines the applicability of intelligent route guidance and control as decision aids for traffic management. Basic requirements for managing traffic are reviewed, concepts for studying traffic flow are introduced, and mathematical models for modeling traffic flow are examined. Measures for quantifying transportation network performance levels are chosen, and surveillance and control strategies are evaluated. It can be concluded that automated decision support holds great promise for aiding the efficient flow of automobile traffic over limited-access roadways, bridges, and tunnels.

Introduction

U.S. automobile traffic has been growing by 4 percent a year to its current level of 2 trillion vehicle-miles, and it is expected to double to 4 trillion vehicle-miles by 2020. According to Federal Highway Administration, if no significant improvements are made in the highway system, congestion delays will increase by as much as 400 percent [1]. According to IVHS America, the annual cost of congestion to the U.S. in lost productivity is estimated at over $100 billion [2]. In many areas there is very little that can be done to increase road capacity. There is not adequate right-of-way next to existing roads, and in many cases the cost of a new highway is prohibitively expensive. It is therefore imperative that new ways be sought to make better use of existing infrastructure.

In 1987 the Federal Highway Administration formed Mobility 2000, a joint effort between the government, industry and academia. This led to the formation of an organization called the Intelligent Vehicle Highway Society of America, or IVHS America [1]. IVHS America aims at improving the level of transportation services that are currently available to the public by integrated systems of surveillance, communications, computer and control process technologies [4].

IVHS technologies have been grouped into four generic elements: Advanced Transportation Management Systems (ATMS), Advanced Driver Information Systems (ADIS), Automated Vehicle Control (AVC), and Commercial Operations [3]. This paper concentrates on ATMS, which involves the management of a transportation network. Implementation of such systems requires development of real-time traffic monitoring and data collection techniques. More precisely, an Advanced Traffic Management System should have the following characteristics, as specified by the proceedings of the Mobility 2000 conference [3,4]:

- real time operation
- responsiveness to changes in traffic flow
- surveillance and detection
- integrated systems
- collaboration of jurisdictions involved
- effective incident control strategies

Effective incident control strategies will be a crucial part of this project. Contrary to widespread belief, not all congestion is due to rush hour traffic; 56% of costs incurred by congestion are due to non-recurrent events or incidents. Incidents include vehicle accidents, unfavorable weather conditions, highway maintenance, and road reconstruction [3]. It is essential to determine the nature and scope of an incident as quickly as possible. The control center should be informed about the incident either through

---

*Student
**Professor
Presented at the Third Conference on CLIPS, Houston, Sept. 1994

the police or other jurisdictional agencies. A more advanced approach would be visual validation of incidents with the use of camera surveillance systems. Effective detection and verification of incidents will lead to lower disruption of traffic flow [3]. Drivers could be routed to alternate paths to avoid unnecessary tie-ups and frustration due to long delays [3].

Traffic control centers would need real-time information about the network condition. An intelligent vehicle/highway system would have to monitor traffic throughout the day. For the near future, sensors will include inductive loops buried just below the surface and ultrasonic sensors mounted overhead. These devices will be able to count the number of vehicles passing a certain point and will gauge their speed. Another alternative would be for image-processing computers to extract traffic data from television pictures from cameras on the arterial network. Recent tests have provided very promising results about the accuracy of inductive loop detectors [1]. Ultimately, improvement of network surveillance technologies and link-time estimation techniques will be crucial in the implementation of an intelligent vehicle/highway system [5]. In the future, vehicles equipped with navigational systems could communicate directly with the control center, giving information about the drivers' locations, speeds, and destinations.

Advanced Traffic Management Systems will have to be integrated with Advanced Traveler Information Systems (ATIS) to ensure higher efficiency of the control system. Drivers will be informed about congestion, roadway conditions, and alternate routes through audio-visual means in the vehicle and through variable message signs at strategic points of the network. Information provided might include incident locations, fog or snow on the road, restrictive speeds, and lane conditions. Two-way real-time communication between vehicles and the control center could be facilitated by radio communications, cellular systems, and satellite communications [4].

Among the benefits of IVHS will be reduction in traffic congestion, reduction in the number of accidents , improved transit service, less fuel wasted, and fewer emissions from idling engines [4]. Fully integrated ATMS/ATIS combinations could reduce congestion in urban areas from 25 to 40%. Unchecked traffic congestion is the largest contributor to poor air quality and wasted fuel consumption. IVHS will not solve all problems in transportation, but it will increase the level of services rendered.

Fundamentals of Traffic Flow Modeling

Evaluating traffic performance requires a thorough understanding of traffic flow characteristics and analytical techniques. The most important macroscopic flow characteristics are flow rate, density, and speed [7]. The flow rate q past a point is expressed as [8]:

$$q = \frac{n}{T} \tag{1}$$

where q: flow rate past a point
n: number of vehicles passing point in time interval T
T: time interval of observation

It is important to recognize that q is sensitive to the selected time interval T during which the measurement began and ended. Whatever the value of T, the most common units for q are vehicles/hr.

Density (or concentration) can be expressed as [7,8]:

$$k = \frac{n}{L} \tag{2}$$

where k: density
n : number of vehicles on road
L : length of road

Density is an instantaneous traffic measurement, and its units are usually vehicles/lane-mile. In most cases, one mile and a single line of vehicles are considered. Traffic densities vary from zero to values that represent vehicles that are completely stopped. The upper limit of k is called the jam density and is on the order of 185 to 250 vehicles per lane-mile, depending on the length of the vehicles and the average distance between vehicles.

Speed is another primary flow variable. Space-mean speed is the average speed of the vehicles obtained by dividing the total distance traveled by total time required, and it is expressed as [8]:

$$u = \left(\sum_{i=0}^{n} s_i \right) / \left(\sum_{i=0}^{n} m_i \right) \quad (3)$$

where u: space-mean speed
s_i: distance traveled by vehicle i on roadway
m_i: time spent by vehicle i on roadway

A very important relationship exists between the three fundamental stream flow variables that have been defined above. This flow relationship is[7]:

$$q = ku \quad (4)$$

A linear speed-density relation has been assumed to simplify the presentation. The relation can be expressed as [7]:

$$u = u_f - \left(\frac{u_f}{k_j} \right) k \quad (5)$$

This relationship indicates that as speed approaches free flow, speed density and flow approach zero. An increase in density causes a decrease in speed until flow is maximized at q_m and speed and density reach their optimum values (u_o, k_o). Further increases in density cause density to reach its maximum value (k_j) and speed and flow to approach zero [7]. A relationship between flow and density can be obtained by substituting equation (5) into (4). This yields [7]:

$$q = u_f k - \left(\frac{u_f}{k_j} \right) k^2 \quad (6)$$

Under low-density conditions, flow approaches zero and speed approaches free-flow speed (u_f). At optimum density, flow is maximized, and speed attains an optimum value. Maximizing the objective function (6) by setting its derivative equal to zero $(dq / dk = 0)$, we find that optimum density occurs at half the jam density $(k_o = k_j / 2)$. This is true only for a linear speed density relationship, but the same reasoning can be applied to other non-linear relationships [7].

The optimum speed is half the free flow speed $(u_o = u_f / 2)$. Because $q_m = k_o u_o$, it is evident that $q_m = u_f k_j / 4$. Once again, it is important to remember that these values are only true for a linear speed-density relation [7]. The flow-density relationship often serves as a basis for highway control. Density is used as the control parameter, and flow is the objective function. At low densities, demand is being satisfied and level of service is satisfactory, but as density increases control is needed to keep densities below or near the optimum density value [7]. Other models have been proposed by transportation planners. This is an optimization problem with traffic flow as the objective function and traffic density as the control parameter. The technology to measure traffic density exists, and its knowledge can help us estimate traffic flow, average speed, travel time, and level of service.

Assumptions of Surveillance and Control Procedures

Every morning there is a large pool of people west of the Hudson river who want to cross it. Every member of this group of "intelligent agents" is part of a continuously changing environment, holding an individual behavioral response to the evolution of the dynamic system.

It has been observed that commuters usually choose the route with the shortest travel time. Consequently, drivers tend to divide themselves between two routes in such a way that the travel times between them are identical. Finally, an equilibrium point is reached in which commuters do not have much choice between two routes since the user cost (travel time) of traversing the two links has stabilized [8].

In many cases, an incident can disturb the user equilibrium. In such instances transportation planners can help establish a new equilibrium where no route is underutilized. By dispensing traffic advisories to the right people, one can be assured that misallocation of transportation resources can be avoided. Computer simulations have proven that the use of route guidance systems can reduce travel time to all drivers by up to 20% [9]. Route guidance was found to be more helpful as the duration of incidents increased, in which cases more drivers had to be routed to achieve an optimal traffic assignment [9]. A corresponding surveillance procedure is illustrated in Figure 1.

In selecting between alternate routes, a user-optimal system chooses a route that minimizes the travel time of the individual driver.

However, a system-optimal system selects a set of routes that minimizes the overall travel time of all drivers [11]. Routing decisions should not be made independently because every decision effects the whole network: if many drivers choose a certain route at the same time, that route may become congested and non-optimal [10].

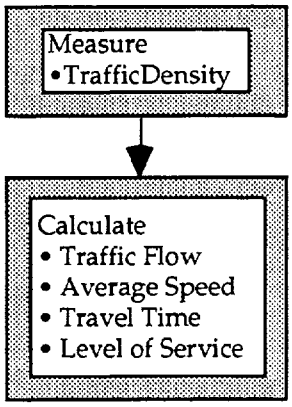


Figure 1 Proposed Surveillance Procedure.

A Declarative Framework for Traffic Control

Traffic control can be partitioned into reflexive, procedural, and declarative functions. Reflexive functions operate at the subconscious level of human thought; they are instantaneous reactions to external stimuli. Procedural actions also operate on a subconscious level, but they are more complex than reflexive functions, following a set of basic rules and actions that represent skilled behavior. Declarative functions operate at the conscious or preconscious level of thought. On the conscious level they require attention; on the preconscious level they require intuition and conceptual formulations. They involve decision making and provide models for system monitoring, goal planning and system/scenario identification [12,13].

A traffic management system requires goal planning and system monitoring. At every instant, all alternatives must be considered and decisions must be made through a deductive process [13]. A declarative model makes such decisions in a process that is similar to human reasoning [6]. All our knowledge, beliefs, and experience of traffic modeling are placed in a declarative framework to provide a system that reasons in an intelligent manner.

This paper focuses on declarative traffic controls. Rules that reflect the controllers knowledge of the response of the control system are established. Programming is implemented as a CLIPS expert system that supports various programming paradigms. CLIPS was chosen because it can be easily integrated with C programs. The rule-based programming paradigm is useful in modeling traffic incidents. In procedural programming, the order in which all the commands are executed is pre-specified. In reality, traffic incidents are often unpredictable. By establishing the appropriate heuristics and rules, the system becomes "intelligent." When it is faced with a certain problem, it uses pattern matching that is appropriate to the existing facts.

Computer systems often are organized in a modular structure to increase computational efficiency. Time can be saved by looking at rules and facts that are relevant at that instant. Modular representation allows partitioning of the knowledge base into easily manageable segments, thus making the system easily expandable. CLIPS incorporation of modules is similar to the blackboard architecture of other systems. Knowledge sources are kept separate and independent, and different knowledge representation techniques can be used. Communication of all sources takes place through the blackboard [15].

Task definition is an important factor in the development and design of such rule-based systems. The ultimate goal is to develop an expert system of expert systems, which is a hierarchical structure that reasons and communicates like a team of cooperating people might [13].

A knowledge-based system called the Traffic Information Collator (TIC) has been developed at the University of Sussex. The TIC receives police reports on traffic incidents and automatically generates appropriate warning messages for motorists. The system operates in real-time and is entirely automatic. The TIC is comprised of five processing modules each holding its own knowledge base [16,17].

Research in air traffic control has been conducted in a similar manner. Cengeloglu integrated an Air Traffic Control Simulator (written in C) with a traffic control decision framework (implemented in CLIPS). The simulator creates a virtual reality of airspace and continuously updates the information

(knowledge base) of the decision framework. Several scenarios containing unexpected conditions and events are created to test the prototype system under hypothetical operating conditions [15].

A complete transportation model should consist of a traffic simulator and a traffic manager. This paper focuses on the decision process of managing and controlling traffic.

Prediction algorithms could be employed to predict the evolution of traffic. This additional knowledge could be used in the control process. Smulders created a model to study the use of filtering on freeway traffic control [18]. Once the actual values of the density and mean speed in all sections of the freeway are available, his model generates predictions for the evolution of traffic over short time periods (e.g., 10 minutes). A state vector contains all the information about the present state of the system. The estimation of a certain state from the measurement vector is done through the use of a Kalman filter [18].

The real-time operation of knowledge-based systems in actual or simulated environments is attained through the use of a cyclic goal-directed process search, with time-sliced procedure steps. Prior to a search, the value of parameters is either *known* or *unknown*. After the search, the status of parameter values may be changed. Repetitive knowledge-base initialization sets the value of every parameter to its default value after each search cycle; all information that was attained in the previous search is deleted. Thus the controller "forgets" the information it acquired in a former search prior to solving a new problem.

The CLIPS knowledge base can be initialized by resetting the program. All the constructs remain unchanged and do not have to be reloaded. This process allows the accommodation of time-varying data in the system [14]. In this sense, real-time application implies some parallel execution features. Several domains of the research space must be searched concurrently [19]. It is noteworthy that not all cyclic search processes of the system have to be synchronous.

Using a Knowledge-Based System for Decision Aiding in Traffic Control

The aim of this project is to design an expert system that evaluates traffic conditions and dispenses travel advisories to commuters and traffic control centers. A decision-support system has been written in the CLIPS programming environment to illustrate several traffic control procedures (Fig. 2). The program is geared toward illustrating a method of traffic control rather than solving a particular problem. Some traffic parameters have been approximated and certain simplifying assumptions have been made to avoid unnecessary computational complexity.

A small network was constructed for initial program development. The network consists of a section of the New Jersey Turnpike and the routes connecting Exits 14, 16, and 18 to the Holland Tunnel, Lincoln Tunnel, and George Washington Bridge. Node and link data for all the roads in New Jersey and New York has been accumulated by the Civil Engineering Department at Princeton University. Future research could be geared toward applying the declarative control procedures developed in this project to a network of a larger scale.

Actual implementation of this system is based on it receiving real-time information about traffic conditions of a network using sensor measurements at different points on the network. The current program uses scenario files, with hypothetical traffic densities from all roads in the network. The program reads these values (as well as the pertinent time period of the day) and creates the appropriate data constructs, which are asserted as facts. The system then matches these facts based on pre-specified heuristics. Once it has concluded a declarative search, it gives certain advisories and recommendations at the CLIPS command prompt. All advisories are made with the idea that they would be broadcast on changeable signs at several roadway locations; they could also be displayed at traffic management centers or on the displays of suitably equipped vehicles. Alternative scenario files test the system under several hypothetical operating conditions.

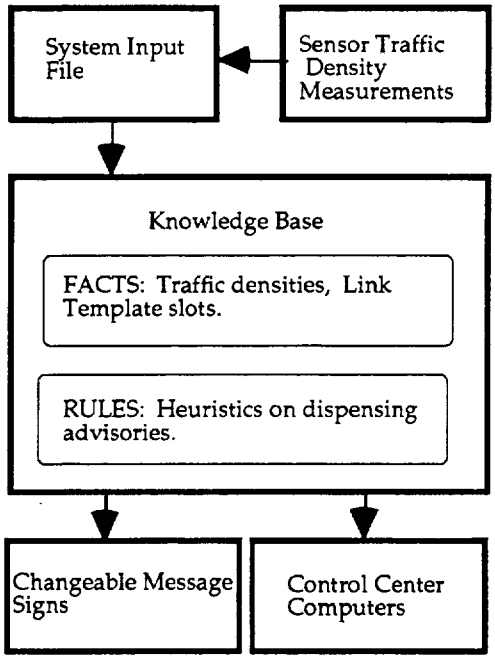


Figure 2 Schematic Representation of System Architecture.

Implementation of the Traffic Management System

The decision-support system, programmed with CLIPS 6.01, supports several procedural functions, that are necessary for the computation of relevant information. Once the density is known, the average travel speed, flow rate, service rate, and level of service can be calculated from the appropriate models and equations.

Historical data of expected traffic demand for the Hudson river crossings have been stored in a function that returns the expected number of vehicles at a certain time period. The use of an appropriate filter would make this function redundant. It provides information about the probable evolution of traffic and is sufficient for the preliminary development of the system. It makes the system "forward looking" and capable of predicting congestion buildup. Historical data could be stored as facts; however, this clogs up the facts list, and the system is forced to look at irrelevant facts.

After reading initial data from the scenario file, the roadway densities are asserted as facts, and they are put onto the facts list. The initialization rules create the appropriate data constructs based on these basic facts and the appropriate procedural functions. Most of the program's knowledge is stored in templates, which are similar to structures in C. Thus every link of the network has it own template, containing information such as speed, density, flow, operational lanes, service rate, and accident-status. Templates are convenient for storing data and can be readily accessed and modified by the user. After all the values of the slots of the templates have been calculated, the templates are asserted as facts. Thereafter, decisions are made by pattern matching on the values of the templates' slots. Data storage is compact so as not to overflow the CLIPS fact list.

The system is readily adjustable to lane closures due to maintenance. For instance, if one of the lanes of the Lincoln Tunnel is closed due to maintenance, then when the knowledge base is initialized, it takes this fact into account. In the case of bad weather, such as a snowstorm, the values of the free-flow speed and the jam density should be reevaluated. Since these deviations from normal operating conditions are incorporated into the system once it is initialized, all decisions made thereafter are adjusted accordingly. System initialization at every time increment ensures that the link-node data is augmented to reflect current conditions. Thus the system is very flexible and can include all foreseeable traffic situations.

Broadcasting travel advisories is a challenging part of the program. For instance, the fact that there is an accident on the route from Exit 14 to the Holland Tunnel is useful for people approaching that exit, but not for people traveling away from it. Thus the system is faced with the decision of whether to make a broadcast and to whom it should be made. Due to the geometry of the network, the broadcast heuristics are different for every decision node. Even in such a small network, there are numerous broadcasting possibilities.

Automated communication between commuters and the control center can be achieved by the appropriate use of advisory rules. The data templates also contain information such as the presence and severity of accidents on the links of the network. Every link's template has a slot that is called "accident status," which can be either TRUE or FALSE. The default value is FALSE, but once an accident occurs it is switched to TRUE. Thereafter, the system ensures that the appropriate people are informed of the incident. Templates also store the

estimated time to restore the link to normal operating conditions and the degree of lane blockage at the accident site.

While radio advisories provide commuters with information about traffic delays and adverse traffic conditions, it is doubtful that they give them all the information they need at the right time. Since not all drivers are tuned to the same station and radio advisories may lack central coordination, the result of broadcasting may not be system-optimal. Changeable message signs ensure that the appropriate people get the right message at the right time. The system is more helpful under such circumstances, since it can help bring the network back to user equilibrium by ensuring that all links are properly utilized.

In general the factors that should be considered in any routing decision are: a) traffic density and velocity profile of main and alternate route, b) length of main and alternate route, c) percentage of divertible traffic volume, and d) demands at on-ramps on both routes [20]. Once the density of the link is known, an average speed can be computed using one of the traffic models described earlier in this paper. Dividing the length of the link by the average speed yields the current (experienced) travel time.

Routing between two alternate routes can be achieved by the use of a tolerance-level measurement. The difference between experienced travel times (or flow) between two routes can be calculated. If it is higher than some prespecified level, then commuters should be routed to the underutilized roadway. The use of individual travel times is user-optimal, whereas the use of traffic flow is system-optimal. Optimizing commuters travel time can probably be done better with the use of shortest-path algorithms, whereas equilibrating flows can be incorporated in an expert system.

Traffic flow is a measure of highway productivity. Ensuring maximum utilization is essentially an optimization problem, with flow as the objective function and density acting as the control parameter. The flow of a link is maximized at the optimal density (k_o). For a linear model, the optimal operating density is half the jam density (k_j). The system has the expertise and knowledge to recognize how far the actual density measurements are from ideal conditions. If the traffic density of a road is less than optimal, then the road is under-utilized. If the density is higher then optimal, then it is over-utilized. An advanced intelligent highway system should be able to monitor and compare traffic flow in both directions of a link to see if the decision for a lane-direction change is warranted. The ultimate goal is to ensure that all routes are utilized properly. This section of the system is subject to a lot of development and improvement since the domain knowledge is uncertain. Currently there is no clear way to route traffic optimally.

The system searches its historical data to see if traffic demand for a link is expected to rise or fall in the next time period. If travel demand is expected to fall and the road is being underutilized, the system suggests that more vehicles be routed to that link. Diversion of traffic flow is an effective method of improving traffic performance and can be achieved by rerouting drivers with specific direction and destinations [20]. Advisories on the New Jersey Turnpike are different for people traveling North than for those traveling South.

The issue of how long a message should be broadcast is significant. Suppose that congestion can be avoided if 30% of drivers respond to the diversion sign. If only 15% of the drivers follow the diversion recommendation, congestion will not improve as expected. A feedback control system could take this into account by deciding to broadcast the diversion message for a longer time period until the desired utilization levels are reached. This approach compensates for all uncertainties in the percentage of divertible traffic flow [20]. The critical design issue is assuring that the system reaches stability quickly.

Knowledge base initialization at frequent time intervals, through the use of sensor measurements, makes this method of broadcasting a closed-loop feedback control process. Since the real-time implementation of this system would rely on cyclic search, a message would be sent every time the system decides that traffic should be diverted. Currently the program does not have real time measurements or simulated traffic demands, so it does not execute a cyclic search. It considers 24 one-hour periods over the span of a day. Real time implementation would require that the CLIPS knowledge be reinitialized at every time increment with the state measurements.

Conclusion

This project has made a step towards emulating an automated decision process for an Advanced Transportation Management System. This non-conventional approach to transportation modeling has examined the applicability of intelligent control techniques in management. Since a fully operational Intelligent Vehicle Highway System will require full integration of symbolic and numerical knowledge, declarative rules have been embedded with procedural code. A framework for modeling traffic incidents has been provided. The link information of the system is augmented on a real-time basis, and advisories are issued based on the current state of the system.

Operation of this system requires that the traffic control center has knowledge of the traffic densities of all links in the transportation network. The technology for making such measurements exists and has been described earlier in this paper. Implementation of this system will require research into how programming software will be integrated with all the system sensors.

Before fully implementing such a control process, it is necessary to choose what degree of automation the system should have. The system should be allowed to run on its own, without human intervention, only when all possible errors have been removed from it.

It is difficult to foresee all the incidents that could happen. Even for such a small network there are many possibilities for issuing travel advisories. As the area that is monitored by sensors becomes larger, it is evident that human operators cannot check everything that is going on. However, the heuristic rules and frequent knowledge-base initialization make the system adaptive to many situations. The size of the knowledge base and the number of advisories are limited only by the available computer memory. Initially the system can be tested in a small area. Thereafter, additional rules for broadcasting to other locations can be added incrementally. Additional details of this research can be found in Ref. 21.

Future Work

Future work can be geared towards expanding the knowledge base by using the object-oriented programming paradigm offered by CLIPS. Node and link data of large networks could be stored compactly in an object-oriented format. Different classes of roads could be defined (e.g. arterial, expressway, intersection). Every link would then be an instance of these classes and it would inherit some of its properties from them.

The cyclic search must be implemented with the use of actual or simulated data, requiring the knowledge base to be reset (initialized) at frequent time intervals. It would be interesting to link CLIPS with a traffic simulator written in C. The simulator could generate traffic demands, and the CLIPS program could issue appropriate advisories.

This system knowledge base is subject to refinement. Additional rules must be added so the system knows what to do in the absence of certain density measurements, which could arise from malfunctioning sensors. Since not all transportation engineers use the same traffic models, it would be desirable to allow the user to use different traffic models or to be able to create his own model with an equation parser. The traffic routing technique and its relevant objective function needs to be reevaluated. Backlogs due to ramp-metering also should be examined. The use of estimators and prediction algorithms would enhance system performance significantly.

A learning control system would be able to learn from its daily experiences. Initially, it could be "trained" on a set of simulated data. Data archiving, on a daily basis, would increase the size of the system's knowledge base. Thereafter, it could evaluate how well it handled a previous accident. Hence, if it was in a similar situation it would use its acquired expertise to issue the appropriate advisories. An intelligent system could detect traffic incidents from unusual sensor readings. Eventually the system would be able to distinguish between weekday and weekend traffic patterns. Considering the recent technological advances in intelligent control systems, the era of the fully automated highway might not be very far away.

Acknowledgement

This work was supported by the National Science Foundation, Grant No. ECS-9216450.

References

[1] Bernstein D., Ben Akiva M., Holtz A., Koutsopoulos H., and Sussman J., "The Case for Smart Highways," Technology Review, July 1992.

[2] Anon., IVHS Strategic Plan Report to Congress, Department of Transportation, Dec. 18, 1992.

[3] Proceedings of a Workshop on Intelligent Vehicle/ Highway Systems, Mobility 2000, San Antonio Texas, Feb. 15-17, 1989.

[4] Euler G.W., "Intelligent Vehicle/Highway Systems: Definitions and Applications," ITE Journal, Nov. 1990.

[5] Ervin, R.D., An American Observation of IVHS in Japan, Ann Arbor, Michigan, 1991.

[6] Stratton D.A., Aircraft Guidance for Wind Shear Avoidance: Decision Making Under Uncertainty, Ph.D. Dissertation, Princeton University, Princeton, 1992.

[7] May A.D., Traffic Flow Fundamentals, Prentice Hall, Englewood Cliffs, 1990.

[8] Morlok E.K., Introduction to Transportation Engineering and Planning, McGraw-Hill , New York, 1978.

[9] Rakha H., Van Aerde M., Case E.R., Ugge A., "Evaluating the Benefits and Interactions of Route Guidance and Traffic Control Strategies using Simulation", IEEE CH2789, June 1989.

[10] Solomon M., New Approaches to the Visualization of Traffic Flows in Highway Networks, Senior Thesis, Princeton University, Princeton, 1992.

[11] Van Aerde M., Rakha H., "Development and Potential of System Optimized Route Guidance Strategies", IEEE CH2789, June, 1989.

[12] Chao T., Design of an Intelligent Vehicle/Highway System Computer Simulation Model, Princeton University, Mechanical and Aerospace Engineering Department Senior Independent Work Final Report, Princeton, 1994.

[13] Stengel R.F., "Probability-Based Decision Making for Automated Highway Driving, to appear in *IEEE Trans. Vehicular Technology* (with A. Niehaus).

[14] Handelman D.A., A Rule-Based Paradigm for Intelligent Adaptive Flight Control, Ph.D. Dissertation, Princeton University, Princeton, June 1989.

[15] Cengeloglu Y., A Framework for Dynamic Knowledge Exchange Among Intelligent Agents, Masters Thesis in Electrical and Computer Engineering, University of Central Florida, Orlando Florida, 1993.

[16] Evans R., Hartley A.F., "The Traffic Information Collator", Expert Systems, Nov. 1990, Vol. 7, No. 4.

[17] Allport D., "Understanding RTA's", Proceedings of the 1988 Alvey Technical Conference, 1988.

[18] Smulders S.A., "Modeling and Filtering of Freeway Traffic Flow", Transportation and Traffic Theory, Elsevier, New York, 1987.

[19] Le Fort N., Aboukhaled, D.Ramamonjisoa, "A Co-Pilot Architecture based on a multi-expert system and a real-time environment", 1993 IEEE International Conference on Systems, Man and Cybernetics, Vol. 5, France Oct. 17-20, 1993.

[20] Cremer M., Fleischmann S., "Traffic Responsive Control of Freeway Networks by a State Feedback Approach", Transportation and Traffic Theory, Elsevier, New York, 1987.

[21] Maravas, A., A Knowledge-Based System for Optimal Control of Traffic Flow, Princeton University, Mechanical and Aerospace Engineering Department Senior Independent Work Report, Princeton, 1994.

DEVELOPMENT OF AN EXPERT SYSTEM FOR POWER QUALITY ADVISEMENT USING CLIPS 6.0

A. Chandrasekaran **P.R.R. Sarma**
Tennessee Technological University
Cookeville, TN 38501

Ashok Sundaram
Custom Power Distribution Program
Electric Power Research Institute
Palo Alto, CA 94303

ABSTRACT

Proliferation of power electronic devices has brought in its wake both deterioration in and demand for quality power supply from the utilities. The power quality problems become apparent when the users' equipment or systems maloperate or fail. Since power quality concerns arise from a wide variety of sources and the problem fixes are better achieved from the expertise of field engineers, development of an expert system for power quality advisement seems to be a very attractive and cost-effective solution for utility applications. An expert system thus developed gives an understanding of the adverse effects of power quality related problems on the system and could help in finding remedial solutions. The paper reports the design of a power quality advisement expert system being developed using CLIPS 6.0. A brief outline of the power quality concerns is first presented. A description of the knowledge base is next given and details of actual implementation include screen outputs from the program.

INTRODUCTION

The introduction of nonlinear loads and their increasing usage, have led to a point where the system voltage and current are no longer sinusoidal for safe operation of the equipment at both industrial and customer levels. Before the advent of electronic and power electronic devices, the current distortions were due to saturation of the magnetic cores in the transformers and motors, arc furnaces and mercury arc rectifiers. Even though the overall effect on the system was there, it had no serious effect on the performance of comparatively more rugged and insensitive equipment. Today, sensitive electronic and microprocessor based equipment are commonly used and they are susceptible to variations and distortions of the sinusoidal wave. Distorted sinusoidal waveforms of voltage and current are produced due to the nonlinear characteristics of the electronic components which are used in the manufacture of any electronic related equipment. The power quality problems arising in the system basically are impulses, surges, sags, swells, interruptions, harmonics, flicker etc. These power quality problems can be a potential threat to the satisfactory operation of the equipment. Thus there is a need to alleviate or eliminate the effects of poor power quality. Engineers are applying their experience with the causes of power quality impairment to recommend solutions to correct the problems.

Expert systems could be used for domain specific problems, such as power quality. The solutions to power quality may be many and it may not be possible to come to a single conclusion. Expert System approach can be useful to get a very successful approximate solution to problems which do not have an algorithmic solution and to ill-structured problems where reasoning may offer a better solution. The opinions of the experts may vary regarding a particular problem, but the level of expertise combined from several experts may exceed that of a single human expert, and this can be made use in an Expert System. The solutions given by an expert system are unbiased but the solutions from an expert may not always be the same. Databases can be accessed by an

expert system and algorithmic conclusions can be obtained wherever possible.

POWER QUALITY (PQ) IN POWER SYSTEMS

The impulses arising in the system are usually due to lightning and they cause insulator flashover, transformer failures, arrestor failures and damage to other substation equipment. Lightning strokes can pass through the transformer neutrals and cause damage to the customer equipment also. This is especially when the lightning strikes close to the distribution transformers. Oscillatory transients can be classified into three groups as low frequency, medium frequency and high frequency transients. Their range is from below 5 kHz to above 500 kHz. The main causes for these transients are capacitor switching, cable switching, back to back capacitor energization, travelling waves from lightning impulses and circuit switching transients. The impact of these transients are very high voltage levels on the secondary side of the transformers, especially due to capacitive coupling between primary and secondary windings, for medium frequency transients. Thus, these transients can also cause equipment failures and disruption of sensitive electronic equipment.

Sags and Swells can be classified into three groups namely instantaneous, momentary and temporary. They may last from 0.5 cycle to 1 minute. Sags may cause dropout of sensitive electronic equipment, dropout of relays, overheating of motors etc. Swells are usually due to single-line-to-ground faults occurring in the system. The extent of voltage rise varies with the type of grounding scheme adopted. An increase in the voltage of 73.2% is for ungrounded systems. The effect of swells could cause Metal Oxide Varistors to be forced into conduction.

Over-voltages and under-voltages last longer than 1 minute. They are caused by load switching, capacitor switching or due to bad system voltage regulation. The impact of these is dropout of sensitive customer equipment which usually require constant voltage.

Harmonics are caused due to nonlinear characteristic of the loads of which converter circuits, arcing devices etc are some examples. Harmonics are multiples of fundamental frequency and are continuous in nature and distort the sinusoidal waveform. Interharmonics are caused by cycloconvertors and arc furnaces. The impact of harmonics can cause overheating of transformers and rotating machinery, maloperation of sensitive electronic equipment, maloperation of frequency sensitive equipment, metering errors, presence of neutral to ground voltage resulting in possible neutral overloading, capacitor failures or fuse blowing, telephone interference,increased power losses, etc.

Noise is caused by the range of components less than 200 kHz. Improper grounding and operation of power electronic equipment are the main causes of noise production. The impact of noise is on telephone interference. Notching is due to commutation in three phase inverters and converters. The number of notches depend on the converter and invertor configuration. The converters and invertor circuits can be operated in a six pulse, twelve pulse, eighteen pulse or more configurations. Flicker is usually caused by presence of components less than 25 Hz. Arc furnaces and intermittent loads like welding machines are some examples. These low frequency components may cause problems with lighting.

The various disturbances leading to power quality deterioration can stem from a wide variety of reasons that are often interdependant and quite complicated. A thorough analysis may be time consuming and inefficient. The solutions also depend upon experience of the system.

EXPERT SYSTEM APPROACH TO PQ

Expert systems came into existence as an outcome of the need for better problem solving methods where a closed form solution is unavailable. Since knowledge of power quality

problems is obtained more through study and experience, development of an Expert System is a viable approach.

An Expert system comprises of user interface, working memory, inference engine, agenda, and knowledge acquisition facility. The user interface is a mechanism by which the user and the expert system communicate. Working memory consists of a global database of facts used by the rules. An agenda is a list of rules with priority created by the inference engine, whose patterns are satisfied by facts or objects in the working memory. The knowledge acquisition facility is an automatic way for the user to enter knowledge in the system rather than by having the knowledge engineer explicitly code the knowledge.

Rule-based, object-oriented and procedural programming paradigms are supported by CLIPS 6.0. The basic components of CLIPS 6.0 are facts list, knowledge base and inference engine. The fact list contains the data on which inferences are derived, knowledge base contains all the rules and the inference engine controls the overall execution. Thus the knowledge base contains the knowledge and inference engine draws conclusions from the knowledge available. The user has to supply the expert system with facts and the expert system responds to the users' queries for expertise.

Knowledge can be represented by rules, semantic networks,parse trees, object-attribute-value triples, frames, logic etc. Each have their own limitation and a suitable field of usage [1]. Trees and lattices are useful for classifying objects because of hierarchical nature.

Decision trees can be used effectively in the development of power quality expert system due to hierarchical nature of power quality problems as usually one problem leads to several problems. A decision structure is both a knowledge representation scheme and a method of reasoning about it's knowledge. Larger sets of alternatives are examined first and then the decision process starts narrowing till the best solution is obtained. The decision trees should provide the solution to a problem from a predetermined set of possible answers. The decision trees derive a solution by reducing the set of possible outcomes and thus getting closer to the best possible answer. Since the problems pertaining to power quality have different effects the solutions and remedial actions may be approximated by successive pruning of the search space of the decision tree.

A decision tree is composed of nodes and branches. The node at the top of the tree is called root node and there is no flow of information to the root node. The branches represent connection between the nodes. The other nodes, apart from the root node, represent locations in the tree and they can be answer nodes or decision nodes. An answer node may have flow of information to and from the other nodes and are referred as child nodes. The decision node is referred as leaf node and represents all possible solutions that can be derived from the tree. Thus the decision node terminates the program with solutions.

Broadly classifying a system, the power quality problems are felt at the distribution system level or a customer level. Thus this can be taken as the root node. At the distribution level the problem may be with the equipment at the substation or with the equipment in the distribution lines. Similarly the problem at the customer level could be with an industrial customer, a commercial customer or a residential customer. Thus these can be referred to as the child nodes. The problems faced at each of these levels may be understood better with each successive answer node, and thus probable answers could be arrived at, at the decision nodes. This approach could be useful if any numeric or monitored data pertaining to the system is not available. Also a probable answer can be concluded, depending on the answers given by the user only by observable impacts on the equipment, say the equipment maloperating or equipment failing or fuse failing, motor getting overheated etc. The efficiency or the level of certainty of solutions given will depend on the information provided by the user regarding the problem.

When monitoring equipment are connected at various points at the customer facility, data pertaining to the various disturbances due to surges, impulses, interruptions, variations in voltage, current etc can be collected or stored. Many such monitors are available and used by the utilities. PQ node program of Electric Power Research Institute (EPRI) uses a number of monitors for collecting disturbance data. The output from these analyzers can be used by the expert system for diagnosis. This approach could be very effective as the user may not be able to provide enough information from his knowledge. This will increase the efficiency of the expert system, as the user may give an incorrect answer or may not be able to answer some questions. The data files can be accessed into the expert systems, which is CLIPS 6.0, in our study and the data can be internally manipulated to come to a certain decision. If necessary, further information can be elicited from the user before arriving at a final solution.

Standard reports of the outputs with graphs of disturbances, possible solutions etc can be generated by the program by opening text files within the program.

IMPLEMENTATION OF THE EXPERT SYSTEM

EPRI is sponsoring a research project for the development of software modules for addressing power quality problems in the utilities. The center for Electric Power of the Tennessee Technological University is co-sponsoring the project. The major objectives of the project are the following :

- Design and test a prototype expert system for power quality advising.
- Develop dedicated, interactive analysis and design software for power electronic systems.
- Develop a concept of neural network processor for the power system disturbance data.

For developing the expert system, CLIPS software designed at NASA/Johnson Space Center with the specific purposes of providing high portability, low cost and easy integration with other systems has been selected. The latest version CLIPS 6.0 for Windows provides greater flexibility and incorporates object-oriented techniques.

Figure 1 shows the configuration of the expert system blocks being analyzed for the power quality advisement. The rules are framed by the experts opinion and the number of rules can be increased and added to the program whenever possible. The monitored data from PQ nodes, user or both can supply the facts, which can be utilized to come to a probable solution. The inference engine and agenda fire the rules on the basis of priority. There is an explanation facility which gives reports and suggests the possible reasons for coming to a conclusion depending on the inputs given.

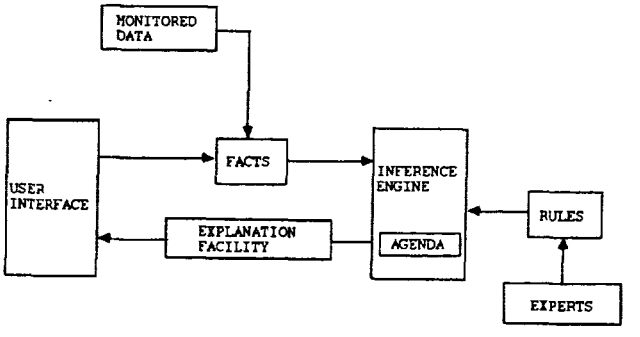


FIG 1 : BLOCK DIAGRAM OF EXPERT SYSTEM

The following is the pattern in which the questions are asked by the Expert System for the responses of the user. Here a batch file is run and CLIPS 6.0. response is given below.

```
CLIPS> (open "s.dat" s "w")
TRUE

CLIPS> (open "r.dat" r "w")
TRUE

CLIPS> (load "sept1.clp")
!!**************************************************************
*********
TRUE
```

```
CLIPS > (reset)

CLIPS > (run)

   ************************************
   *                                  *
   *   Expert System For Power Quality Advisement   *
   *                Developed By      *
   *   Dr. A. Chandrasekaran and P.R.R. Sarma       *
   *          Center For Electric Power             *
   *          Tennessee Technological University    *
   *          Cookeville, Tennessee.                *
   *                                  *
   ************************************

            Where is the complaint from?

            1. Distribution-system
            2. Customer
1

            Where is the problem occurring ?

            1. Circuits
            2. Sub-Station
2

            What is the complaint ?

            1. Service-Interruption
            2. Device-Maloperation
            3. Equipment-Failure
2

            Which of these has the problem ?

            1. Transformer
            2. Circuit-Breaker
            3. Control-Circuitry
1

            Is the system voltage normal ?

            1. Yes
            2. No
1

            Is the hum of the transformer excessive ?

            1. Yes
            2. No
1

            Is the transformer getting overheated ?

            1. Yes
            2. No
1

            Is the transformer getting overheated even
            under normal load conditions ?

            1. Yes
            2. No
1

            Is the transformer supplying power to
            heavy lighting and power electronic loads ?

            1. Yes
            2. No
1

            Are there capacitor banks in the substation ?

            1. Yes
            2. No
1

            Is the substation provided with harmonic
            filters to filter harmonics ?

            1. Yes
            2. No
2

            Probable cause:

            Place filters to prevent saturation
            and overheating of the transformers
            when it supplies power to power
            electronic loads.

CLIPS > (exit)
```

The following is the symptoms file "s.dat", which was storing the symptoms pertaining to the problem is given below.

```
****************************************
*                                      *
*          THIS EXPERT SYSTEM IS BEING         *
*                                      *
*              DEVELOPED BY            *
*                                      *
*      Dr. A.CHANDRASEKARAN & P.R.R. SARMA     *
****************************************

SUMMARY OF SYMPTOMS OBSERVED:

** The complaint is from the distribution system.

** The trouble is in the substation.
```

** Complaint is device maloperation.

** The problem pertains to the transformer.

** The hum of the transformer is excessive.

** The transformer is getting overheated.

** The transformer is supplying power to lighting and power electronic loads.

** The transformer is getting heated under normal load conditions.

The following is the possible reasons file "r.dat", which was storing the possible reasons pertaining to the problem is given below.

** The transformer hum could be more due to loose core bolts, presence of harmonics etc.

** Transformer may get overheated due to overloading, saturation, insignificant faults in winding etc.

** Transformer can get overheated under normal load conditions due to harmonics, insignificant faults in transformer etc.

** Harmonics may be in the system when the power is being supplied to lighting and power electronic loads.

** Harmonics may be present if harmonics are not filtered out.

The text files generated can be imported into Word perfect, Winword etc for making the final reports. At present, efforts are being made to integrate the data obtained from the PQ analyzer into the CLIPS program as a part of the EPRI project. The monitored data reports from the PQ analyzer can be used to arrive at conclusions about the power quality problems especially with regard to the origin of the disturbances from the users' queries. Also the reports could incorporate disturbance graphs obtained from the data specified in the input files of the monitored data.

The concepts of the theories of uncertainty and Fuzzy Logic can be utilized using FuzzyCLIPS in the development of the power quality expert system, especially in cases where the questions like " Is the hum of transformer excessive?" have to be answered. The possibilities for 'excessive' may be more than normal, high, very high etc. FuzzyCLIPS may be used for answers of this sort and the efficiency of the solutions thus can be increased.

CONCLUSION

The development of the power quality expert system shows that expert system usage is definitely a viable alternative. Analysis of monitored data can be used to identify the sources causing power quality deterioration. Suitable conclusions can be drawn to recommend mitigating the power quality problem sources to the customers on one hand and the equipment manufactures on the other. An expert system approach can also be useful to educate customers on actions that can be taken to correct or prevent power quality problems.

REFERENCES

1. Giarratano Joseph and Riley Gary, "Expert Systems, Principles and Programming", 2nd edition, PWS Publishing Company, Boston, 1993.

2. Adapa Rambabu, "Expert System Applications in Power System Planning and Operations", IEEE Power Engineering Review, February 1994, pp. 12 to 14.

3. Liebowitz Jay, DE salvo A. Daniel, "Structuring Expert Systems", Yourdon Press, Prentice Hall Building, Engelwood Cliffs, N.J. 07632, 1989.

4. "A Guide to Monitoring Power Distribution Quality", phase 1, Electric Power Research Institute Project 3098-01, Report TR-103208, Palo Alto, California, April 1994.

QPA-CLIPS: a language and representation for process control

Thomas G. Freund*

Abstract

QPA-CLIPS is an extension of CLIPS oriented towards process control applications. Its constructs define a dependency network of process actions driven by sensor information. The language consists of 3 basic constructs: TASK, SENSOR and FILTER. TASKs define the dependency network describing alternative state transitions for a process. SENSORs and FILTERs define sensor information sources used to activate state transitions within the network. *deftemplate*'s define these constructs and their run-time environment is an interpreter knowledge base, performing pattern matching on sensor information and so activating TASKs in the dependency network. The pattern matching technique is based on the repeatable occurrence of a sensor data pattern. QPA-CLIPS has been successfully tested on a SPARCStation providing supervisory control to an Allen-Bradley PLC 5 controller driving molding equipment.

1.0 Introduction - the need

Process control is the science and, at times, art of applying and holding the right setpoint value and/or mixing the right amount of an ingredient at the right time. But, when is the time right ? And, if we know enough about the material or mixture, what is the right setpoint value and/or amount that has to be mixed ?

Materials scientists and engineers have spent years of painstaking experimentation and analysis to characterize the behavior of metals, plastics, and composites. Along the same vein, manufacturing engineers and factory floor operators have developed many person-years worth of practical "know-how" about material behavior under a variety of processing conditions.

In addition, though there is always room for improvement, significant strides have been made in useful sensor technology for acquiring information on material behavior. As the types of materials used in everyday products increase in complexity, the demand increases for embedding a better understanding about material behavior directly into the control of their manufacturing processes.

---

* Author's address - Pratt & Whitney, 400 Main Street, Mail Stop 118-38, East Hartford CT 06108. This work was supported by a grant from the National Center for Manufacturing Sciences, Ann Arbor, MI 48108 and in cooperation with Allen-Bradley Co. (Milwaukee, WI) and Erie Press Systems (Erie, PA)

QPA-CLIPS, the language and its run-time environment, is a means to directly tie our best understanding of material behavior to the control of material forming or curing processes. This is accomplished by intelligent mapping of sensor information on material behavior to changes in process parameter values. The changed parameter values, in turn, are used to directly drive process equipment .

2.0 The problem

Material forming processes transform a prescribed volume of material(s) into a desired net shape. In curing, material is processed in a way that changes material properties to a set of desired characteristics. In either case, the proper choice of tooling along with a properly defined process cycle, or "recipe", are crucial to the consistent production of quality parts. Variations in material behavior from an expected norm must be accounted for in both the initial conditions of the material and, particularly, the cycle or "recipe" driving the process equipment.

2.1 Controlling a forming or curing process

Attaining a desired final form and material characteristics in forming or curing processes requires effective control of force and heat application. Control of force involves application of mechanical pressure to distort or redistribute a set volume of material within a constraining volume, defined by the tooling (i.e., dies) used in the process.

In the case of forming processes, the material usually requires to be more pliable than its natural state at room temperature. Heat application then becomes an integral part of the pressure application. For curing processes, the application of heat itself, with some pressure being applied in certain cases, is at the core of the processes. In either case, heat control, then, must be *synchronized* with force control and the state of the material must be monitored in-process to determine the right point in the process cycle where heat and/or force application is needed.

One must also take into account the physical limitations imposed by the machinery used for the process. Repeated extreme rates of heat or force application can lead to frequent machine breakdowns and, consequently, make the process economically undesirable.

2.2 "Listening" to the material

In-process monitoring is not the act of collecting streams of historical data for off-line analysis; though this is an important step in creating effective process control. When we collect information during a dialogue, we don't necessarily capture every single word and nuance. Rather, we normally record essential points and supporting information that capture the theme or goal of what is being exchanged.

In a similar way, analyzing process data involves the ability, as in listening, to differentiate between steady or normal process behavior and patterns , or *events*, indicating the onset of a change in material state.

As a simple example, **Figure 1** shows a plot of sensor data over time during a curing process. Point A indicates the onset of the peak, point C, in the sensor data; while B indicates that we are past the region around C. If C is indicator of an optimum material condition for heat or force application, understanding data patterns in A and/or B can be used to trigger that application.

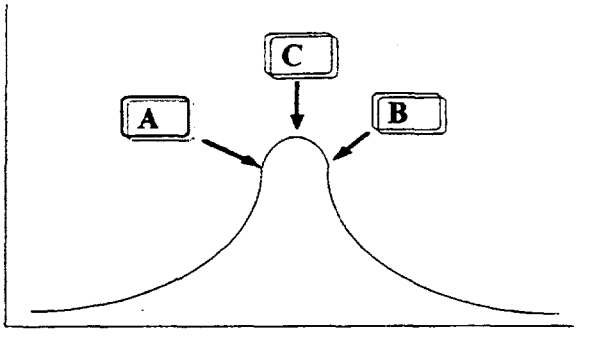


Figure 1

So that, creating a process "recipe" becomes a matter of identifying those key patterns or events, understanding their relationship to heat or force application, and finally linking that application to the occurrence of these events.

3.0 A solution : QPA-CLIPS

QPA-CLIPS, or Qualitative Process Automation CLIPS, is a language with a supporting run-time environment used to describe and run process "recipes" as a set of causal linkages between a specific sensor event(s) and application of a heat or force. The concept of qualitative process analysis was developed as a technique for intelligent control based on interpretation of sensor information [1]. QPA-CLIPS is an implementation of this concept in the CLIPS environment.

3.1 Architecture

The execution model of QPA-CLIPS is basically a traversal across a dependency network. The nodes in that network describe one or more sensor events, their related application actions, and one or more pre-conditions which must be satisfied in order to activate or traverse the node.

Traversal through a node consists of 3 phases:

(1) satisfaction of pre-conditions,

(2) detection of sensor events,

(3) performing required heat or force application.

Transition to the next phase cannot occur until successful completion of the current phase. Once phase 3 is completed, one of the remaining nodes in the network is chosen for traversal using the same 3-phase method. If all nodes have been traversed, traversal is halted. Pre-conditions are either traversal through another node, unconditional (i.e. START node), or the successful diagnostic checks on a sensor. The flow chart in **Figure 2** summarizes the traversal process.

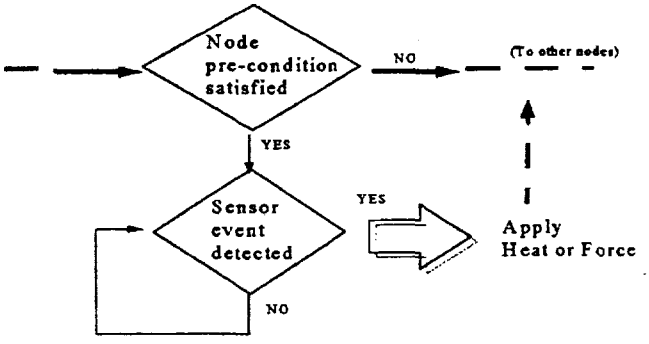


Figure 2

Node traversal is implemented as an interpreter knowledge base exploiting the underlying CLIPS search methods.

There are currently 3 constructs making up the QPA-CLIPS language: TASK, SENSOR, and FILTER. They are defined within the interpreter knowledge base through a set of *deftemplate* 's [2] . The contents of a node are described by the TASK. SENSOR defines sources of raw sensor data; while FILTER applies mathematical functions to SENSOR data. A collection of TASKs describing a dependency network, along with the required SENSORs and FILTERs, is referred to as a *process model* and is consistent with the GRAFCET standard based on Petri Nets [3].

3.1.1 TASK

TASKs encapsulate the pre-conditions for node traversal, sensor event descriptions, and the prescribed heat or force application. In BNF notation form, this translates to:

```
( TASK
    (name <name>)
    (start-when  <pre-condition>)
```

(look-for <sensor-event>)
(then-do <application>))

<name> is a unique label used whenever another TASK refers to this TASK throughout the process model. <pre-condition> is a string type that can be either:

<task-name> COMPLETE,

or

<sensor> OK

or

START

In the first form, the pre-condition becomes the completion or traversal through another node, or TASK. The second form checks completion of a diagnostic check for a sensor. Sensor diagnostics are not currently an explicit construct in QPA-CLIPS. Rather, they are embedded in the sensor I/O functions integrated with CLIPS. The third form of <pre-condition> is makes that TASK the initial traversal point in the dependency network.

<sensor-event> is a string type that can have multiple occurrences of the form:

<source> <direction> <nominal> <tolerance> <repeat>

<source> is the label for a SENSOR or FILTER providing data identifying this event. <direction> can be RISES, DROPS, or HOLDS. The remaining items are numbers describing respectively a threshold value that must be reached, the tolerance band around the nominal value, and the number of times that SENSOR or FILTER data must meet this threshold-and-tolerance- band occurrence without interruption.

So for example, the sensor event clause of

flow RISES 50 0.01 3

translates to data from SENSOR flow must rise at least 0.01 above 50 cc/min. for 3 consecutive times in order to be identified as this sensor event.

<application> is of the form :

<parameter> <amount>

where <parameter> is a label for a process parameter and <amount> is a new value or setting for that parameter. An example is :

PR 5

or set pressure to 5 tons. Application of heat or force then is basically a change in the value of a memory location which, in turn, drives the process.

3.1.2 SENSOR

SENSORs are one of 2 possible sources of data for a sensor event defined within a TASK and defined as follows:

(SENSOR
  (name <name>)
  (max-allowed <value>)
  (min-allowed <value>))

<name> is a unique label used whenever a TASK or FILTER refer to this SENSOR throughout the process model. *max-allowed* and *min-allowed* provide the allowable range of values for the sensor data. As an example, a flowmeter within a process model can be defined as:

(SENSOR
  (name flowmeter)
  (max-allowed 100)
  (min allowed 5))

The rate of the flowmeter can range between 5 and 100 cc/min. Currently , retrieval of sensor data is accomplished through user-defined functions embedded within CLIPS. Association between a SENSOR and these functions is performed through the QPA-CLIPS interpreter. The implementation of SENSOR assumes use of one sensor for the process. This suffices for current applications of QPA-CLIPS. However, for multiple sensor input, the SENSOR construct will be modified to include a reference to a specific function, referred to a the SENSOR source, or:

(SENSOR
  (name <name>)
  (max-allowed <value>)
  (min-allowed <value>)
  (source <function>))

3.1.3 FILTER

FILTERs are the other source of data for a sensor event and is defined as:

(FILTER
  (name <name>)
  (max-allowed <value>)

(min-allowed <value>)
(change <sensor>)
(using <formula>))

<name> is a unique label used by a TASK whenever referring to this FILTER in order to identify a sensor event. Both *max-allowed* and *min-allowed* are used in the same manner as in SENSOR. But, here, they refer to the calculated value created by this FILTER. <sensor> is the label referring to the SENSOR supplying raw sensor data to this FILTER. <formula> is a string describing the combination of mathematical functions used to translate the raw sensor data. so, for example, a formula string of "SLOPE LOG" is the equivalent of:

$$d(\log S)/dt, \quad \text{where S is the raw sensor data.}$$

Future enhancements to the FILTER construct will be the ability for expressing formulas in more natural algebraic terms. So, "SLOPE LOG", for example, will take on a form like dLOG / dt.

3.2 Building a process model

The first step in developing a process model is a through description of how the process runs, what information can be extracted about the process during a cycle run, what sensors are available to extract this information, and how is the sensor information related to application of heat and/or force to the material. These relationships can be investigated through techniques such as Design of Experiments or Taguchi methods. Once they are verified, a dependency network can be build from these relationships in order to specify alternative paths for running the process cycle; depending on the sensor data patterns being extracted. This network can then be encoded as a process model.

As a simple example, a molding process for a panel in a cabinet enclosure is now switching to a new material, requiring that pressure be triggered based on material temperature. For this material, a constant heat setting must be applied throughout the cycle. A sensor, called a thermowidget, was selected to retrieve in-process material temperature. Its operating range is 500°F and 70°F. So, its SENSOR definition is:

(SENSOR
(name thermowidget)
(max-allowed 500)
(min-allowed 70))

Two pressure applications are required by the process: an initial setting to distribute material throughout the mold and a final setting to complete the part. The initial setting must be applied when the material temperature reaches a value of 100°F and the final setting when the rate of change of the material temperature is 0. We first need a FILTER to define the slope of the temperature or:

```
( FILTER
   (name thermoslope)
   (max-allowed 10)
   (min-allowed -10)
   (change thermowidget)
   (using "SLOPE") )
```

We achieve the pressure application through two TASKs:

```
( TASK
   (name first-setting)
   (start-when "START")
   (look-for "thermowidget RISES 100 0.1 5")
   (then-do "PR 2") )

( TASK
   (name final-setting)
   (start-when "first-setting COMPLETE")
   (look-for "thermoslope DROPS 0 0.01 3")
   (then-do "PR 5") )
```

Another TASK can be added to complete the cycle and release the finished part.

4.0 Run-time environment

A process model is executed through a QPA-CLIPS interpreter: a CLIPS knowledge base consisting of the *deftemplate*'s defining the TASK, SENSOR, and FILTER constructs, a rule set which carries out the interpretation cycle, and *deffunction*'s supporting the rule set. Underlying this knowledge base are the user-defined functions integrating the process model with sensors and process equipment.

4.1 The Interpreter

At the core of the interpreter are a set of rules which cycle through the currently available TASKs in a process model and implement the 3-phase method for node traversal mentioned above. When the QPA-CLIPS environment along with the appropriate process model is invoked, control transfers to the TASK containing the pre-condition of START. From that point on, the node traversal method takes over. The QPA-CLIPS interpreter knowledge base can be ported to any CLIPS-compatible platform.

4.2 Sensor and controller integration

Sensors are integrated through user-defined functions embedded within CLIPS. They currently interact with sensor hardware through serial (RS-232-C) communications. These functions are the only platform dependent components within QPA-CLIPS.

Integration of heat and force application is done through a memory mapping paradigm between sensor events and parameter settings where TASK actions, as defined in the

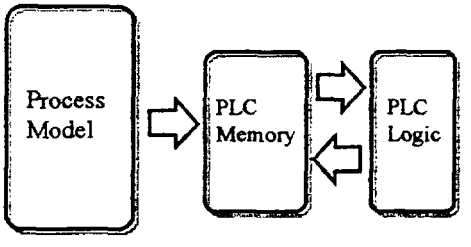


Figure 3

then-do clause, are mapped to the memory location of another control system (i.e. a PLC) which, in turn, directly drives process machinery (see Figure 3). These new settings then trigger signals to heaters or motors driving the equipment.

5.0 Current status

A version of QPA-CLIPS has been demonstrated on a composite molding application. The system consists of a SPARCStation 2 running QPA-CLIPS linked to an Allen-Bradley PLC5/40 driving a molding press. It has successfully created a number of production quality parts for a jet engine.

6.0 Future enhancements

Though the current version of QPA-CLIPS has been demonstrated to work successfully, several opportunities for enhancements have been mentioned in the description of its constructs. In addition, other opportunities for enhancements exist in the development and run-time environments of QPA-CLIPS.

6.1 A GUI for QPA-CLIPS

Currently, a process model is created through the use of a text editor, such as emacs or textedit. The completed model is tested on a simulated run-time environment. Taking a cue from the visual development tools such as those found under Microsoft Windows, a graphical development environment for a process model will greatly ease the development of a process model. The syntax of the QPA-CLIPS constructs are simple enough for

process engineers to work with. Nevertheless, as process models grow in complexity, a need will arise for easy-to-use model navigation and debugging tools seamlessly connected to the simulated run-time environment.

6.2 Automating process model creation

Experience with developing process models has shown that the bulk of the time is actually spent in experimentation; trying to establish the relationships between sensor data and heat/force application points. What is needed to streamline this aspect of the model creation process is the partial or complete automation of the experimentation phase leading to the automatic creation of a process model; since a new material system will require a new process model. The work on DAT-GC by Thompson et al [4] offer a good start in that direction.

6.3 Exploiting fuzzy linguistics

In the definition of the *look-for* clause of a TASK construct, one needs a rather accurate description of not only the goal or threshold value to be reached, but also a tolerance band around that value as well as a repeatability count on the number of times in a row that the event must appear. There are cases where a less exacting or *fuzzy* [5] description will suffice. So, using our example model from 3.2 above, the *look-for* clause for the final pressure application can then be:

(look-for "thermoslope dropping IMMEDIATELY to ZERO")

IMMEDIATELY is a fuzzy term similar to the repeatability count. ZERO, on the other hand, describes a fuzzy term associated with a range of precise values for a SENSOR or FILTER, which can be described with a new QPA-CLIPS construct such as:

```
( BEHAVIOR
   (name <name>)
   (for <source>)
   (range <value-list>) )
```

where <name> is the fuzzy term, <source> is the label for the SENSOR or FILTER, and <value-list> contains the values defining the membership function [5] for this BEHAVIOR.

6.4 Exploiting parallelism

It goes without saying that node traversal within a process model is inherently a method that can easily converted to parallel processing. Having a parallel version of CLIPS operating in an affordable parallel processing platform will enable QPA-CLIPS to easily operate in a process control scenario requiring multiple sensors.

7.0 Conclusions

QPA-CLIPS, an extension of CLIPS for process control, is a proven tool for use by process engineers in developing control models for forming or curing processes. Its simple syntax and integration into CLIPS provides a powerful, yet straightforward, way to describe and apply control of a process driven by sensor events, or distinct patterns in sensor data. Several enhancements have been suggested and currently strong interest exist in the automatic generation of process models and fuzzifying the description of sensor events.

References

[1] Park, Jack, **Toward the development of a real-time expert system**, 1986 Rochester FORTH Conference Proceedings, Rochester NY, June 1986, pp. 23-33

[2] CLIPS Reference Manual, Vol. I, **Basic Programming Guide**, CLIPS Version 5.1, Sept. 1991

[3] David, Rene and Hassane Alla, **Petri nets for modeling of dynamic systems - a survey**, *Automatica*, Vol. 30, No. 2, pp.175-202, 1994

[4] Levinson, Richard, Peter Robinson, and David E. Thompson, **Integrated perception, planning and control for autonomous soil analysis**, Proc. CAIA-93

[5] Terano, Toshiro, Kiyoji Asai, and Michio Sugeno, **Fuzzy Systems Theory and its applications**, Academic Press, 1992

omit

Session 2B: Fuzzy Logic, Neural Networks, and Program Understanding

Session Chair: Bob Shelton

Fuzzy Expert Systems Using CLIPS

Thach C. Le
The Aerospace Corporation
Los Angeles, California

Abstract

This paper describes a CLIPS-based fuzzy expert system development environment called FCLIPS and illustrates its application to the simulated cart-pole balancing problem. FCLIPS is a straightforward extension of CLIPS without any alteration to the CLIPS internal structures. It makes use of the object-oriented and module features in CLIPS version 6.0 for the implementation of fuzzy logic concepts. Systems of varying degrees of mixed Boolean and fuzzy rules can be implemented in FCLIPS. Design and implementation issues of FCLIPS will also be discussed.

I. Introduction

Production systems, better known as data-driven rule-based expert systems, attempt to encode human problem-solving knowledge. The architecture of production systems originated from research in cognitive psychology which observed that human behavior tends to be modularized and reactive in nature [1]. Consequently, the production system architecture was designed to allow for modularized and independent pieces of knowledge to be encoded in the form of if-then rules or production rules. One of the first implementations of a production system language is OPS5 (Official Production Systems) which is based on LISP. The Software Technologies Group of NASA at Houston, Texas later reimplemented a similar language called CLIPS (C-Language Integrated Production System) in C mainly to increase speed and portability. Because of CLIPS' flexibility and source code availability, it quickly became popular among expert system developers. Since its introduction several years ago, its has been frequently enhanced to embrace the latest advances in software technology.

Fuzzy systems, based on fuzzy set theory, also seek to encode human problem-solving knowledge. One of the most important contribution of fuzzy logic to the field of heuristic systems is the introduction of the concept of linguistic variables [5]. Linguistic variables are linguistic terms that are used to represent qualities of objects or concepts as in natural languages. In fuzzy systems, linguistic variables are represented by a mathematical function from which a value can be derived to indicate the similarity of an object to the ideal form. The implication is rather significant: instead of modeling the real world as a discrete world, linguistic variables allow a whole spectrum of continuous possibility to be modeled. For example, instead of speaking about a world of discrete terms such as circles, squares, and triangles, one can use linguistic variables to describe a larger set of the objects in terms of the degree to which an object is similar to circles, squares or triangles. In addition to linguistic variables, there is a set of operations in a fuzzy system. In the last few years, fuzzy systems have proven their effectiveness in solving difficult problems, particularly in the area of control [5]. Several commercial hardware and software tools have been introduced recently to build fuzzy systems.

The similar and complementary nature of fuzzy systems and expert systems have naturally suggested the idea of fuzzy expert systems in which the traditional Boolean production system architecture is extended to include the concepts of fuzzy systems. By combining the proven techniques of production systems with the fuzzy systems' representation and manipulation power, we believe the resulting system can effectively address even more challenging AI problems. Since the CLIPS source code is available at almost no cost, there have been attempts to extend CLIPS to a fuzzy expert system development environment. The most notable examples are the FuzzyCLIPS from the Knowledge Systems Laboratory of the National Research Council of Canada [6], and, also using the same name, the FuzzyCLIPS by Togai Infralogics Inc. [7]. In this paper, we will focus on our own implementation. The similarities and differences of these CLIPS-based systems will be discussed at a later time. An important quality of FCLIPS is the simplicity of its design, while still maintaining the ability to support a wide range of fuzzy expert systems.

In the following sections, we will describe the architecture of a typical Boolean production system exemplified by CLIPS, the architecture of a fuzzy system, the design and implementation issues of FCLIPS, the FCLIPS application to the cart-pole balancing problem, and future FCLIPS extensions.

II. Production Systems

Production systems are based on the principles of modularity and contextual dependent reaction in human behavior. A typical production system has three distinct modules, a *production memory* for containing production rules, a *working memory* for facts, and an *inference engine* for execution [1]. The production memory contains rules which have the form of:

if <antecedent> then <consequent>

An example of a CLIPS rule is as follows:

```
(defrule My_House
        (the house number is 123)
        (the street name is Pennsylvania)
=>
        (printout t "It is my house"))
```

Antecedent, also called the rule's left hand side (LHS), contains Boolean predicates that state the conditions for the actions to take place as specified in the consequent, or the right hand side (RHS). A rule becomes a candidate for execution if its antecedent is satisfied by facts in the working memory.

The working memory contains facts which describe the current state of a situation or a problem. The set of facts in the working memory is often changed as new data arrives or as the result of a rule being executed at each inference cycle.

The inference engine decides which rule is to be executed given the current set of facts in the working memory. It first performs matching between rules and existing facts. In CLIPS this matching is implemented using Rete net [2], known as the best pattern matching algorithm. After matching, there may be more than one matched rule. The set of matched rules is called the *conflict set.* Only one rule from the conflict set will be selected to be executed via the process called *conflict resolution.* Once a rule is selected, its RHS is

executed which often affects the working memory by adding, modifying or deleting facts from the working memory. As the result of the working memory being changed, a new conflict set is produced. The cycle goes on until the conflict set is empty or a halt instruction is encountered.

The current version of CLIPS provides several powerful features to the basic architecture of production systems. It has a built-in object-oriented system which greatly enhances its representation capability. Instead of using attribute-value predicates as in OPS5 to model problems, one can now use classes and objects, which are more natural and expressive.

Another important feature of CLIPS is modularity [8]. Rules that share some commonality can be grouped into a module. Modules allow the implementation of context-switching. At any time, only one module can be active and only its rules participate in the matching and selection process. The concept of modularity allows more efficient organization of the production memory and improves the scalability of a production system in terms of its scope and speed [3].

In general, production systems are quite effective in providing heuristic solutions to problems. They have been used in many real-world problems [5]. But while object-orientation greatly improved the representation capability of production systems, the rule predicates are still constrained to only Boolean statements which are not always natural to model real-world problems. This is an area where fuzzy systems offer an attractive alternative.

III. Fuzzy Systems

Fuzzy systems are based on fuzzy set theory [4]. They are similar to Boolean production systems in their use of if-then rules. However, they are different in two major aspects. First, the LHS and RHS predicates of a fuzzy rule are fuzzy statements, not Boolean statements. Secondly, fuzzy rules whose RHS have linguistic variables describing the same subject, are all executed and collectively contribute to the final conclusion. In a Boolean production system, rules compete to be solely executed. Example 1 shows fuzzy expert system rules:

```
Rule_1:   if    Temperature is Hot
                Pressure is High
          then
                set Change to Large

Rule_2:   if    Temperature is Cold
                Pressure is Medium
          then
                set Change to Small
```

Example 1: Fuzzy rules for a fuzzy system

The terms "Hot", "Cold", "High", "Medium", "Large" and "Small" are linguistic variables. A linguistic variable describes the quality of a concept or an object in term of *degree of truth*. The degree of truth is defined by a *membership function* that maps the domain of a concept into a degree of truth between 0 and 1. Figure 1 shows examples of membership functions of Temperature linguistic variables.

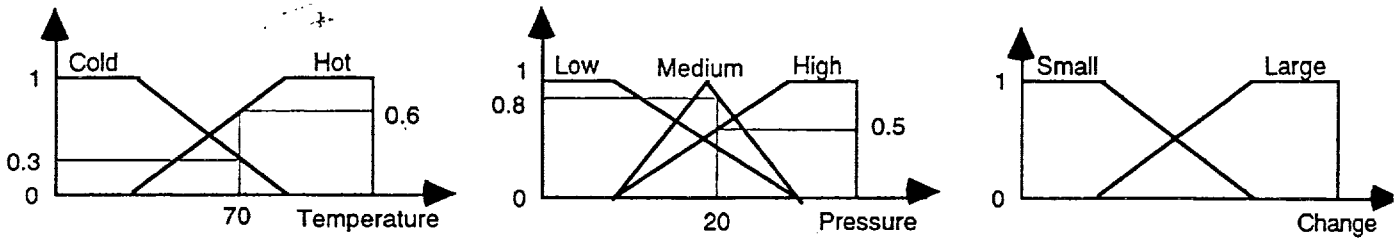


Figure 1: Membership functions for Temperature, Pressure and Change

Given a temperature value of 70, and a pressure value of 20, the statement "Temperature is Hot" in Rule 1 is evaluated to a truth value of .6. Similarly, "Temperature is Cold" is evaluated to .3, "Pressure is High" to .5, and "Pressure is Medium" to .8.

The typical method of evaluating a rule is by taking the minimum of the truth values of the LHS of the rule and applying an alpha-cut [5] to its RHS. The evaluations of Rule 1 and Rule 2 are shown in Figure 2. The results from both rules are combined using the *maximum envelop operation* and then defuzzified to give the final answer. There are several methods for defuzzification such as the centroid or the mean-of-max methods [5]. Figure 2 shows the defuzzification of the variable Change using the centroid method.

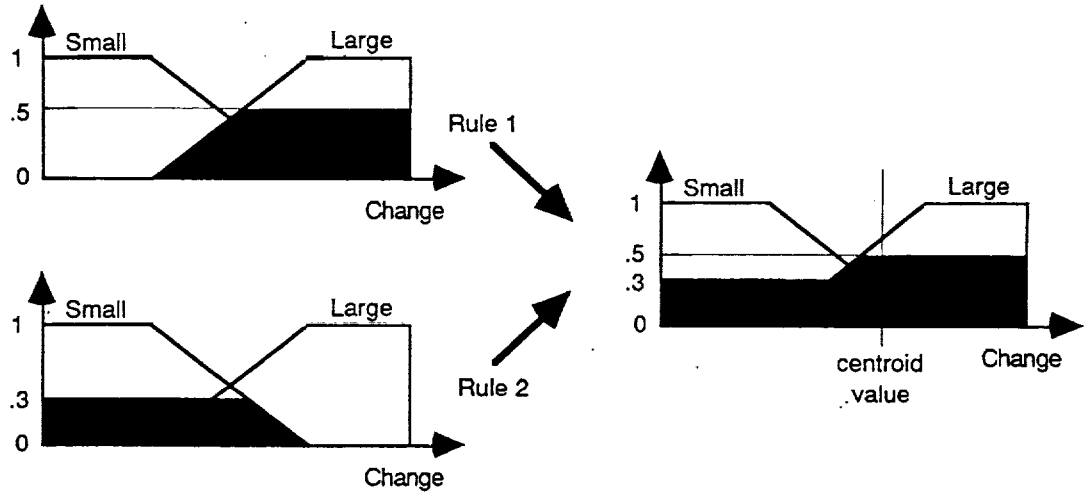


Figure 2: MIN-MAX operation and defuzzification using centroid method

As new values of Temperature and Pressure become available, the evaluation cycle repeats and a new value for Change is generated.

Fuzzy system technology has been applied successfully to many difficult control problems. It offers a straightforward way to encode human experience of solving real control problems into machine automation. The features of linguistic variables and rule contribution from fuzzy systems have proven to be very effective in modeling and solving many problems for which the continuity of variables involved is critical.

There have been several commercial tools developed to facilitate the implementation of fuzzy systems. However, they often allow only rules of common consequent and therefore, small in number. This small and monolithic characteristic of current fuzzy system implementations prevents them from addressing problems that require deep reasoning such as those in the areas of diagnosis, planning, or design.

Combining the best features of fuzzy systems and Boolean production systems will create a general system that can address more effectively a greater range of problem domains. Such systems are called fuzzy expert systems. In the following sections we will describe our fuzzy system extensions to CLIPS, including design and implementation issues.

IV. Implementation of Fuzzy Expert Systems in CLIPS

Our goal is to simply extend CLIPS to allow linguistic variables to be used in rules and to provide the necessary fuzzy operations to support them. The addition of support for linguistic variables in CLIPS is straightforward and does not pose any conflict with the original philosophy of production systems. However, the conflicting approaches of rule competition in Boolean production systems and rule cooperation in fuzzy systems needs to be resolved in a way that does not violate the original philosophies of either technique. Our approach to the CLIPS extensions is to preserve the integrity of CLIPS as much as possible and to only introduce new constructs when necessary.

Linguistic Variable and Operations

A linguistic variable is defined by a membership function. We implement a membership function as a CLIPS class called MFunc. A membership function can then be defined as an object instance of the class MFunc. Its value is specified as a piece-wise linear function which is a sequence of (x,y) coordinate pairs in the form (x1 y1 x2 y2 .... xn yn). For example, the Hot function is specified as follows:

```
(make-instance Hot of MFunc (Func 0 0 10 1 50 0))
```

Once the linguistic variable is defined, a value, for example, 8, can be tested against the membership function to give the corresponding degree of truth, or truth value, using the function "deg".

```
(deg 8 Hot)
```

The above statement will return a truth value of .8. Another very useful function is the Boolean function "is" which checks out if a value is within the domain of a membership function. It returns true if the value is within the domain and false otherwise. For examples,

(is 8 Hot) returns true, and (is -100 Hot) returns false.

In addition to the class MFunc, a fuzzy variable is represented by the class FVar. For example, "Change", an object instance of the class FVar, is first defined as follows:

```
(make-instance Change of FVar)
```

A fuzzy variable can have a value of nil or of a piece-wise linear function. This value can be combined in the MIN-MAX fashion with another linear piece-wise function to yield a new value.

Using the function "is" in the LHS, the fuzzy rule Rule_1 in section III can be rewritten in the CLIPS defrule construct as follows:

```
(defrule Rule_1
        (Temperature ?x&:(is ?x Hot))
        (Pressure ?y&:(is ?y High))
=>
        (bind ?m (fmin ?x Hot ?y High))
        (set Change Large ?m))
```

Since the LHS only contains Boolean predicates, it fits into the CLIPS defrule construct. Truth values of Temperature and Pressure are not evaluated until the rule is executed. This is designed to avoid unnecessary evaluations during the rule matching stage. If the rule is chosen for execution, the MIN-MAX operation is performed as specified in the RHS.

The function "fmin" returns the *minimum* value of the truth values in the LHS. The function "set" then uses this value to perform an alpha-cut on the membership function Large, which is then combined with the current value of the fuzzy variable Change using the *maximum* envelop operation.

With a few new classes and functions, a fuzzy rule can now be written in CLIPS format. The next section discusses issues related to the interaction of multiple fuzzy rules.

Fuzzy Processing

When multiple fuzzy rules exist in the production memory, a fuzzy cycle has to be implemented so that the contributing results to fuzzy variables from each rule can be integrated into a single value. We achieve this goal in FCLIPS by having a fuzzy inference rule of lower priority or salience value which is executed at the end of each fuzzy cycle. The set of fuzzy rules in section III can be rewritten as follows:

```
(defmodule Fuzzy_Module

        (defrule Rule_1
                (Temperature ?x&:(is ?x Hot))
                (Pressure ?y&:(is ?y High))
        =>
                (bind ?m (fmin ?x Hot ?y High))
                (set Change Large ?m))

        (defrule Rule_2
                (Temperature ?x&:(is ?x Cold))
                (Pressure ?y&:(is ?y Medium))
        =>
                (bind ?m (fmin ?x Cold ?y Medium))
                (set Change Small ?m))

        (defrule Fuzzy_Engine
                (declare (salience -10))
```

```
        ?f1 <- (Temperature ?x)
        ?f2 <- (Pressure ?y)
=>
        (retract ?f1 ?f2)
        (<check stopping condition> ?x ?y)
        (<defuzzify, apply and reset> Change)
        (<get new values for Temperature & Pressure>))
)
```

Example 2: A CLIPS-based fuzzy module

During a fuzzy cycle, the fuzzy variable Change accumulates the results from rules executed. At the end of the cycle, the Fuzzy_Engine rule is executed. Change is typically defuzzified and the result is asserted into the working memory or sent to an external environment.

To avoid potential interference from other rules in the production memory, this set of rules can be conveniently grouped into a CLIPS module. With a few extra fuzzy related classes and functions, CLIPS is now capable of supporting traditional fuzzy systems as described in section III. There are no changes necessary to the conflict resolution or rule matching modules of CLIPS. Yet, a new kind of mixed systems is also now possible. In these systems, not only can the LHS of a rule have any number of mixed Boolean or fuzzy predicates, but the RHS can also. This is a major improvement from traditional fuzzy systems whose rules' RHS have to be about the same subject.

Of course, due to the differences in the nature of production systems and fuzzy systems, there are several issues that need to be addressed for systems that allow mixed rules: (1) how to prioritize mixed rules in conflict resolution, (2) how to group fuzzy rules of different RHS' into fuzzy modules, (3) how to differentiate fuzzy cycles and Boolean cycles, and (4) how to propagate fuzzy conclusions. To illustrate these issues more clearly, we will use the examples of the following rules. The assumed truth values of antecedents are given on the right.

			truth values
rule 1:	if	(A > 10)	1
		(B is Small)	.3
	then		
		(set C to Large)	
		(set D to Medium)	
rule 2:	if	(A is Large)	.7
		(D is Small)	.3
	then		
		(set C to Medium)	
		(set E to Small)	
rule 3:	if	(A > 25)	1
		(X > 25)	1
	then		
		(set E to Large)	

Example 3: Mixed rules with different truth values

For the first issue, based on the truth values alone, how should these rules be prioritized? We argue that they all should have the same priority despite the fact that the truth values for some fuzzy predicates are less than others. The fuzzy statement "B is Small" with truth value of .3 is no less significant than "A is Large" with truth value .7, because truth values in fuzzy logic are not the same as uncertainty values which have been used as a criteria for conflict resolution in some systems. "B is Small" with truth value .3 is a description with complete certainty of how small B is, which is indicated by the truth value of .3. Besides, in traditional fuzzy systems, a small truth value of a fuzzy predicate in a rule does not affect the rule's conclusion validity, but only the actual value of the conclusion. Consequently as long as a fuzzy predicate has a truth value of greater than 0 in FCLIPS, it is considered equivalent to a Boolean statement of value true. This interpretation is particularly significant in that it does not require extra processing for mixed rules during the rule matching stage.

The second issue points to the fact that traditional fuzzy systems only allow rules that address the same subject on their RHS. In the example above, there is no obvious way to group these three rules into fuzzy modules since their RHS' are not exactly about the same subjects. Rule 1's RHS is about C and D, rule 2 C and E, and rule 3 E. It turns out that we don't have to group them, as the following discussion will show.

The third issue concerns the interpretation of the significance of a fuzzy rule, a Boolean rule and a mixed rule. This interpretation is important for determining how conflict resolution should be changed. There are two interpretations for the significance of fuzzy rules and fuzzy modules versus a Boolean rule. In the first interpretation, a fuzzy module is considered equivalent to a Boolean rule in a Boolean production system, since all the rules in a fuzzy module can be thought of as being executed in parallel and contributing to a single conclusion. This interpretation is valid and in common use [7]. It can already be implemented in FCLIPS by using the existing defmodule construct to group fuzzy rules of the same module, as shown in Example 2.

In the second interpretation, all three kinds of rules are equivalent and compete for execution on the same grounds. This interpretation is intuitive and less restrictive, but it poses a question about when a fuzzy cycle starts and stops, i. e. when is a fuzzy variable defuzzified? In FCLIPS, this question of fuzzy cycle is resolved by introducing rules of higher salience value specifying defuzzifying conditions for each fuzzy variable mentioned in the rules' RHS. An example of a defuzzifying rule for the fuzzy variable C in Example 3 is as follows:

```
Rule DefuzzifyingC:
        if      (declare (salience 10))
                (TimeStamp > 10 )
                (C not equal nil)
        then
                (<defuzzifying, apply and reset> C)
```

There are similar defuzzification rules for D and E. The second issue concerning how to group rules into fuzzy modules is no longer an issue, because there is no fuzzy module, only defuzzifying conditions for fuzzy variables in a rule consequent. Both interpretations are implementable under FCLIPS. The first is more traditional and of common use. The second is more general and interesting.

And finally, the fourth issue concerns the multiple-step reasoning where conclusions reached remain in fuzzy form to match the rules in the following cycles. By avoiding

defuzzification in intermediate reasoning steps, less information will be lost. Given the following two rules,

Rule 1:	if	(A is Small)
	then	(B is Large)
Rule 2:	if	(B is Small)
	then	(C is Medium)

assume that Rule 1 is executed and generates a fuzzy value X for "B is large". The truth value for "B is Small" in rule 2 will then be calculated as [15]:

truth value = 1 - MeanSquareError (X, Small)

This operation is not yet (but will be soon) implemented in FCLIPS. Currently, B has to be defuzzified after rule 1 execution before being matched into rule 2 antecedent.

In summary, FCLIPS is an extension to CLIPS to provide a fuzzy expert system development environment. Despite its simplicity, it is capable of supporting a wide range of fuzzy expert systems. An FCLIPS implementation has been fully prototyped and successfully tested in CLIPS code. It is currently being ported to C. In addition, a ToolTalk interprocess communication facility was added to FCLIPS to communicate with external processes. In the next section, an example using FCLIPS is described.

V. Example: The Cart-Pole Balancing Problem

The Cart-Pole Balancing problem, also called the Inverted-Pendulum problem, is a popular test case for fuzzy systems. In this section, we describe the structure and key components of a FCLIPS program that solves this problem. The problem is stated as follows:

A cart is moveable along 1 dimension with a pole on top as shown in Figure 3. Its state variables, which include the pole *angle*, the pole *angular velocity*, the current *position* and *velocity* of the cart, are known. Calculate the appropriate force to apply to the cart to keep the pole balanced.

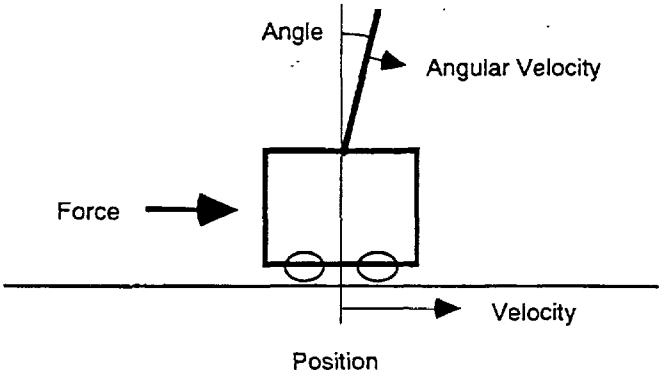


Figure 3: Cart-Pole Balancing problem

First, a function called "model" that simulates the dynamics of the cart-pole system is defined. It takes the applied force and the current state variables as arguments and generates a new set of state variables.

```
(deffunction model (?force ?polePosition ?poleVelocity
                                ?cartPosition ?cartVelocity) ...)
```

Six membership functions for the pole angle or pole position, A_NegMed, A_NegSmall, A_Zero, A_PosSmall, A_PosMed, and A_VerySmall, are defined:

```
(make-instance A_NegMed    of MFunc (Func -50 0 -20 1 -15 1 -10 0))
(make-instance A_NegSmall  of MFunc (Func -20 0 -10 1 0 0))
... (more make-instance's)
```

And similarly, for Angular Velocity: Av_NegSmall, Av_Zero, Av_PosSmall, and Av_VerySmall; for Distance: D_Neg, D_Zero, D_Pos, D_VerySmall; for Velocity: V_Neg, V_Zero, V_Pos, D_VerySmall; and for Force: F_NegMed, F_NegSmall, F_NegVerySmall, F_Zero, F_PosVerySmall, F_PosSmall, and F_PosMed.

The initial configuration of the cart-pole system is defined in the startup rule. The values can be changed for testing purposes.

```
(defrule startup
=>
        (assert (PolePosition 5))
        (assert (PoleVelocity 0))
        (assert (Position 0))
        (assert (Velocity 0)))
```

There are 9 rules devised to balance the pole position:

```
(defrule Rule1
        (PolePosition ?x&:(is ?x A_PosMed))
        (PoleVelocity ?y&:(is ?y Av_Zero))
=>
        (bind ?m (fmin ?x A_PosMed ?y Av_Zero))
        (set Force F_PosMed ?m))

(defrule Rule2
        (PolePosition ?x&:(is ?x A_PosSmall))
        (PoleVelocity ?y&:(is ?y Av_PosSmall))
=>
        (bind ?m (fmin ?x A_PosSmall ?y Av_PosSmall))
        (set Force F_PosSmall ?m))

... ( more defrules' )
```

And one deffuzzification rule:

```
(defrule engine
        (declare (salience -10))
        ?f1 <- (PolePosition ?x)
        ?f2 <- (PoleVelocity ?y)
        ?f3 <- (Position ?z)
```

```
        ?f4 <- (Velocity ?w)
=>
        (retract ?f1 ?f2 ?f3 ?f4)                          ; delete old facts
        (bind ?torq (centroid (send [Force] get-Val)))     ; defuzzification
        (bind ?vals (model ?torq ?x ?y ?x ?w))             ; new values
        (assert (PolePosition (nth$ 1 ?vals)))             ; create new facts
        (assert (PoleVelocity (nth$ 2 ?vals)))
        (assert (Position (nth$ 3 ?vals)))
        (assert (Velocity (nth$ 4 ?vals))))
```

The function "centroid" performs defuzzification using the centroid method [5]. The new values can be sent to graphs, files or simulation. In our testing, the results were sent via ToolTalk to a separate graphic simulation process for displaying the cart-pole system behavior. After some tuning of the membership functions, the system was able to balance the pole.

Conclusions

FCLIPS is a simple and straight-forward implementation of a fuzzy expert system development environment based on CLIPS. The CLIPS object-oriented and modularity features were exploited to implement fuzzy logic concepts into CLIPS. Despite the simplicity of FCLIPS, it can be used to develop systems containing a wide range of mixed Boolean and fuzzy rules. With FCLIPS, fuzzy expert system solutions can be introduced to problems which typically require deep reasoning such as those in the areas of diagnosis, planning, or design.

Bibliography

[1] Brownston, L., Farrell, R., Kant, E., and Martin, N., Programming Expert Systems in OPS5, Addison Wesley Publishing Company, 1985.

[2] Forgy, C., "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem," Artificial Intelligence, vol. 19, 1982, p. 17-37.

[3] Le, T., Homeier, P., "Portable Inference Engine: An Extended CLIPS for Real-time Production Systems", Space Operations Automation and Robotics (SOAR '88), Dayton OH, July 20-23, 1988, Proceeding p. 187-192.

[4] Graham, I. and Jones, P., *Expert Systems - Knowledge, Uncertainty and Decision*, Chapman and Hall, 1988.

[5] Bezdek, J.C., Ruspini, E., Zadeh, L.A., et al, *Fuzzy Logic Inference Systems*, Short Course Notes, Intelligent Inference Systems Corp., 1993.

[6] *FuzzyCLIPS Version 6.02 User's Guide,* Knowledge Systems Laboratory, Institute for Information Technology, National Research Council of Canada. May 1994.

[7] Teichrow, S. J., Horstkotte, R. E., Fuzzy-CLIPS, The C Language Integrated Production System with Fuzzy Logic Capability. NAS9-18335, August 17, 1990.

[8] CLIPS Reference Manual, CLIPS Version 6.0, Software Technology Branch, Lyndon B. Johnson Space Center, June 2nd, 1993.

S12-63
34072
p. 13

N95-19637

NEURAL NET CONTROLLER FOR INLET PRESSURE CONTROL OF ROCKET ENGINE TESTING

Luis C. Trevino
NASA-MSFC, Propulsion Laboratory
Huntsville, Alabama

ABSTRACT

Many dynamic systems operate in select operating regions, each exhibiting characteristic modes of behavior. It is traditional to employ standard adjustable gain PID loops in such systems where no apriori model information is available. However, for controlling inlet pressure for rocket engine testing, problems in fine tuning, disturbance accommodation, and control gains for new profile operating regions (for R&D) are typically encountered [2]. Because of the capability of capturing i/o peculiarities, using NETS, a back propagation trained neural network controller is specified. For select operating regions, the neural network controller is simulated to be as robust as the PID controller. For a comparative analysis, the Higher Order Moment Neural Array (HOMNA) method [1] is used to specify a second neural controller by extracting critical exemplars from the i/o data set. Furthermore, using the critical exemplars from the HOMNA method, a third neural controller is developed using NETS back propagation algorithm. All controllers are benchmarked against each other.

I. INTRODUCTION

An actual propellant run tank pressurization system is shown in Figure 1.1 for liquid oxygen (LOX). The plant is the 23000 gallon LOX run tank. The primary controlling element is an electro-hydraulic (servo) valve labeled as EHV-1024. The minor loop is represented by a valve position feedback transducer (LVDT). The major or outer loop is represented by a pressure transducer (0-200 psig). The current controller is a standard PID servo controller. The reference pressure setpoint is provided by a G.E. Programmable Logic Controller. The linearized state equations for the system are shown below:

$$\dot{x}_1 = x_2 - (0.8kg + c)x_1 \qquad (1.1)$$

$$\dot{x}_2 = 5kg\ au - (0.8kg\ c + d)x_1 + x_3 \qquad (1.2)$$

$$\dot{x}_3 = 5abkg\ u - (0.8kg\ d + f)x_1 + x_4 \qquad (1.3)$$

$$\dot{x}_4 = -0.8kg\ fx_1 \qquad (1.4)$$

where kg=1, servo valve minimum gain. Based on previous SSME test firings, the average operating values for each state

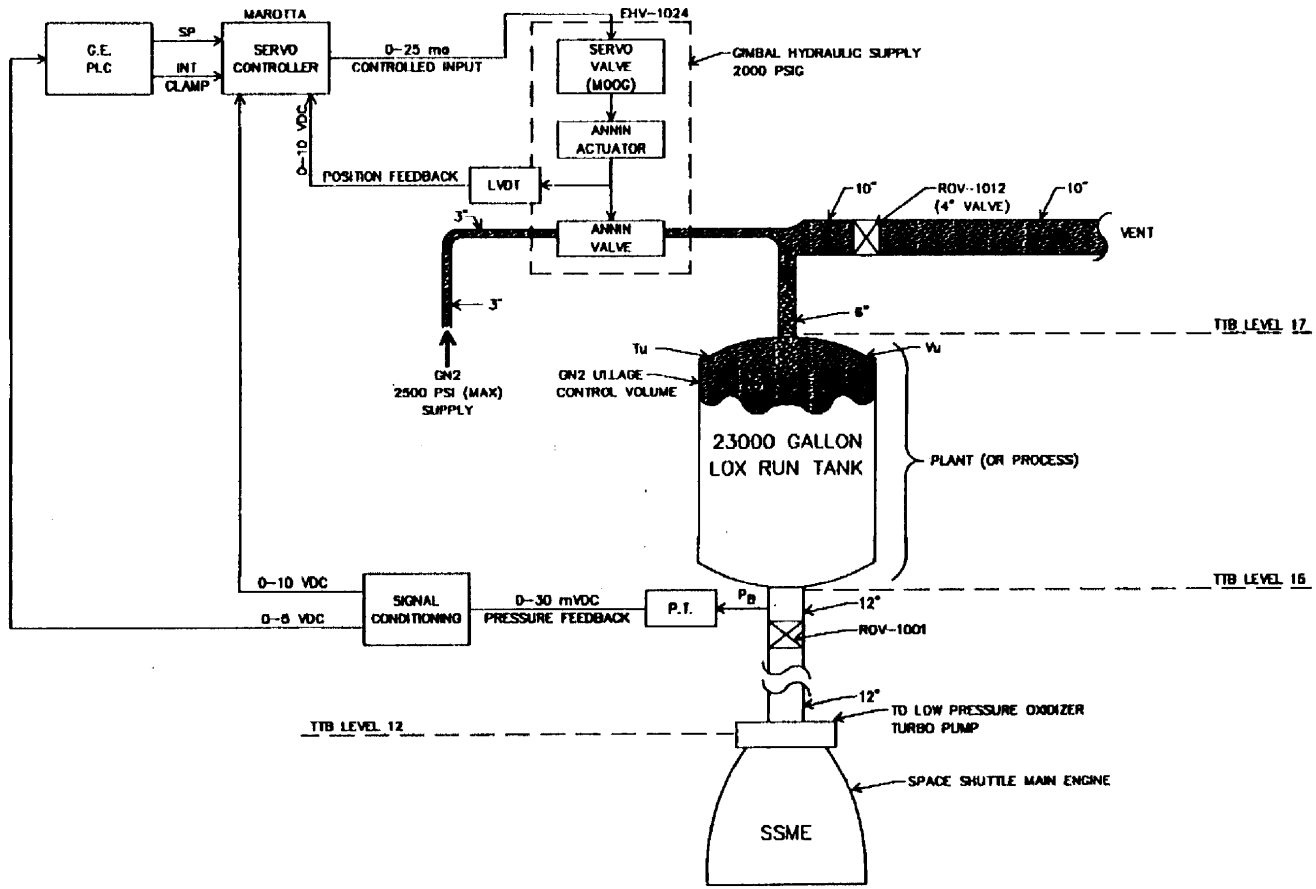


Figure 1.1 Technology Test Bed
Propellant Lox Run Tank Pressurization System

variable are determined to be

x_1 = P_B:0-76 psig x_3 = V_u:250-350 ft^3

x_2 = T_u:150-300 R^o x_4 = L:0-1 inch

where P_B = bottom tank pressure

T_u = ullage temperature

V_u = ullage volume

L = valve stem stroke length

Using those ranges, the following average coefficients are algebraically determined:

a = 120.05 d = 5995.44

b = 89.19 f = 14.70

c = 214.30

II. Methodology

1. Using a developed PID-system routine from [2], an i/o histogram is established in the required format per [1] for a select cardinality of 300. Figure 2.1 portrays the scheme. A ramp control setpoint signal (from 0-120 psig) served as the reference trajectory. The input portion of the histogram is selected to be a five dimensional (300x5) matrix, four successive delayed samples and the current sample. The output portion is a one dimensional (300x1) vector. Therefore, the i/o histogram is simply represented by a 300x6 matrix.

2. Using the captured i/o data set and NETS back propagation algorithm, a neural network is next established with a 5-10-10-1 architecture. The trained network is next simulated as the controller for the system. Figure 2.2 illustrates the simulation scheme.

3. Using a developed HOMNA (KERNELS) algorithm [1], a reduced training i/o set is specified. The input portion of the set, "S", will provide the mapping of real time system inputs to the neural net controller (NNC). The output segment of the set is represented by the last column vector of the i/o set.

4. After configuring the reduced i/o set into the needed formats, using MATLAB, the gain equation (2.1) is executed.

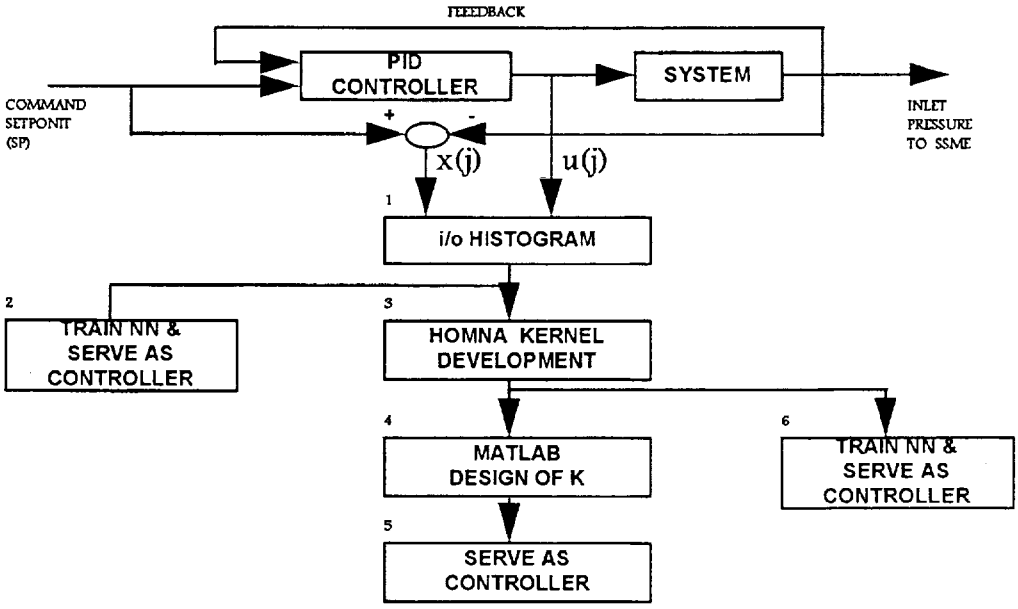


Figure 2.1 Scheme For Building i/o HIstogram and Training Set

$$K = YG^{-1} = Y\Psi(SS^*) \qquad (2.1)$$

where K = neural gains (row vector) for single neural layer
Y = NNC controller output signature row vector
S = established matrix set of step 3
Ψ = any (decoupled) operation: exponential, etc.

For this project, Ψ was identical with that used in the literature of [1], namely the exponential function. "K" serves as a mapping function of the input, by way of "S", to the NNC output, u(j). Here, u(j) serves as control input to the system and is determined by equation (2.2) [1].

$$u(j) = K\Psi(Sx(j)) \qquad (2.2)$$

where x(j) is the vector input. In accordance with the dimensions of the i/o histogram, a five dimensional input is used and is accomplished using successive delays. Namely, a typical input for any given sample is represented by

$$[x(j)\ x(j-1)\ x(j-2)\ x(j-3)\ x(j-4)]$$

The overall HOMNA scheme is embedded in the neural controller block of Figure 2.2 as a 5-5-1 architecture (single hidden layer).

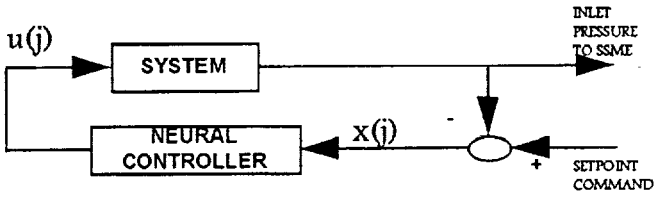


Figure 2.2. Simulation Scheme for NNC and System

5. For select cases to be presented, integral control for the HOMNA system was presented according to the following scheme of [1].

$$\Delta = \frac{1}{N}\sum[y(j)-\bar{Y}(j)] \qquad (2.3)$$

where N = window (sampling) size
$y(j)$ = current system output
$\bar{y}(j)$ = desired output, or command setpoint, sp

6. Using the training i/o set of part 3, a separate neural network controller is established, again using NETS. The simulation scheme is similar to that of part 2.

7. PID system response plots are generated for a further comparitive analysis.

III. **RESULTS**

Table I. Case Summary

Case	System	Setpoint Command	Noise	Integral Control
1	Neural	Ramp	no	n/a
2	HOMNA	Ramp	no	None
3	HOMNA	Ramp	no	None
4	PID	Ramp	no	Present
5	Neural	Ramp	yes	n/a
6	HOMNA	Ramp	yes	Present
7	PID	Ramp	yes	Present
8	Neural	Profile	no	n/a
9	HOMNA	Profile	no	None
10	HOMNA	Profile	no	Present
11	PID	Profile	yes	Present
12	Neural	Profile	yes	n/a
13	HOMNA	Profile	yes	None
14	HOMNA	Profile	yes	Present
15	Neural	Ramp	no	n/a

* Case for procedural step 6

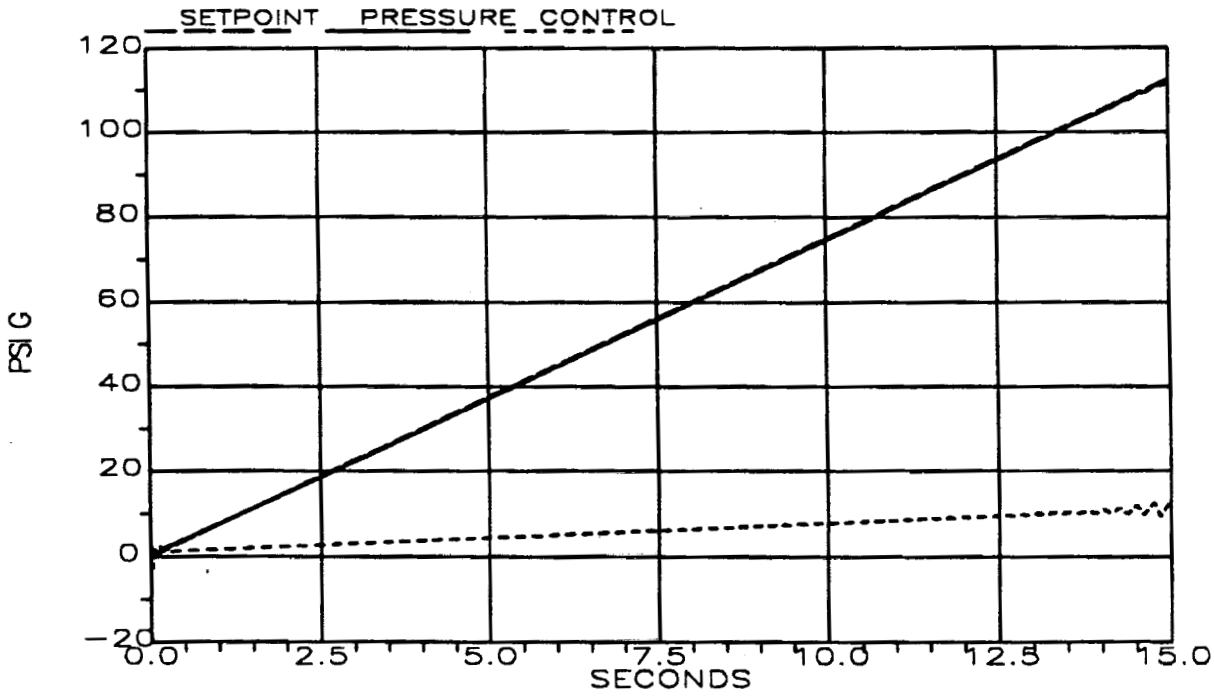


Figure 3.1 Case 1, 3, and 4 Simulation Results

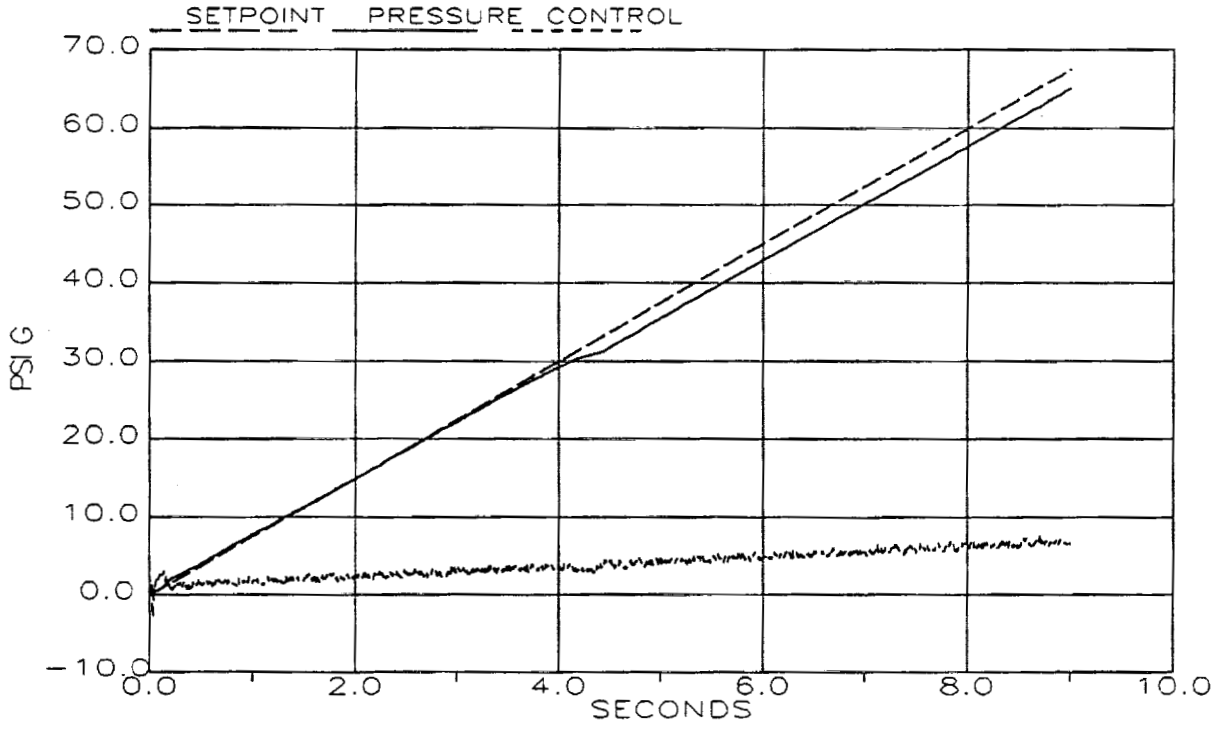


Figure 3.2 Case 2 Simulation Results

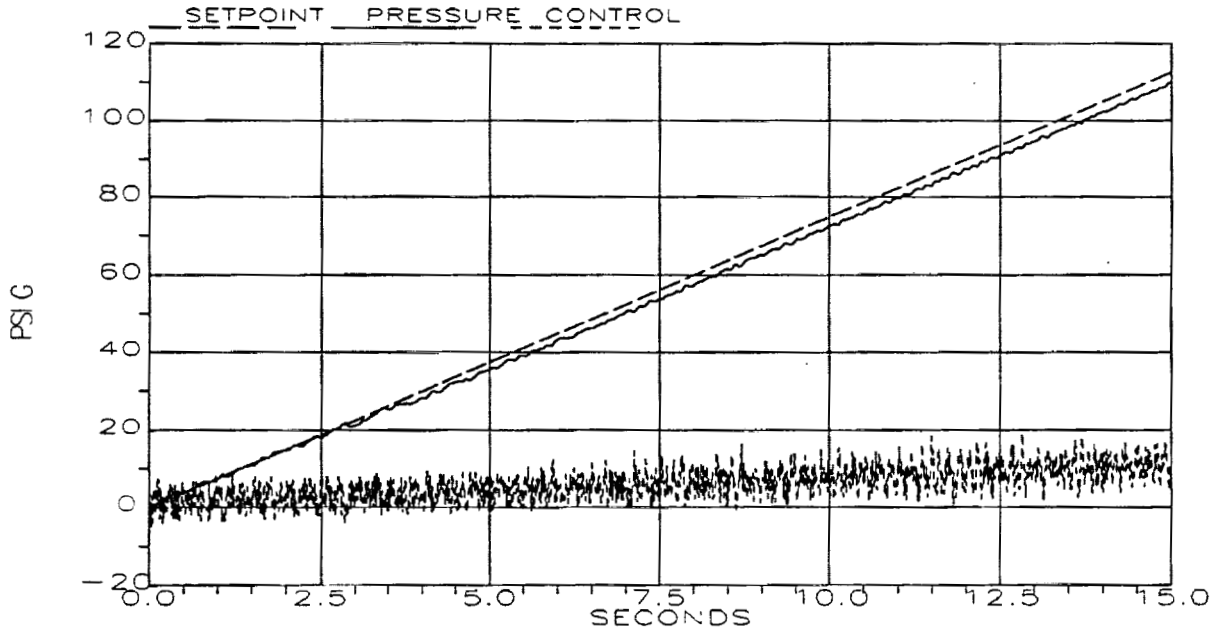


Figure 3.3 Case 5 Simulation Results

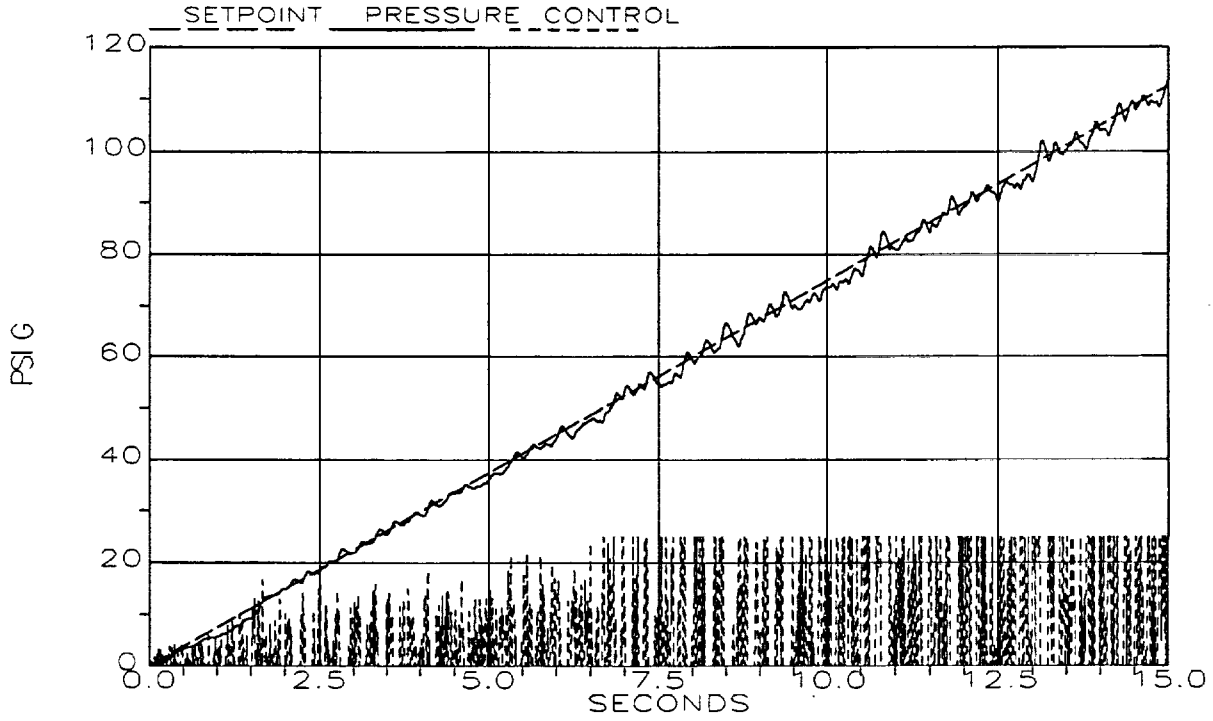


Figure 3.4 Case 6 Simulation Results

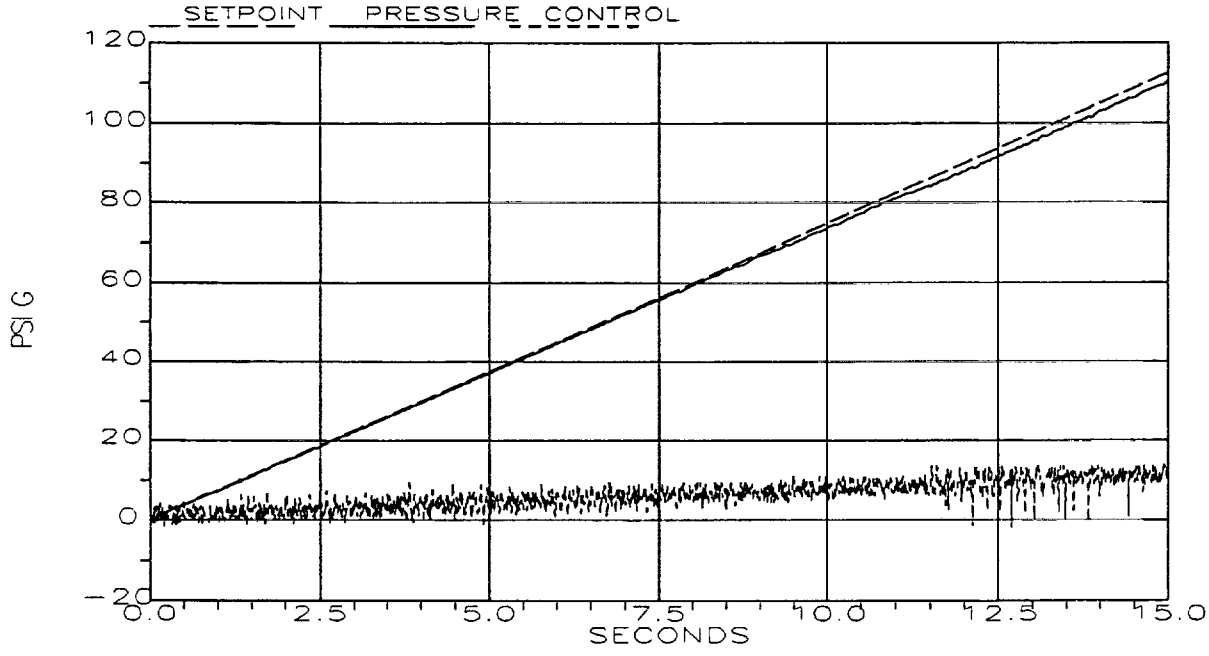


Figure 3.5 Case 7 Simulation Results

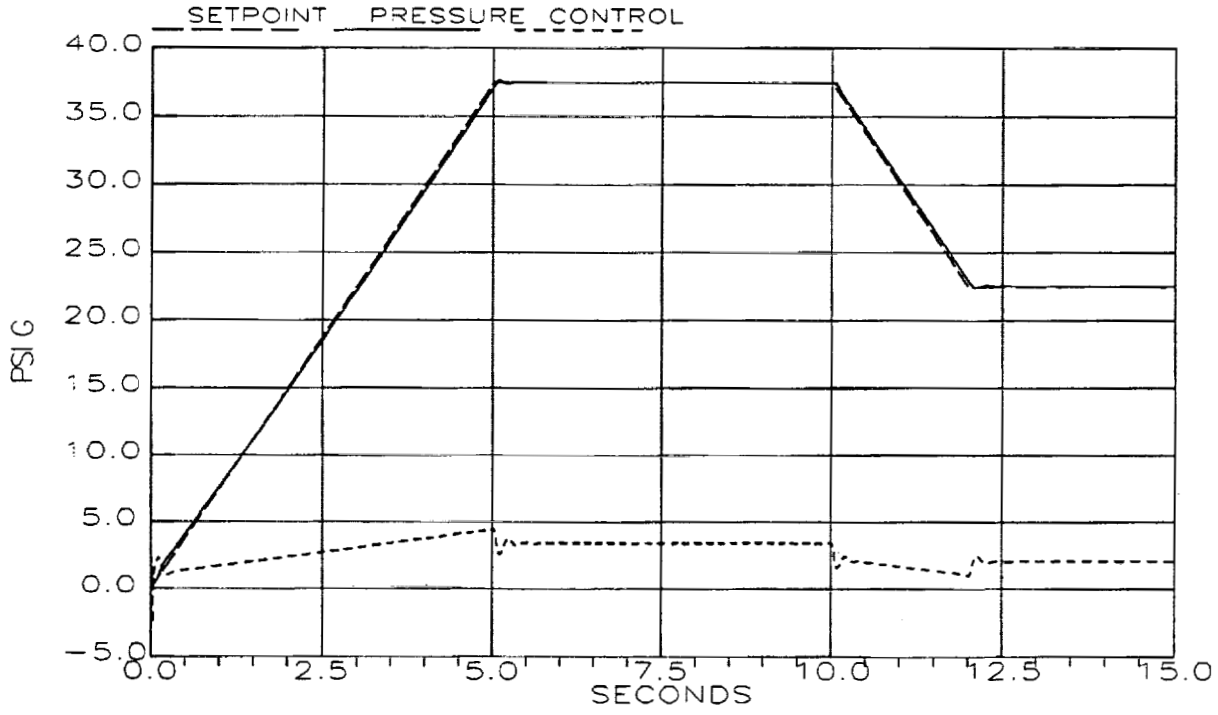


Figure 3.6 Case 8 and 10 Simulation Results

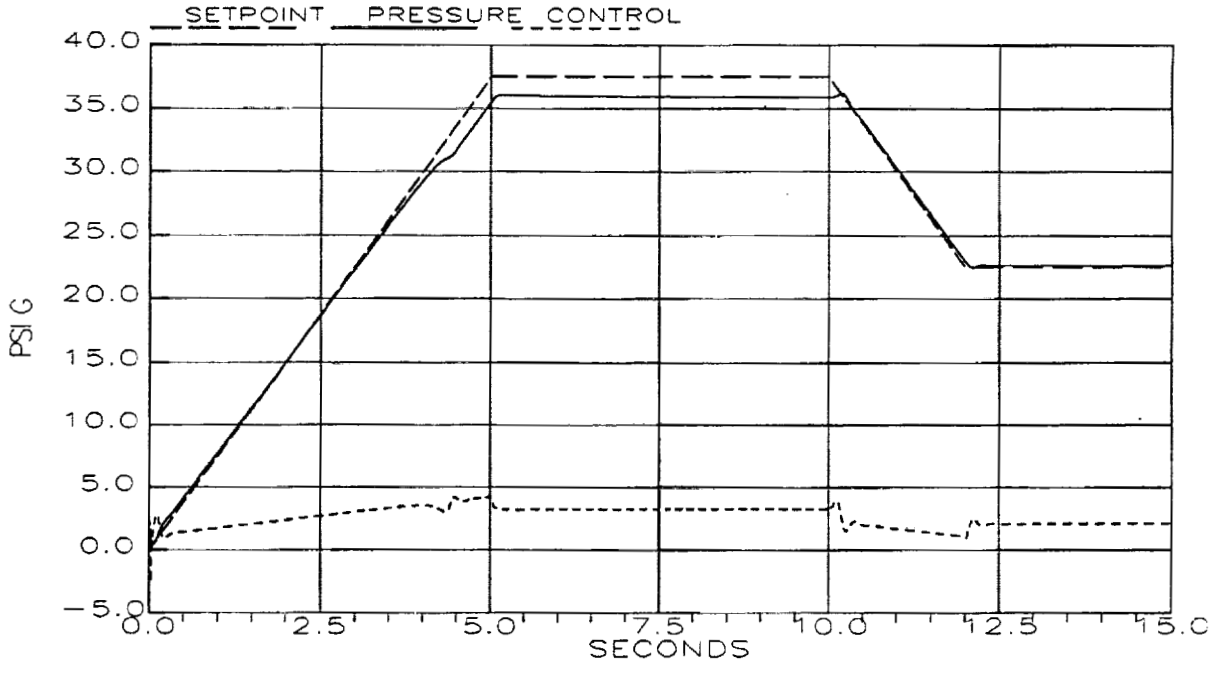


Figure 3.7 Case 9 Simulation Results

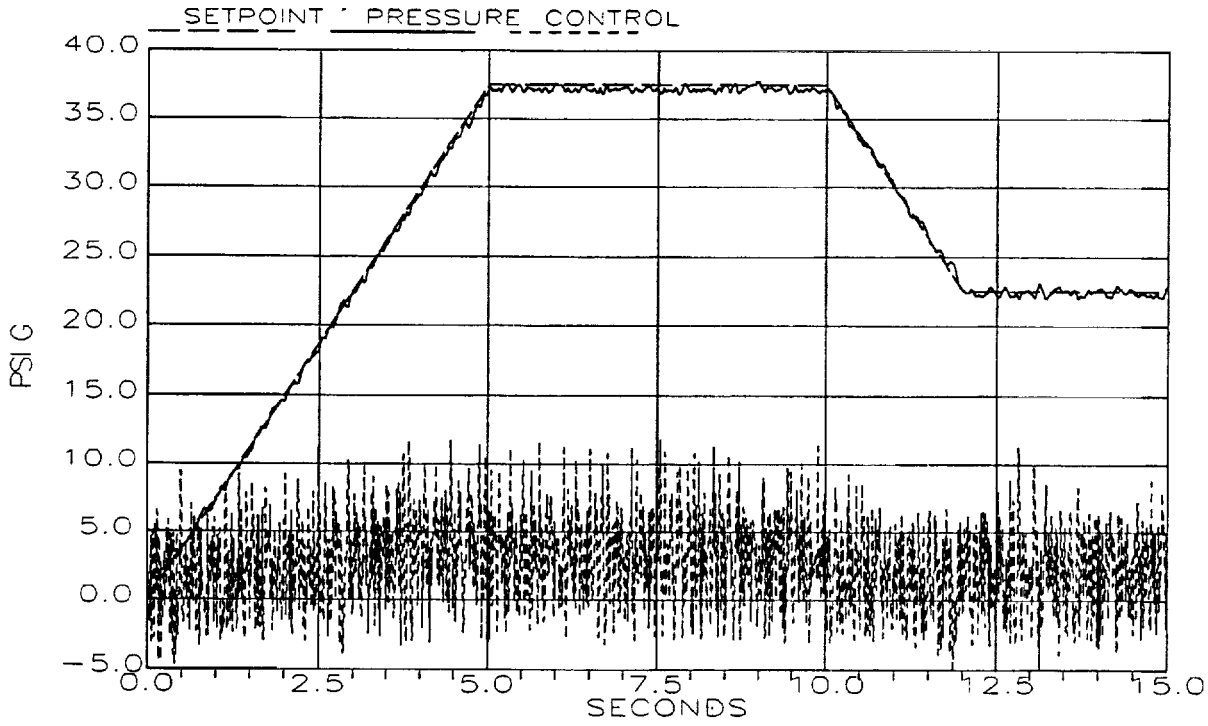


Figure 3.8 Case 11 Simulation Results

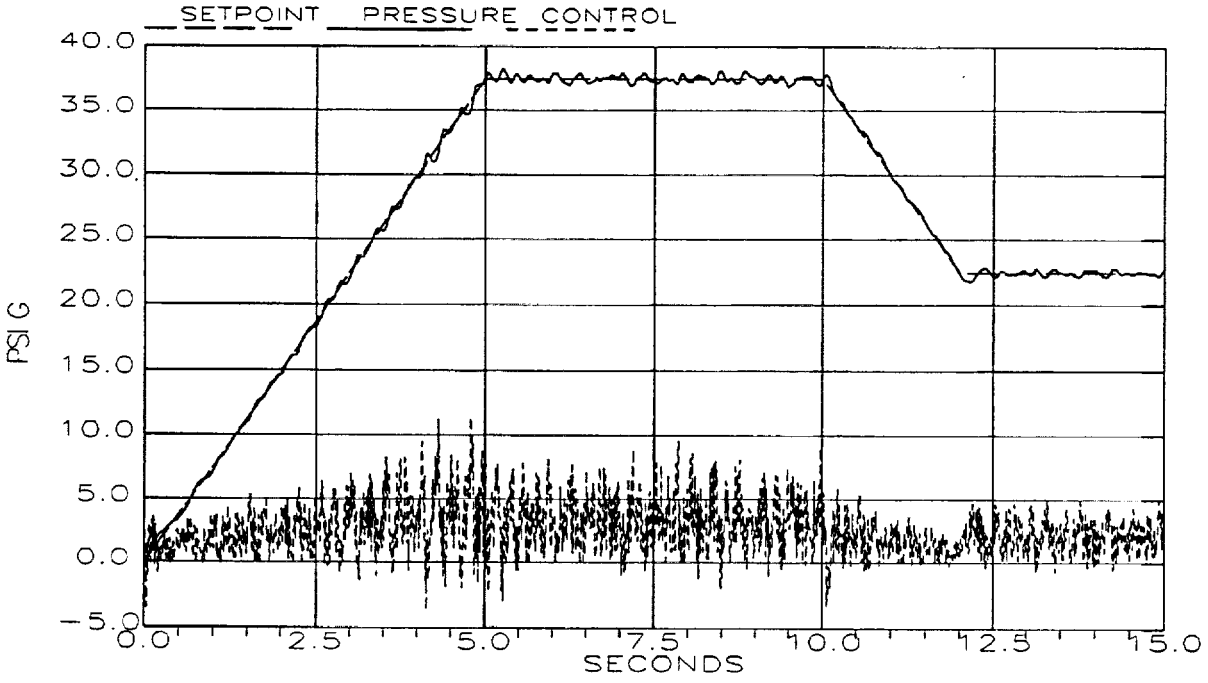


Figure 3.9 Case 12 and 14 Simulation Results

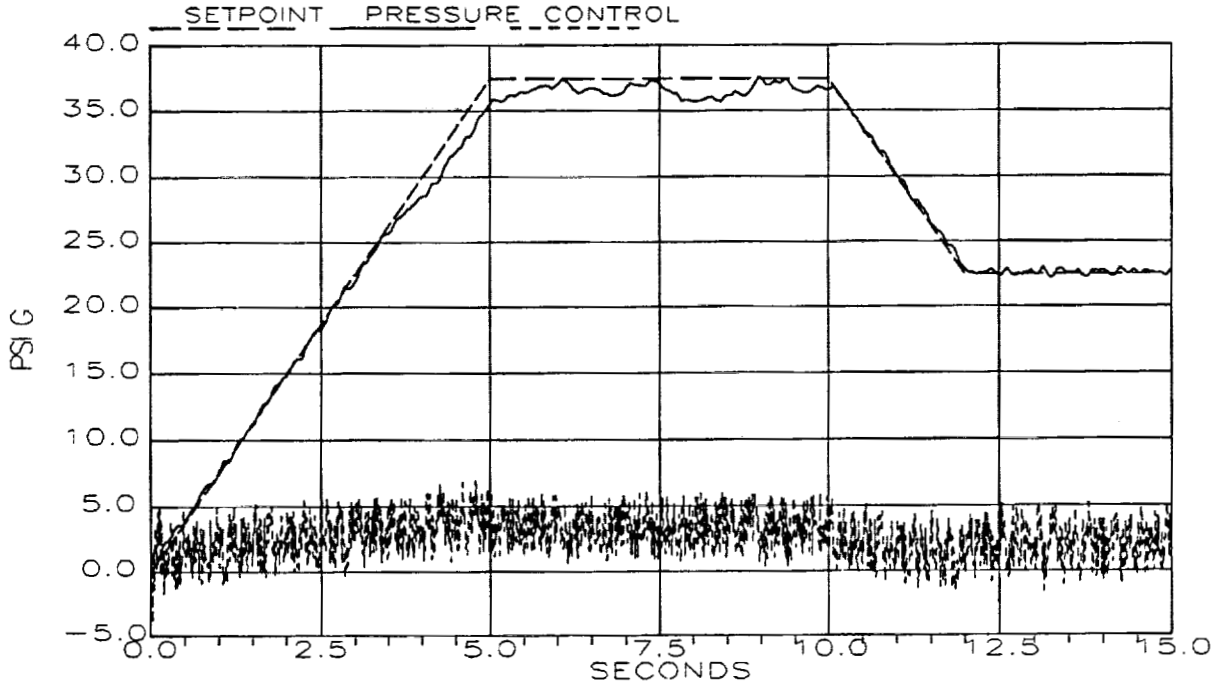


Figure 3.10 Case 13 Simulation Results

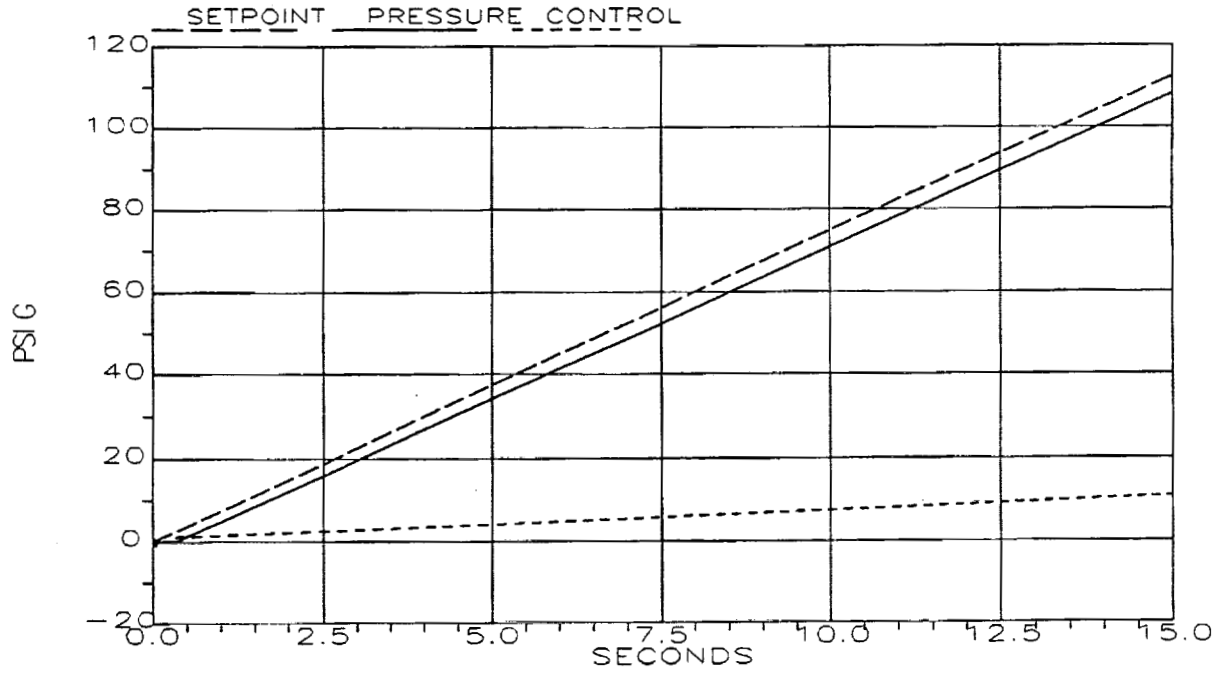


Figure 3.11 Case 15 Simulation Results

IV. DISCUSSION AND CONCLUSIONS

From Table 3.1 and the presented simulation results, the back-propagation trained and the HOMNA neural systems are proven to track varying command setpoints within the bounds of the training i/o histogram. Without incorporation of the integration scheme of [1], the HOMNA system still proved its tracking ability, though with varying levels of offsets. The proportional-integral-derivative (PID) system results exhibited no offsets due to the inherent integral scheme of the PID controller. From the simulation results, it is concluded that the integration scheme of [1] was simple to employ with equally satisfying results. Both back-prop trained, HOMNA, and PID systems proved their ability to accommodate for the varying levels of random noise injection.

In the use of NETS for the back propagation trained neural network, the ability to adjust the learning rate, momentum term, and the scaling factor (globally or locally) allowed for various configurations for starting conditions in the training. For the this project, the default global momentum was used, 0.09. A global learning rate of 0.5 through 1 was used for all cases. A scaling factor of 0.1 was used for all cases.

For the HOMNA trained system, larger i/o histogram sets were attempted with no significant difference in performance for select cases. With more effort or other techniques, it is believed that the difference could be corrected. In this project it was discovered that stripping the first few exemplar vectors from the i/o histogram (or the established training set) made a significant difference in the performance. For some cases, without stripping the first few inherent exemplar state vectors resulted in erroneous results ranging from wide dispersion (between setpoint and system state) to complete instability. The justification for stripping the first few exemplars stems from the scheme of [1]. That is, for the first few exemplars there is always inherent membership in the training set kernel. For select cases, the effects of stripping the exemplars before or after the Kernels algorithm software routine had no indicative difference.

For the neural controller of step 6 (i/o training set generated by the Kernels algorithm), Figure 3.11 illustrates that the controller can still track the command setpoint; however, the amount of offset, unseen in other backprop cases where the training set was the full i/o histogram (300 samples), is obviously due to the reduced size of the training set (40 samples). This was expected since the purpose of the Kernels algorithm is to select critical exemplars from a large data set. It is these critical exemplars that best represents the set (or population) as a whole. The choice of a back-prop trained or a HOMNA based neural controller to serve as a standalone or parallel backup to an existing PID controller is certainly realizable.

REFERENCES

1. Porter, W. A. and Liu, Wie, "Neural Controllers For Systems With Unknown Dynamics," The Laboratory for Advanced Computer Studies, The University of Alabama in Huntsville, March 1994.

2. Trevino, L. C., **MODELING, SIMULATION, AND APPLIED FUZZY LOGIC FOR INLET PRESSURE CONTROL FOR A SPACE SHUTTLE MAIN ENGINE AT TECHNOLOGY TEST BED**, Masters Thesis, The University of Alabama in Huntsville, June, 1993.

CLIPS TEMPLATE SYSTEM FOR PROGRAM UNDERSTANDING

Ronald B. Finkbine, PhD.
Department of Computer Science
Southeastern Oklahoma State University
Durant, OK 74701
finkbine@babbage.sosu.edu

ABSTRACT

Program Understanding is a subfield of software re-engineering and attempts to recognize the run-time behavior of source code. To this point, the success in this area has been limited to very small code segments. An expert system, HLAR (High-Level Algorithm Recognizer), has been written in CLIPS and recognizes three sorting algorithms, Selection Sort, Quicksort and Heapsort. This paper describes the HLAR system in general and, in depth, the CLIPS templates used for program representation and understanding.

INTRODUCTION

Software re-engineering is a field of Computer Science that has developed inconsistently since the beginning of computer programming. Certain aspects of what is now software re-engineering have been known by many names: maintenance, conversion, rehosting, reprogramming, code beautifying, restructuring, and rewriting. Regardless of the name, these efforts have been concerned with porting system functionality and/or increasing system conformity with programming standards in an efficient and timely manner. The practice of software re-engineering has been constrained by the supposition that algorithms written in outdated languages cannot be re-engineered into robust applications with or without automation.

It is believed by this author that existing software can be analyzed, data structures and algorithms recognized, and programs optimized at the source code level with expert systems technology using some form of intermediate representation. This intermediate form will provide a foundation for common tool development allowing intelligent recognition and manipulation. This paper describes a portion of the HLAR (High-Level Algorithm Recognition) system [1].

The Levels of *understanding* in the software re-engineering and Computer Science fields is displayed

in Figure 1 [2]. The lowest of these, *text*, is realized in the simple file operations; opening, reading, writing and closing. The next level is *token* understanding and occurs in compilers within their scanner subsystem. Understanding at the next level, *statement*, and higher generally does not occur in most compilers, which tend to break statements into portions and correctly translate each portion, and, therefore, the entire statement. One exception is the *semantic* level which some compilers perform when searching for syntactically correct but logically incorrect segments such as *while (3 < 4) do S*.

```
[7] Program
[6] Plan
[5] Semantic
[4] Compound Statement
[3] Statement
[2] Token
[1] Text
```

Figure 1: Levels of Understanding

Software re-engineering requires that higher-level understanding take place and the act of understanding should be performed on an intermediate form. The representation method used in this research project is the language ALICE, a very small language with a Lisp-like syntax. It is intended that programs in existing high-level languages be translated into ALICE for recognition and manipulation. This language has only five executable statements; *assign*, *if*, *loop*, *call* and *goto*. All syntactic sugar is removed, replaced with parentheses. Figure 2 displays a simple assignment statement. The *goto* statement is used for translating unstructured programs and the goal is to transform all unstructured programs into structured ones.

```
(assign x 0)
```

Figure 2: Assignment Statement

FACT REPRESENTATION

Prior to initiation of the HLAR system, programs in the ALICE intermediate form are translated into a set of CLIPS facts which activate (be input to and fulfill the conditions of) the low-level rules. Facts come in different types called templates (records) which are equivalent to frames and are ``asserted'' and placed into the CLIPS facts list. Slots (fields) hold values of specified types (integer, float, string, symbol) and have default values. As a rule fires and a fact, or group of facts, is recognized, a new fact containing the knowledge found will have its slots filled and asserted. Continuation of this process will recognize a larger group of facts and, hopefully, one of the common algorithms.

The *assign* statement from the previous Figure is translated into the equivalent list of facts in Figure 2. This is a much more complicated representation, and not a good one for programmers, but much more suitable for an

```
(general_node (number 1)
  (sibling 0)
  (address "2.10.5")
  (node_type assign_node))
(assign_node (number 1)
 (lhs_node 1)(rhs_node 1)
(general_node_parent 1))
(identifier_node (number 1)
  (operand 2)(name "x"))
(expression_node (number 1)
  (operand_1 2))
(integer_literal_node
  (number 1)
  (operand 2)(value 1))
```

Figure 3: Facts List

expert system.

The code of this Figure displays a number of different templates. The *general_node* is used to keep all statements in the proper order within a program and the *node_type* slot describes the type of statement associated with the node. The template *assign_node* with its appropriate *number* slot , its *lhs_slot* slot points to the number one *operand_node*, which is also pointed to by the operand slot in the number one *identifier_node*. The *rhs_node* slot of the assignment points to the number one *expression_node* and its *operand_1* slot points to the number two *operand_node* which is also pointed to by the operand slot in the number one *integer_literal_node*.

This is a more complex representation of a program than its ALICE form, but it is in a form that eases construct recognition. An ALICE program expressed in a series of CLIPS facts is a list and requires no recursion for parsing. Each type of node (i.e. *general, assign, identifier, integer_literal, operand, expression*) are numbered from one and referenced by that number. Once the ALICE program is converted into a facts list, low-level processing can begin.

TEMPLATE REPRESENTATION

Templates in CLIPS are similar to records or frames in other languages. In the HLAR system, templates are used in a number of different areas including general token, *plan* and knowledge representation. A properly formed template has the general form of the keyword *deftemplate* followed by the name of the template, an optional comment enclosed in double quotes and a number of *field* locations identifying attributes of the template. Each attribute must be of the simple types in the CLIPS system: *integer, string, real, boolean* or *symbol* (equivalent to a Pascal enumerated type) and default values of all attributes can be specified.

Currently there are three types of template recognition; *general* templates are used to represent the statement sequence that constitute the original program., *object* templates are used to represent the clauses, statements and algorithms that are recognized, and *control* templates are used to contain the recognition process.

GENERAL TEMPLATES

The names of all the general templates are listed in Figure 4. These are the templates that are required to represent a program in a list of facts having been translated from its original language which is similar to a compiler derivation tree.

```
general_node
argument_node
assign_node
call_node
define_routine_node
define_variable_node
define_structure_node
evaluation_node
expression_node
if_node
loop_node
parameter_node
program_node
identifier_node
integer_literal_node
struc_ref_node
struc_len_node
```

Figure 4: General Templates

The *general_node* referenced in this Figure is used to organize the order of the statements in the program. Prior to the HLAR system being initialized, a utility

program parses the ALICE program depth-first and generates the facts list needed for analysis. The *general_node* template is utilized to represent the order of the statements and contains no pointers to statement nodes. The statement-type nodes identified in the next section contain all pointers necessary to maintain program structure. Each token-type has its own templates for representation within an ALICE program. Included are node types for routine definitions, the various types of compound and simple statements, and the attributes of each of these statements. The various types of simple statement nodes contain pointers back to the *general_node* to keep track of statement order. These statement node templates contain the attributes necessary to syntactically and semantically reproduce the statements as specified in the original language program, prior to translation into ALICE.

In an effort to reduce recognition complexity, which is the intent of this research, specialty templates for each item of interest within a program have been created. An earlier version of this system had different templates for each form, instead of one template with a symbol-type field for specification. This refinement has reduced the number of templates required, thus reducing the amount of HLAR code and the programmer conceptual-difficulty level. The specialty templates are listed in Figure 5.

```
spec_call
spec_parameter
spec_exp
spec_assign
loop_algorithm
spec_eval
minimum_algorithm
swap_algorithm
if_algorithm
sort_algorithm
```

Figure 5: Special Templates

OBJECT TEMPLATES

Expressions are the lowest-common denominator of *program understanding*. They can occur in nearly all statements within an ALICE program. To reduce the complexity of *program understanding*, each expression for which we search is designated as a special expression with a specified symbol-type identifier. Expressions and structure references present a particular problem since they are mutually recursive as expressions can contain structure references and structure references can contain expressions. This problem came to light as the HLAR system became too large to run on the chosen architecture (out of memory) with the number of templates, rules and facts present at one time. Separation of the rules into several passes required that expressions and structure references be detected within the same rule group. Examples *a[i]* and *a[i+1] > a[i]* represent these concepts.

Figure 6 contains the specialty template for expressions. This template contains a number of fields for specific values, but most of interest is the field *type_exp*. Identifiers that represent specific expressions are listed as the allowed symbols. Complexity is reduced in this representation since multiple versions of the same statement can be represented by the same symbol. An example is two versions of a variable increment statement; $x = x + 1$ and $x = 1 + x$.

```
(deftemplate spec_exp
 (field type_exp (type SYMBOL)
  (default exp_none)
   (allowed-symbols exp_none
      exp_0 exp_1 exp_id exp_first
      exp_div_id_2 exp_plus_id_1
      exp_minus_id_1 exp_plus_id_2
      exp_minus_id_2 exp_mult_id_2
      exp_plus_div_id_2_1
      exp_minus_div_id_2_1
      exp_div_plus_id_id_2
      exp_ge_id_id exp_gt_const_id
      exp_gt_id_id exp_gt_id_const
      exp_gt_id_minus_id_1
      exp_lt_id_minus_id_1
      exp_gt_strucid_strucid
      exp_lt_strucid_strucid
      exp_lt_strucid_struc_plus_id_1
      exp_gt_strucid_id
      exp_ne_id_true
      exp_or_gt_id_min_id_1_ne_id_t
      exp_or_gt_id_id_ne_id_t
      exp_strucfirst))
(field id_1 (type INTEGER)
 (default 0))
(field id_2 (type INTEGER)
 (default 0))
(field id_3 (type INTEGER)
 (default 0))
(field exp_nr (type INTEGER)
 (default 0)))
```

Figure 6: Expression Template

Recognition of various forms of the *assign* statement occurs within the variable rules of HLAR. Common statements such as increments and decrements are recognized, as well as very specific statements such as $x = y / 2 + 1$.

Variable analysis is performed from one or more *assign* statements. The intent is to classify variables according to their usage, thus determining knowledge about the program derived from programmer intent and not directly indicated within the encoding of the program. An example would be saving the contents of one specific location of an array into a second variable such as *small = a[0]*.

Next come the multiple or compound statements such as the *if* or *loop* statements. These statements are the first of the two-phase rules to fire a *potential* rule and an *actual* rule. The *potential* rule checks for algorithm form as is defined against by the standard algorithm. The *actual* rule verifies that the proper statement containment and ordering is present. This allows for smaller rules, detection of somewhat *buggy* source code and elimination of *false positive* algorithm recognition.

The *loop* and *if* statements are the first compound statements that are handled within HLAR. Both have an *eval* clause tested for exiting the *loop* or for choosing the boolean path of the *if*. Both of these statements require a field to maintain the limits of the control of the statement. Generally, a *loop* or *if* statement will contain statements of importance within them and the concept of statement

containment will be required to be properly accounted for.

Higher-level algorithms, such as a *swap*, *minimization*, or *sort*, require that *subplans* be recognized first. This restriction is due to the size of the Clips rules. The more conditional elements in a rule, the more difficult and unwieldy for the programmer to develop and maintain.

Control templates are listed in Figure 7. These are used to control the recognition cycle within Clips. Control templates are necessary after preliminary recognition of a *plan* by a *potential* rule to allow for detection of any intervening and interfering statements prior to the firing of the corresponding *actual* rule.

```
not_fire
assert_nots_scalar
asserts_not_struc
```

Figure 7: Control Templates

Figure 8 is a hierarchical display of the facts from the previous Figure. It expresses the relationships among the nodes and shows utilization of the integer pointers used in this representation.

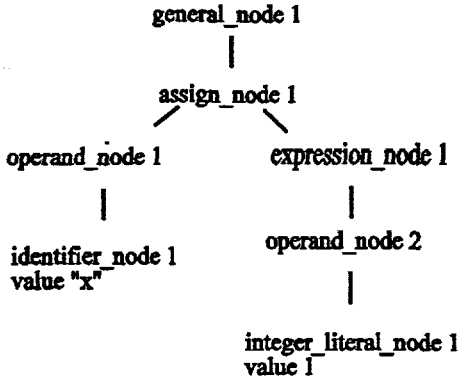


Figure 8: Derivation Tree

STATEMENT RECOGNITION

This section will describe the recognition process of a simple variable increment as displayed in Figure 9 as an example of the recognition process. The first rule firing will recognize a *special expression*, an identifier plus the integer constant one. The second rule firing will recognize a variable being assigned a *special expression* and a third rule will recognize that the identifier on the right hand side of the statement and the variable on the left hand side are the same, thus signifying an incrementing assignment statement.

```
(assign x (plus x 1))
```

Figure 9: Variable Increment

These three rule firings process one statement and further rule conditional elements will attempt to group this statement with related statements to recognize multi-statement constructs. An example would have this *assign* statement within a *loop* statement, and both preceded by an $x = 0$

initialization statement. This would indicate that the variable *x* is an index of the *loop* statement.

SUMMARY AND FUTURE RESEARCH

This research has lead to the development of a template hierarchy for *program understanding* and a general procedure for developing rules to recognize program *plans*. This method has been performed by the researcher, but will be automated in future versions of this system.

There are currently three algorithm *plans* recognized including the selection sort (SSA), quicksort (QSA) and heapsort (HSA). The SSA requires 50 rule firings, the QSA requires 90 firings, and the HSA, the longest of the algorithms at approximately 50 lines of code and taking over 60 seconds on a Intel 486-class machine requires 150 firings. The complete HLAR recognition system contains 31 templates, 4 functions, and 135 rules.

The research team intends to concentrate on algorithms common to the numerical community. The first version of the HLAR system has been successful at recognizing small algorithms (less than 50 lines of code). Expansion of HLAR into a robust tool requires: rehosting the system into a networking environment for distributing the recognition task among multiple CPUs, automating the generation of recognition rules for improved utility, attaching a database for consistent information and system-wide (multiple program) information, and a graphical user-interface.

AUTHOR INFORMATION

The author has: a B.S. in Computer Science, Wright State University, 1985; an M. S. in Computer Science, Wright State University, 1990; and a Ph.D. in Computer Science, New Mexico Institute of Mining and Technology, 1994. He is currently an Assistant Professor of Computer Science at Southeastern Oklahoma State University.

REFERENCES

1. 'High-Level Algorithm Recognition,' R. B. Finkbine, Ph.D. Dissertation, New Mexico Institute of Mining and Technology, 1994.

2. 'Pat: A Knowledge-based Program Analysis Tool,' M. T. Harandi and J. Q. Ning. In 1988 IEEE Conference of Software Maintenance, IEEE CSP, 1988.

AN IMPLEMENTATION OF FUZZY CLIPS AND ITS APPLICATIONS UNCERTAINTY REASONING IN MICROPROCESSOR SYSTEMS USING FUZZY CLIPS

Yuen & Lam

Abstract unavailable at time of publication.

amit

Session 3A: Data Analysis Applications

Session Chair: Jim Harrington

USING EXPERT SYSTEMS TO ANALYZE ATE DATA

Jim Harrington
Honeywell Military Avionics
MS 673-4
13350 US Highway 19 N.
Clearwater FL, 34624
harrington@space.honeywell.com

ABSTRACT

The proliferation of Automatic Test Equipment (ATE) is resulting in the generation of large amounts of component data. Some of this component data is not be accurate due to the presence of noise. Analyzing this data requires the use of new techniques. This paper describes the process of developing an expert system to analyze ATE data and provides an example rule in the CLIPS language for analyzing trip thresholds for high gain/high speed comparators.

INTRODUCTION

We are seeing a proliferation of "Simple" Automatic Test Equipment (ATE) based on personal computers. Large quantities of test data is being generated by these test stations. Some of this data will not accurately represent the true characteristics of the equipment being tested, particularly when the power supply in the personal computer is being used to power the ATE circuitry. This paper discusses a methodology for developing an expert system to examine the data files generated by the ATE. This expert system can be used to produce a data file containing the "most probable" data values with the effect of power supply noise removed. These "most probable" data values are based on specific statistical processes and special heuristics selected by the rule base.

THE NEED FOR A NEW APPROACH OF DATA ANALYSIS

Power supply noise can become a significant source of error during testing of high speed/high accuracy Analog to Digital (A/D) and Digital to Analog (D/A) converters (10-bits or greater) or high speed/high gain comparators. This power supply noise can cause:

- erratic data values from an A/D converter,
- wandering analog outputs (which can translate to harmonic distortion) from a D/A converter, and
- false triggers from a comparator.

A 10-bit A/D converter, optimized to measure voltages in the 0 to 5 V range, has a voltage resolution of 4.88 mV (i.e., the least significant bit--LSB--represents a voltage step of 4.88 mV). The 5 volt power bus on a personal computer (i.e., an IBM or third party PC) can have noise spikes in excess of 50 milivolts (mV). These noise spikes can, therefore, represent an error of greater than ten times the value of the LSB.

Even though the noise spikes on the PC power bus are considered high frequency (above a hundred Megahertz), high speed A/D converters and high speed comparators can capture enough energy from them to affect their operation.

The power supply noise is both conducted and radiated throughout the PC enclosure. Simple power line filters do not prevent noise from entering the ATE circuitry and affecting the tests

being performed. Much of the noise is cyclic in nature, such as that resulting from Dynamic Random Access Memory (DRAM) refresh and I/O polling. Other noise can be tied to specific events such as disk access.

The best method of improving the accuracy of the ATE is proper hardware design. Using power line filters does provide some noise isolation. Metal enclosures can also reduce susceptibility to radiated emissions. Noise canceling designs will also help. Self powered, free standing ATE which is optically coupled to the PC is probably the cleanest solution (from a power supply noise perspective). However, there are cases when an existing design must be used, or when cost and/or portability factors dictate that the ATE be housed within the PC. In these situations, an expert system data analysts can be used to enhance the accuracy of the test data.

The Traditional Software Approach

The traditional software approach to remove noise from a data sample is to take several readings and average the values. This approach may be acceptable if the system accuracy requirement is not overly stringent. A procedural language such as FORTRAN would be a good selection for implementing a system that just calculates averages. If A/D or D/A linearity is being tested, the averaging approach may not be accurate enough. The problem with this particular approach is that all of the data values that have been "obviously" affected by noise (as defined by "an expert") are pulling the average away from the true value.

The Expert System Approach

An expert system could be devised to cull out the "noisy" data values before the average is taken, resulting in a more accurate average. The problem is to develop an expert system that can be used to identify when noise might be effecting a data value.

THE PROCESS

The first step in developing an expert system to remove noisy data from the calculations is to define how the noise affects the circuits. For example, a voltage ramp is frequently used to test the linearity of an A/D converter and the trip points of a comparator. How does noise on that voltage ramp affect the performance of the circuitry being tested?

As a voltage source ramps down from 5 volts to Ø volts, noise can cause high speed A/D converters to output a data set with missing and/or repeated data values. Figure 1 shows the possible effect of random noise on the output of an A/D converter; the data line in this graph represents the source voltage that has been corrupted by noise and then quantized to represent the output of an A/D converter. Even though the general trend may be linear, the actual data is not.

Noise on a Ø to 5 volt ramp can cause a high speed comparator to change states too early or too late. Another common effect of noise on a comparator is a "bouncy" output (turning on and off in quick succession before settling to either the "on" or "off" state). Figure 2 shows the effect of noise on the output of the comparator.

The next step is to identify how the noise appears on the output of the equipment being tested. As seen in Figure 1, noise can cause runs of numbers with the same output from the A/D converter (e.g., three values in succession that the A/D converter interprets as 4002 mV). We also see missing data codes (as in the run of 3919, 3919, 3933, 3933--repeating the data values 3919 and 3933 but missing the data values 3924 and 3929 mV). Figure 2 shows an unexpected

"on" and "off" cycle from the comparator before the voltage ramp reached the true trip point of 3970 mV. An additional cycle to the off state occurs at the end of the grapg as seen in the comparator output going to 0 V, it should have stayed in the high (+5 V) state.

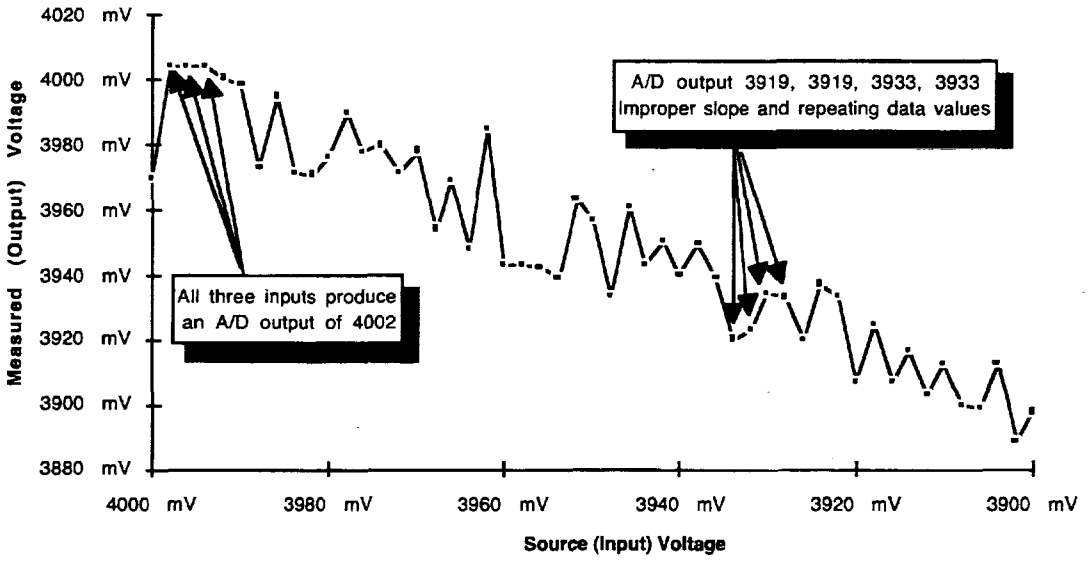


Figure 1. A/D Output Errors Caused by a Noisy Voltage Source

For brevity, the rest of this paper focuses on the comparator example; the approaches discussed for the comparator are directly applicable to the A/D converter as well.

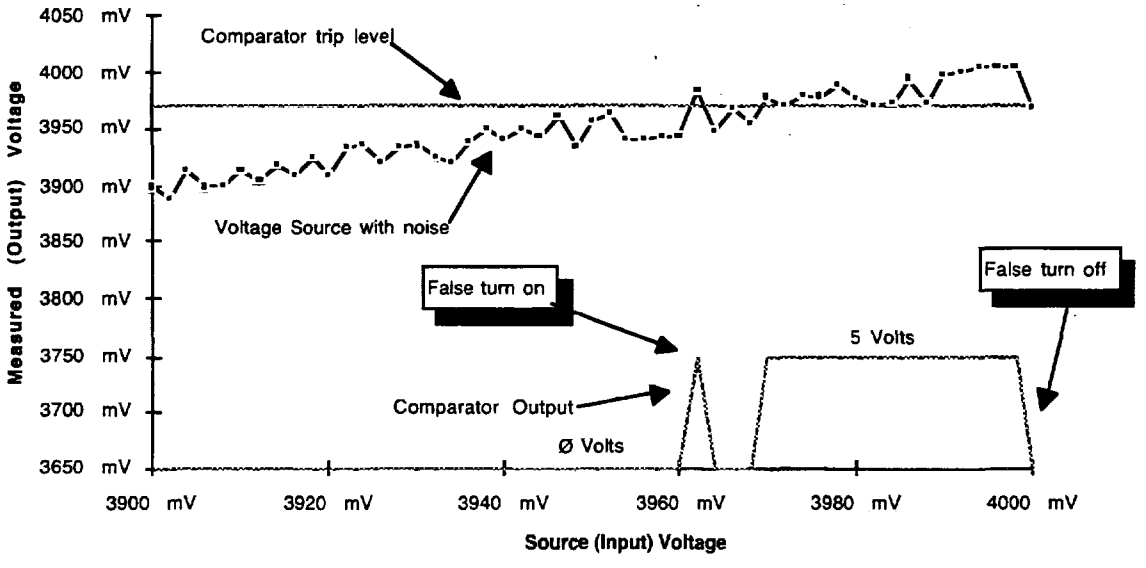


Figure 2. Comparator Output With a Noisy Source Signal

In most test set-ups, there will be no synchronization between any cyclic noise in the PC and the data acquisition process because most computer systems fill a buffer before dumping the data to the disk drive (this is ture unless the code generated for data acquisition goes through the specific process of "read data" "forced write to the disk" "read data" "forced write to the disk" etc.). This means that the cyclic noise will affect the data at random times during the data taking process. The non-cyclic random noise will, of course, appear at random times. Therefore, if we repeat the test multiple times, the noise should manifest itself at different places in the test sequence each time.

The noise shown in Figures 1 and 2 is random, with peaks as high a 50 mV. These plots were constructed to demonstrate the effect of noise on the circuitry. Under normal conditions, with the test voltage generated in a PC, the actual source voltage would look like the data shown in Figure 3. The "Source Voltage ramp" in Figure 3 shows noise spikes as high as 65 mV at regular intervals. The area between the noise spikes has some low—level asynchronous noise. The results of the combination of low level random noise and high level cyclic noise on a comparator is shown at the bottom of the graph (the voltage trip level is still set at 3970 mV).

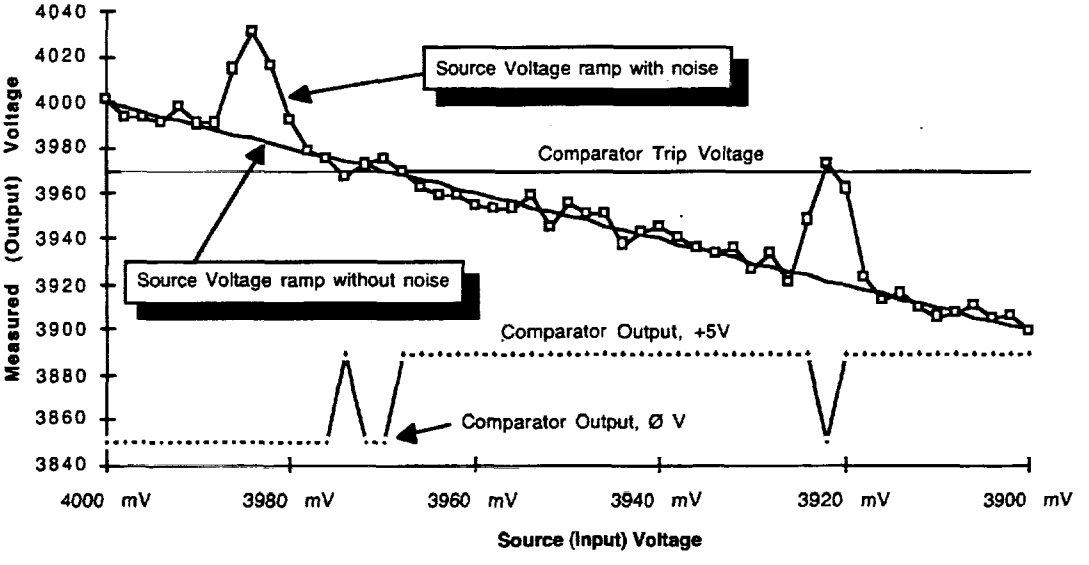


Figure 3. Noise Waveform During Test

Table I lists the voltages at which the comparator output shifts between Ø V and 5 V. The table also shows the voltages of the initial "bounce" on and off ("Off to On #1" and "On to Off #1"), and the inadvertent bounce off and back on ("On to Off #2" and "Off to On #3"). A basic set of rules starts to emerge when this noise spectrum is combined with knowledge of the effect of noise on the data taking process.

DEVELOPING THE RULE SET

Three sets of data were accumulated for each threshold analysis. These data sets were separated in time by only seconds as the ATE reset itself; this meant that the basic environmental factors were fairly consistent (an important factor when using any statistical process on data). The following rules are shown in order of precedence, from most important to least important.

Salience levels were used to ensure the most important rule would fire before a rule of lesser importance.

Once the optimum data value is selected by a rule, the data is written to an output file. The old data is then removed from the fact list to prevent a rule of lower precedence from firing and changing the data value again.

Table I. Comparator Input Threshold Voltage Test

Comparator ID	*Off to On # 1*	*On to Off # 1*	*Off to On # 2*	*On to Off # 2*	*Off to On # 3*
001	3974	3972	3968	3922	3920
•					
•					
•					

Rule 1

The first rule developed was one of common sense. If the comparator does not show any bounces in any of the three data sets, and the data values are identical, that value should be used.

Rule 2

The second rule is similar. If two of the three data sets contain no bounces and the comparator output changes from Ø V to 5 V at the same input ramp voltage, the data value found in those two data sets is used. This rule takes effect (is instanciated) regardless of the number of bounces in the third data set (i.e., the third data set could contain a single trip level or 3 trip levels--as in Figure 3--or 100 trip levels). In mathematical terms this is called selecting the mode of the data (i.e., the most commonly repeated value in a data set [1]).

The second rule handles the possible situation of the noise spike occurring just prior to the time the true source voltage exceeds the trip level. The noise spike would cause the comparator to trip early. The philosophy behind the rule is that the high level noise spikes seen in Figure 3 can cause the comparator to switch early in one data set, but the probability of the noise spike affecting two data sets the same way is extremely low.

This rule will also fire if none of the three data sets have bounces, and the values are the same. If both rule 1 and rule 2 fire, the trip level will be recorded in the output file twice. Repeated data in the output file can cause other data analysis programs to have problems. This is the reason the old data (the data used to infer the optimum trip level) is removed by each rule that fires; so that the rule with the highest precedence will remove the trip level data, preventing a secondary rule from also firing.

Rule 3

The next rule handles the case where no bounces are detected, but the trip levels in the three data sets are all different. In this situation, it is difficult to determine which data set contains good data and which might contains noisy data. We decided to average the two closest together

values. This averaging represents the effect of finding a trip level that is between two steps in the voltage ramp (e.g., the addition of low-level noise causing the comparator to trip at one level one time and another level the second time). If all three data values are equally dispersed, the average of all three values is taken. The mathematics used to implement this rule in effect takes the median of the data when all three data sets were used (an easy way of selecting the average of three equally dispersed numbers).

In hindsight, another method of approaching the same rule is to track the interval of the 65 mV noise spike. By tracking the last occurrence of comparator output bounces, the period of the large cyclic noise spike can be determined (this approach requires an additional pass through the data to determine the period of the cyclic noise). The period of the cyclic noise can be used as an additional constraint, and the data set(s) that are closest to the middle of their periods can be averaged.

At this point, we have covered all of the easy rules. The following rules rely more on heuristics developed by the project research team (i.e., the domain experts).

Rule 4

The fourth rule covers the case when two of the data sets contain no bounces, but the data values are different, and the third file contains bouncy data. In this situation, we decided to average the data from the two non-bouncy data sets. An additional precaution was added to this rule in that if either of the two values is more than ±5% from the average value, the operator is notified of the comparator being tested and the two trip voltage levels.

Rule 5

If only one of the data sets doesn't contain bouncy data, the data value of that one data set is used. This rule covers the case when "the experts" feel that two of the data sets are affected by some high level noise spikes. It is "felt" that the one file non-bouncy data set contains the most accurate trip level on the most frequent basis.

Rule 6

The final case is when all three data sets contain bounces. This means that all three data sets have been affected by noise. In Figure 3, the noise spikes are only one reading increment wide. Both the on-off cycle and the off-on cycle are in the inverted state for one data reading. The difference between the first two data values in Table I is 2 mV; this is also true for the last two data values. The difference between the second and third data values is 4 mV, and the difference between the third and fourth data values is 46 mV. The comparator settles to the "On" state at 3968 mV. The rule that forms out of this analysis is to select for the trip level (when the comparator turns "On") the voltage that has the greatest difference between it and the adjacent "Off" voltage (e.g., in Figure 3, the trip level would be selected as 3968 mV, the point which starts the 46 mV span before the next "Off" spike).

The average of all 5 data values is 3951.20 mV; an error of 16.8 mV compared to the 3968 mV trip level. The average of the 3 values which cause an "Off to On" transition of the comparator is 3954 mV; an error of 14 mV. The increase in accuracy is easily seen in this example. But, we are not done yet. What if all of the transitions of the comparator occur at equal voltage intervals?

There is no clear cut heuristic when all of the "Off to On" and "On to Off" data values are evenly spaced. The system we developed defaulted to the first "Off to On" transition. Depending on the system implementation, the middle "Off to On" transition may be the best selection.

The coding of this rule is a little tricky. The code in Figure 4 shows how this rule could be implemented in CLIPS.

```
(defrule Compromise "This rule finds the most probable data value out of a set of bounces"
  (Evaluate_Data)      ;flag fact that allows the compromise rule to function
  ?Orig_Data <- (?set&set_1|set_2|set3 $?data)  ;get the data
  (test (>= (length $?data) 2))       ;test for bounces
=>
  (retract ?Orig_Data) ;Get rid of the data set that is to be replaced
  (bind ?elements (length $?data))
  (if (evenp ?elements)
    then
      (assert (?set 0)) ;Bit ended in original state--no data
    else
      (bind ?delta 0) ;Set up the variable to find the largest difference
      (bind ?y 2) ;Pointer to the first even number (position) data value
      (while (<= ?y ?elements)
      (bind ?x (- ?y 1)) ;Pointer to the position preceding ?y
      (bind ?z (+ ?y 1)) ;Pointer to the position following ?y
      (bind ?test_val (abs (- (nth ?x $?data) (nth ?y $?data))))
      (if (> ?test_val ?delta)
          then
            (bind ?pointer ?x)
            (bind ?delta ?test_val)
        )
        (bind ?test_val (abs (- (nth ?y $?data) (nth ?z $?data))))
        (if (> ?test_val ?delta)
          then
            (bind ?pointer ?z)
            (bind ?delta ?test_val)
      )
      (bind ?y (+ ?y 2)) ;Increment ?y to the next even position
    )
    (assert (?set =(nth ?pointer $?data))) ;put the new data on the fact list
  )
)
```

Figure 4. CLIPS Code Implementing Rule Number 6

The actual searching of the data set is performed in the "while" statement found on the right hand side of the rule (the "then" part of the rule, found after the "=>" symbols).

Rule 6 is be performed on all three data sets separately. Remember that this rule is invoked only when bounces are seen in all three data sets. This point helps to alleviate some of the worry surrounding the case where all of the trip levels are equally dispersed. Once all three data sets are evaluated and a single voltage level is selected, the three new values are put on the fact list to be operated on by the first 5 rules. Only rules number 1 through 3 actually apply since none of the data generated by rule 6 would contain bounces.

CONCLUSIONS

This process demonstrates a new approach to analyzing data taken by ATE. Increases in accuracy of 14 to 17 mV is demonstrated in rule 6. The other rules select the heuristic or statistical procedure used to determine the best data value from the multiple data sets provided. This paper illustrates the method of developing a set of rules, implementable in CLIPS, for defining the presence of noise in a data set and for removing the noisy data from the calculations.

ACKNOWLEDGMENTS

I would like to thank Richard Wiker and Roy Walker for their diligent work in helping to define the heuristics involved in the data analysis.

REFERENCES

1. Spiegel, Murray R., "Chapter 3: The Mean, Median, Mode and Other Measures of Central Tendency," SCHAUM'S OUTLINE SERIES: THEORY AND PROBLEMS OF STATISTICS, McGraw-Hill Book Company, New York, 1961, pp. 47 - 48.

REAL-TIME REMOTE SCIENTIFIC MODEL VALIDATION

Richard Frainier and Nicolas Groleau
Recom Technologies, Inc.

NASA Ames Research Center, m/s 269-2
Moffett Field, CA 94035-1000, USA

ABSTRACT

This paper describes flight results from the use of a CLIPS-based validation facility to compare analyzed data from a space life sciences (SLS) experiment to an investigator's pre-flight model. The comparison, performed in real-time, either confirms or refutes the model and its predictions. This result then becomes the basis for continuing or modifying the investigator's experiment protocol. Typically, neither the astronaut crew in Spacelab nor the ground-based investigator team are able to react to their experiment data in real time. This facility, part of a larger science advisor system called Principal-Investigator-in-a-Box, was flown on the Space Shuttle in October, 1993. The software system aided the conduct of a human vestibular physiology experiment and was able to outperform humans in the tasks of data integrity assurance, data analysis, and scientific model validation. Of twelve pre-flight hypotheses associated with investigator's model, seven were confirmed and five were rejected or compromised.

INTRODUCTION

This paper examines results from using a CLIPS-based scientific model validation facility to confirm, or refute, a set of hypotheses associated with a Shuttle-based life-science experiment. The model validation facility was part of a larger software system called "PI-in-a-Box" (Frainier et al., 1993) that was used by astronauts during the October, 1993 Spacelab Life Sciences 2 (SLS-2) mission. The model validation facility (called Interesting Data Filter in the PI-in-a-Box system) compares the output of the scientific data analysis routines with the investigator's pre-flight expectations in real-time.

The model validation facility compares analyzed data from the experiment with the investigator's model to determine its fit with pre-flight hypotheses and predictions. The fit can be either statistical or heuristic. Deviations are reported as "interesting". These deviations are defined as "needing confirmation", even if not part of the original fixed protocol. If confirmed, then at least a portion of the theoretic model requires revision. Further experiments are needed to pinpoint the deviation. This idea is at the heart of the iterative process of "theory suggesting experiment suggesting theory".

THE ROTATING DOME EXPERIMENT

The PI-in-a-Box software system was associated with a flight investigation called the Rotating Dome Experiment (RDE). This was an investigation into the effects of human adaptation to the micro-gravity condition that exists in Earth-orbiting spacecraft. A sensation, called "angular vection", was induced in a set of human subjects by having them view a rotating field of small, brightly-colored dots. After a few seconds, the subject perceives that s/he is rotating instead of the constellation of dots. This perception of self-rotation generally persists throughout the time that the dots are rotating, though occasionally the subject realizes that it is in fact the dots that are

rotating. This sudden cessation of the sensation of vection is termed a "dropout". With the RDE, the field of dots rotates in a set direction (clockwise/counter-clockwise) with a set speed for 20 seconds. There is a 10-second pause, and then the rotation resumes (though with a new direction and/or angular speed). There are six such 20-second trials for each experiment run.

There are three experiment conditions for RDE subjects. In the first, called "free-float", the subject grips a biteboard with his/her teeth in front of the rotating dome (and is otherwise floating freely). In the second, called "tether", the subject is loosely tethered to the front of the rotating dome without the biteboard. In the third, called "bungee", the subject is attached to the "floor" of the laboratory by a set of bungee cords, again teeth gripping a biteboard.

There are eight main parameters measured with respect to angular vection during the RDE. Four of these parameters are "subjective", meaning that the subject consciously reports them by manipulating a joystick/potentiometer: These are the time interval from the start of dome rotation to the onset of the sensation of vection (measured in seconds), the average rate of perceived vection (expressed as a percent of the maximum), the maximum rate of perceived vection (also expressed as a percent of the maximum), and the number of times during a 20-second trial that the sensation suddenly ceases (the dropout count, an integer). The remaining four parameters are "objective", meaning that the subject's involuntary movements are recorded. These are the first and second head movements associated with the torque strain gage mounted on the biteboard and the same head movements associated with subject neck muscle activity detectors (electromyograms). These eight parameters were measured for each 20-second trial of a run.

In the flight system, twelve distinct hypotheses were identified. These were all associated with the joystick-generated subjective parameters. They are:

1. There should be some sensation of angular vection.
2. The average time for the onset of the sensation of vection for the six trials of a run should be greater than 2 seconds* .
3. The average time for the onset of the sensation of vection for the six trials of a run should be less than 10 seconds.
4. Early in a mission, before adaptation to micro-gravity is significantly underway, the average of the six trials' maximum sensation of vection should be less than 90%.
5. Late in a mission, after adaptation to micro-gravity is complete, the average of the six trials' maximum sensation of vection should be more than 80%.
6. Tactile cues decrease the sensation of vection, therefore, the average of the six trials' maximum sensation of vection for a free-float run should be more than that of a bungee run.
7. The average of the six trials' average sensation of vection should be more than 30%.
8. The average of the six trials' average sensation of vection should be less than 80%.
9. Tactile cues decrease the sensation of vection, therefore, the average of the six trials' average sensation of vection for a free-float run should be more than that of a bungee run.
10. There should be at least one dropout during a bungee run.
11. There should not be an average of more than two dropouts per trial during a free-float run.
12. The average of the six trials' dropout count for a free-float run should be less than that of a bungee run.

MODEL VALIDATION

Real-time, quick-look data acquisition and analysis routines extract significant parameters from the experiment that are used by the model validation facility (see Figure 1). With the RDE,

*Strictly speaking, the numeric value of this hypothesis (and most other hypotheses) was subject to adjustment on a subject-by-subject basis as a result of pre-flight "baseline data" measurements. This was due to the significant variability between individual human subjects.

predictions were formed on the basis of data from two sources. The first source was previously-collected flight data. (The RDE was flown on three earlier missions: SL-1, D-1, and SLS-1.) The second source was from SLS-2 crew responses recorded on earth before the flight during baseline data collection sessions. These predictions were used to define thresholds that, if violated, indicated significant deviations from the investigator's model. Many space life-sciences investigations (including the RDE) are exploratory in nature, and the investigator team expected significant deviations for perhaps 20% of the experiment runs. When detected, these deviations were made available for display to the astronaut-operator. It is then that the reactive scientist briefly reflects on the situation and try to exploit the information to increase the overall science return of the experiment. This would most likely result in a change to the experiment protocol.

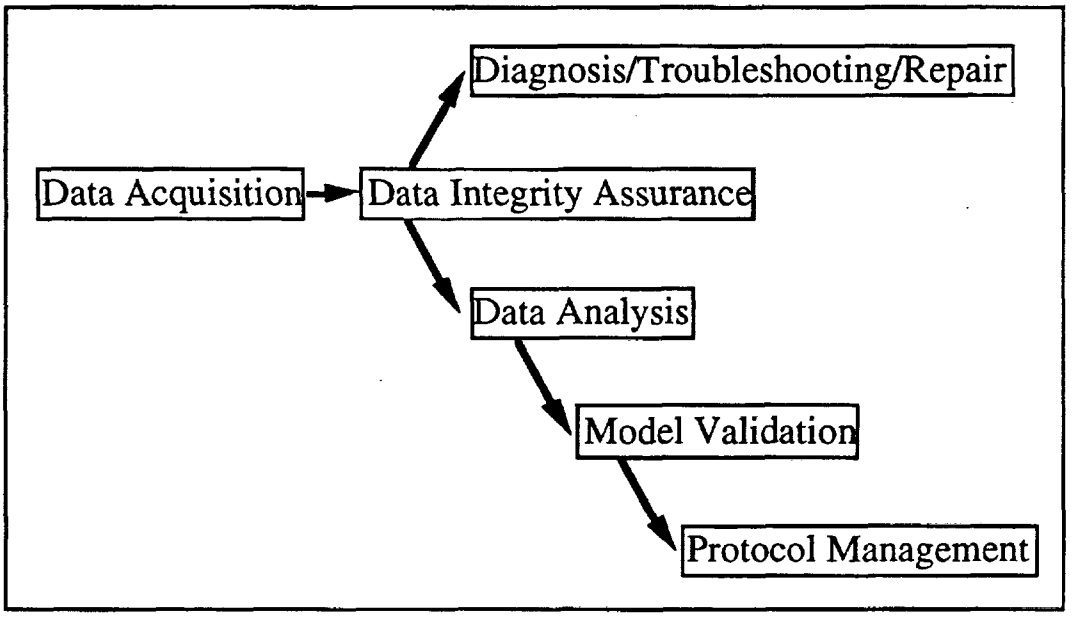


Figure 1: Flow of control.

For the PI-in-a-Box system, the model validation facility was named the Interesting Data Filter (IDF). The IDF was a set of CLIPS rules and facts that compared current experiment results with the investigator's preflight expectations. There were approximately two dozen rules† and 40 facts that comprised the pre-flight hypotheses.

FLIGHT RESULTS

Results of the flight use with respect to the 12 hypotheses are listed in Table I:

- the first column identifies the hypothesis number from the list of hypotheses presented earlier.
- the second column is the binomial probability of observing the given outcome assuming 95% of the run results agree with the model.
- the third column is our conclusion with respect to the hypothesis given the overall SLS-2 flight evidence. The hypothesis is rejected when the probability of observing the flight results given the hypothesis is < 0.001; it is compromised when the probability of observing the flight results given the hypothesis is < 0.01; it is suspect‡ when the probability of observing the flight results given the hypothesis is < 0.05; and it is accepted otherwise.

†See Appendix for listing of CLIPS rules.

‡This case does not occur for this data set.

- the fourth column summarizes the mission results. This is expressed as a ratio where the denominator represents the number of experiment runs producing data that bears on the hypothesis and the numerator represents the subset of those experiment runs whose data supports the hypothesis. The entry "n/a" denotes that the hypothesis was not applicable to that flight day.
- the last three columns present a more detailed view of the results from each of the three flight days (fd) when the system was in use.

These results indicate that seven of 12 pre-flight hypotheses were accepted. Five hypotheses were either rejected or compromised, indicating a need to modify the existing model with respect to the pattern of human adaptation to weightlessness over time, with respect to the importance of dropouts as an indication of adaptation, and with respect to the influence of tactile cues.

Table I: Flight data confirmation of vection-related hypotheses.

hyp #	probability	result	mission total	FD2	FD8	FD11
1	1.0000	accepted	38/38	12/12	14/14	12/12
2	0.2642	accepted	18/20	6/6	8/8	4/6
3	1.0000	accepted	38/38	12/12	14/14	12/12
4	1.0000	accepted	12/12	12/12	n/a	n/a
5	0.0000	rejected	0/12	n/a	n/a	0/12
6	0.0040	compromised	17/22	3/4	9/10	5/8
7	0.8576	accepted	37/38	11/12	14/14	12/12
8	1.0000	accepted	38/38	12/12	14/14	12/12
9	0.0040	compromised	17/22	4/4	8/10	5/8
10	0.0000	rejected	5/13	1/4	3/5	1/4
11	1.0000	accepted	13/13	4/4	5/5	4/4
12	0.0009	rejected	14/22	4/4	5/10	5/8

CONCLUSION

A scientific model validation facility has been devised for space science advisor systems that appears to be a useful framework for confirming or refuting pre-flight hypotheses. This facility is a key step to achieving truly reactive space-based laboratory science.

ACKNOWLEDGMENTS

The authors would like to thank other members of the PI-in-a-Box team, past and present, for their efforts, especially Lyman Hazelton, Silvano Colombano, Michael Compton, Irv Statler, Peter Szolovits, Larry Young, Jeff Shapiro, Guido Haymann-Haber, and Chih-Chao Lam. Thanks as well to NASA, and especially Peter Friedland, for management support. This work was funded by the NASA Office of Advanced Concepts and Technology AI program.

REFERENCES

Frainier Richard, Groleau Nicolas, Hazelton Lyman, Colombano Silvano, Compton Michael, Statler Irv, Szolovits Peter, and Young Larry, "PI-in-a-Box: a knowledge-based system for space science experimentation", AI Magazine, Menlo Park, CA, vol. 15, no. 1, Spring, 1994, pp. 39-51.

APPENDIX: MODEL VALIDATION FLIGHT RULE-SET

```
;; ;these rules follow CLIPS v4.3 syntax

;;start up IDF
(defrule idf-startup
   (idf)
   (disk-drive ?disk)
   (interface-directory ?dir)
   =>
   (bind ?predictions-file (str_cat ?disk ?dir "BDC-predictions"))
   (bind ?input-file (str_cat ?disk ?dir "idf-input"))
   (load-facts ?predictions-file)
   (load-facts ?input-file)
   (assert (idf-result not-interesting)) ; the default result
   (open (str-cat ?disk ?dir "idf-stats") idf-stats "w")) ; no append w/o file size limit checking!


;; formula: SD_square = (1/n)(sum_of squares - (square of sum)/n)
(defrule compute_statistics_for_six_trials
   (declare (salience 10))
   (?parameter trial_data ?t1 ?t2 ?t3 ?t4 ?t5 ?t6)
   =>
   (bind ?sum (+ (+ (+ (+ (+ ?t1 ?t2) ?t3) ?t4) ?t5) ?t6))
   (bind ?sum_of_squares (+ (** ?t1 2) (** ?t2 2) (** ?t3 2) (** ?t4 2) (** ?t5 2) (** ?t6 2)))
   (bind ?mean (/ ?sum 6))
   (bind ?SD_square (/ (- ?sum_of_squares (/ (** ?sum 2) 6)) 6))
   (bind ?SD (sqrt ?SD_square))
   (assert (?parameter sum ?sum))
   (assert (?parameter sum of squares ?sum_of_squares))
   (assert (?parameter experiment_result ?mean))
   (assert (?parameter standard deviation ?SD)))


;;; Parameter-specific rules to detect interestingness


;; ONSET_OF_VECTION

;; Onset of vection is interesting if it's non-existent (that is, less than 0.03 seconds)
(defrule no-vection-detected
   (declare (salience 5))
   (Onset_Of_Vection experiment_result ?x&:(< ?x 0.03))
   =>
   (assert (no-vection-detected)))

(defrule no-vection-detected--interesting
   (declare (salience 5))
   (no-vection-detected)
   (Maximum_Vection_Intensity experiment_result ?x&:(< ?x 10))
   =>
   (assert (Onset_Of_Vection conclusion potentially_interesting "No vection was detected.")))

;; Onset of vection is interesting if it's consistently < threshold (but >= 0.03)
(defrule onset_of_vection_less_than_2
   (Onset_Of_Vection experiment_result ?mean_found)
   (subject ?subj)
   (BDC-datum ?subj quick_onset ?threshold)
   (test (and (> ?mean_found 0.03) (< ?mean_found ?threshold)))
   =>
   (bind ?msg (str_cat "Mean onset of vection is less than " ?threshold " seconds"))
   (assert (Onset_Of_Vection conclusion potentially_interesting ?msg)))
```

```
;; Onset of vection in flight is interesting if it's consistently > threshold
(defrule onset_of_vection_greater_than_10
   (environment flight)
   (subject ?subj)
   (BDC-datum ?subj slow_onset ?threshold)
   (Onset_Of_Vection experiment_result ?mean_found&:(> ?mean_found ?threshold))
   =>
   (bind ?msg (str_cat "Mean onset of vection is greater than " ?threshold " seconds"))
   (assert (Onset_Of_Vection conclusion potentially_interesting ?msg)))


;; MAXIMUM_VECTION_INTENSITY

(defrule early_interesting_maximum_vection
; Early in flight (Day 0 / Day 1), maximum vection is interesting if it's consistently > threshold.
   (environment flight)
   (day 0|1)
   (subject ?subj)
   (BDC-datum ?subj early_hi_max ?threshold)
   (Maximum_Vection_Intensity experiment_result ?mean_found&:(> ?mean_found ?threshold))
   (not (no-vection-detected))
   =>
   (bind ?msg (str_cat "Max vection intensity mean is greater than " ?threshold "%"))
   (assert (Maximum_Vection_Intensity conclusion potentially_interesting ?msg)))


(defrule late_interesting_maximum_vection
; Late in the flight, maximum vection is interesting if it's consistently < threshold
   (environment flight)
   (day ?day&:(> ?day 7))              ; "Late" is Day 8 or later
   (subject ?subj)
   (BDC-datum ?subj late_lo_max ?threshold)
   (Maximum_Vection_Intensity experiment_result ?mean_found&:(< ?mean_found ?threshold))
   (not (no-vection-detected))
   =>
   (bind ?msg (str_cat "Max vection intensity mean is less than " ?threshold "%"))
   (assert (Maximum_Vection_Intensity conclusion potentially_interesting ?msg)))

;; Max vection is interesting if tactile > free-float
(defrule free-below-bungee--interesting--maximum-vection
   (body_position free-flt)
   (Maximum_Vection_Intensity experiment_result ?ff)
   (Maximum_Vection_Intensity running_mean ?val&:(> ?val ?ff)) ; bungee mean
   =>
   (assert (Maximum_Vection_Intensity conclusion potentially_interesting
         "Subj's max. vection < bungee cond. max. vection")))

(defrule bungee-above-free--interesting--maximum-vection
   (body_position bungee)
   (Maximum_Vection_Intensity experiment_result ?b)
   (Maximum_Vection_Intensity running_mean ?val&:(> ?b ?val)) ; free-float mean
   =>
   (assert (Maximum_Vection_Intensity conclusion potentially_interesting
         "Subj's max. vection > free-float cond. max. vection")))


;; AVERAGE_VECTION_INTENSITY

(defrule low_average_vection_intensity
; Average vection intensity is interesting if it's consistently < threshold
   (environment flight)
   (subject ?subj)
   (BDC-datum ?subj lo_average ?threshold)
   (Average_Vection_Intensity experiment_result ?mean_found&:(< ?mean_found ?threshold))
```

```
   (not (no-vection-detected))
   =>
   (bind ?msg (str_cat "Avg. vection intensity mean is less than " ?threshold "%"))
   (assert (Average_Vection_Intensity conclusion potentially_interesting ?msg)))

(defrule high_average_vection_intensity
; Average vection intensity is interesting if it's consistently > threshold
   (environment flight)
   (subject ?subj)
   (BDC-datum ?subj hi_average ?threshold)
   (Average_Vection_Intensity experiment_result ?mean_found&:(> ?mean_found ?threshold))
   (not (no-vection-detected))
=>
   (bind ?msg (str_cat "Avg. vection intensity mean is greater than " ?threshold "%"))
   (assert (Average_Vection_Intensity conclusion potentially_interesting ?msg)))

;; Average vection is interesting if tactile > free-float
(defrule free-below-bungee--interesting--average-vection
   (body_position free-flt)
   (Average_Vection_Intensity experiment_result ?ff)
   (Average_Vection_Intensity running_mean ?val&:(> ?val ?ff)) ; bungee mean
   =>
   (assert (Average_Vection_Intensity conclusion potentially_interesting
         "Subj's ave. vection < bungee cond. ave. vection")))

(defrule bungee-above-free--interesting--average-vection
   (body_position bungee)
   (Average_Vection_Intensity experiment_result ?b)
   (Average_Vection_Intensity running_mean ?val&:(> ?b ?val)) ; free-float mean
   =>
   (assert (Average_Vection_Intensity conclusion potentially_interesting
         "Subj's ave. vection > free-float cond. ave. vection")))


;; DROPOUTS

;; Number of dropouts is interesting if it's consistently 0 under tactile conditions
(defrule interesting_dropouts_tactile
   (environment flight)
   (body_position bungee)
   (Dropouts experiment_result 0)
   (not (no-vection-detected))
   =>
   (assert (Dropouts conclusion potentially_interesting
         "There were no dropouts with bungees attached")))

;; Number of dropouts is interesting if it's consistently >2 under free-float conditions
(defrule interesting_dropouts_free
   (environment flight)
   (body_position free-flt)
   (Dropouts experiment_result ?mean_found&:(> ?mean_found 2))
   =>
   (assert (Dropouts conclusion potentially_interesting
         "Mean number of free-float dropouts is greater than 2")))

;; Number of dropouts is interesting if tactile consistently < free-float
(defrule free-above-bungee--interesting--dropouts
   (body_position free-flt)
   (Dropouts experiment_result ?ff)
   (Dropouts running_mean ?val&:(> ?ff ?val)) ; bungee mean
   =>
   (assert (Dropouts conclusion potentially_interesting
         "Subj's dropout count > bungee cond. dropout count")))
```

```
(defrule bungee-below-free--interesting--dropouts
   (body_position bungee)
   (Dropouts experiment_result ?b)
   (Dropouts running_mean ?val&:(< ?b ?val)) ; free-float mean
   =>
   (assert (Dropouts conclusion potentially_interesting
         "Subj's dropout count < free-float cond. dropout count")))


;;; OUTPUT STATS

(defrule output_idf_stats
   (?parameter experiment_result ?mean)
   (?parameter standard deviation ?SD)
   (body_position ?cond)
   (subject ?subj)
   =>
   (fprintout idf-stats "Subject: " ?subj " Cond: " ?cond " " ?parameter " mean: " ?mean " SD: " ?SD crlf))

;; Output Interestingness info to "Session History" file for Session Manager
(defrule record-interestingness--start
   (declare (salience -100))
   (?parameter conclusion ?interesting ?source)
   ?f <- (idf-result not-interesting)
   (disk-drive ?disk-drive)
   (interface-directory ?interface-dir)
=>
   (retract ?f)
   (assert (idf-result interesting))
   (open (str-cat ?disk-drive ?interface-dir "history-session") history-session "a")
   (assert (record-interestingness)))

; potentially_interesting      => medium
; certainly_interesting        => high
(defrule record-interestingness
   (record-interestingness)
   ?int <- (?parameter conclusion ?interesting ?source)
   (subject ?subj)
   (body_position ?cond)
   (current-step ?step)
   (this-session ?session)
=>
   (retract ?int)
   (if      (eq ?interesting potentially_interesting)
   then     (bind ?level medium)
   else     (bind ?level high))
   (fprintout history-session "(int-hist class interesting session " ?session " step " ?step " subj "
         ?subj " cond " ?cond " source " ?source " level " ?level ")" crlf))

(defrule record-interestingness--end
   (record-interestingness)
   (not (?parameter conclusion ?interesting ?source))
=>
   (close)
   (assert (ctrl--stop idf)) ; inhibit rules-control "abnormal" message
   (printout "hyperclips" "interesting")) ; return to HyperCard

(defrule no-interesting-results
   (declare (salience -200))
   (idf-result not-interesting)
=>
   (close)
   (assert (ctrl--stop idf))
   (printout "hyperclips" "as-expected")) ; return to HyperCard
```

A CLIPS-BASED EXPERT SYSTEM FOR THE EVALUATION AND SELECTION OF ROBOTS

Mohamed A. Nour
Felix O. Offodile
Gregory R. Madey
(gmadey@synapse.kent.edu)
Administrative Sciences
Kent State University
Kent, Ohio 44242
USA

ABSTRACT

This paper describes the development of a prototype expert system for the intelligent selection of robots for manufacturing operations. The paper first develops a comprehensive, three-stage process to model the robot selection problem. The decisions involved in this model easily lend themselves to an expert system application. A rule-based system, based on the selection model, is developed using the CLIPS expert system shell. Data about actual robots is used to test the performance of the prototype system. Further extensions to the rule-based system for data handling and interfacing capabilities are suggested.

INTRODUCTION

Many aspects of today's manufacturing activities are increasingly being automated in a feverish pursuit of quality, productivity, and competitiveness. Robotics has contributed significantly to these efforts; more specifically, industrial robots are playing an increasingly vital role in improving the production and manufacturing processes of many industries [6].

The decision to acquire a robot, however, is a nontrivial one, not only because it involves a large capital outlay that has to be justified, but also because it is largely complicated by a very wide range of robot models from numerous vendors [6]. A non-computer-assisted (manual) robot selection entails a number of risks, one of which is that the selected robot might not meet the task requirements; even if it does, it might not be the optimal or the most economical one. Mathematical modeling techniques, such as integer programming, are rather awkward and inflexible in tackling this problem. The reason for this is that the robot selection process is an ill-structured and complex one, involving not only production and engineering analysis, but also cost/benefit analysis and even vendor analysis. Its ill-structured nature does not readily lend itself to tractable mathematical modeling. Therefore, nontraditional approaches, such as expert systems (ES) or artificial neural networks (ANN), seem intuitively appealing tools in these circumstances.

When the decision maker (DM) is charged with making the selection decision, he or she is being called upon to play three roles at the same time, namely (1) financial analyst, (2) robotics expert, and (3) production manager. In other words, the decision maker would need to make three different (albeit related) decisions: (1) choosing the best robots that match the task requirements at hand, (2) choosing the most cost effective one(s) from those that meet the requirements, and (3) deciding from which vendor to order the robot(s). We shall call these decisions *technical*, *economic*, and *acquisitional*, respectively. Clearly, these are very complex decisions all to be made by the same decision maker. Supporting these decisions (e.g., by a knowledge-based system) should alleviate the burden from the decision maker and bring some consistency and confidence in the overall selection process. The success of the ES technology in a wide range of application domains and problem areas has inspired its use as a vehicle for automating decisions in production and operations management [1, 19], as well as the robot selection decision [16, 17].

In this paper, a three-stage model is presented for the robot selection process. The model is *comprehensive* enough to include the major and critical aspects of the selection decision. It is implemented in a CLIPS-based prototype expert system. The rest of the paper is organized as follows. In the following section, we review previous work and, in the third section, we present our three-stage model to robot selection. In the fourth section, the implementation of the prototype expert system is discussed. Limitations of, and extensions to the prototype expert system with database management (DBMS) capabilities are provided in section five. We conclude the paper in section six.

MOTIVATION AND RELATED WORK

The application of knowledge-based systems in production and operations management has been investigated by a number of researchers [1]. In particular, the application of ES in quality control [3], in job shop scheduling [18, 19], and industrial equipment selection [11, 20] has been reported in the literature. The robot selection problem is prominent in this line of research [8, 15, 17].

In an earlier paper by Knott and Getto [9], an economic model was presented for evaluating alternative robot systems under uncertainty. The authors used the present value concept to determine the relative worthiness of alternative robot configurations. Offodile, *et al.* [14, 15] discuss the development of a computer-aided robot selection system. The authors developed a coding and classification scheme for coding and storing robot characteristics in a database. The classification system would then aid the selection of the robot(s) that can perform the desired task. Economic modeling can then be used to choose the most cost-effective of those selected robots. Other related work includes Offodile *et al.* [16], Pham and Tacgin [17], and Wang, *et al.* [21].

A review of the above literature indicates that these models are deficient in at least one of the following measures:

- *Completeness:* We suggested earlier that the robot selection problem involves three related decisions. The models presented in the literature deal invariably with at most two of these decisions. The other aspects of the selection decision are thus implicitly assumedto be unimportant for the robot selection problem. Experience suggests that this is not the case, however.
- *Generality:* the models presented are restricted to specific production functions, e.g., assembly. Many of today's production functions are not monolithic but rather a collection of integrated functions, i.e., welding, assembly, painting, etc. In particular, industrial robots are by design multi-functional and a selection model should be robust enough to evaluate them for several tasks.
- *Focus:* The focus in the literature is often more on robot characteristics (robot-centered approach) than on the task the robot is supposed to perform (task-centered approach). We posit that task characteristics should be the primary focus in determining which robot to choose, not the converse.

We propose a three-stage model that captures the overall robot selection process, with primary emphasis being given to the *characteristics* and *requirements* of the *task* at hand. The proposed task-centered model is *comprehensive* in the sense that it covers the robot selection problem from the task identification, through robot selection, to vendor selection and, possibly, order placement. The model is also *general* in the sense that it applies to a wider range of industrial robot applications. While this selection model is different from previous approaches, it incorporates in a systematic manner all the critical decisions in any sound robot selection process. The sequential order of these decisions, and the related phases, is important from a logical as well as an efficiency standpoints. We cannot, for example, separate the technical decision from the economic decision, for a robot that is technically well suited to do the job might not be economical; and vice versa. We

shall present our model in the following section and in the subsequent sections discuss its implementation in a knowledge-based system.

A THREE-STAGE MODEL FOR ROBOT SELECTION

Figure 1 depicts the three-stage robot section scheme proposed in this paper. We present a general discussion of the scheme in the subsequent subsections.

- *Technical decision:* This is the first and the most critical decision to be made. It is the formal selection of one or more candidate robots that satisfy the minimum requirements and characteristics of the task to be performed. It is technical in the sense that it would normally be made by the production or process engineer upon a careful analysis of the technical characteristics of both the task and the robot. This decision is the most difficult of the three. A thorough analysis is required to arrive at the initial set of feasible robots.

- *Economic decision:* This is a decision involving the economic merit of the robot. More specifically, it is a decision about the most cost-effective robot alternative(s) considering both initial cost (purchase price) and operating costs. The purpose of the initial and operating costs is twofold: (1) to allow for a rough justification for the robot, and (2) to allow for a choice to be made among rival robots. Suppose, for example, that we had a choice of two robots from Stage One (to be described shortly)—one which is adequate for the task and costs within reasonable range; the other is more technologically advanced but costs well beyond what is considered economical for the task in question. Clearly the estimated cost of the latter would force us to choose the former robot.

- *Acquisitional decision:* This is simply deciding which vendor to acquire the robot(s) from. The choice of a vendor is based not only on purchase price, but also on service and quality.

The following three stages implement the above decisions in a systematic manner.

Stage One: *Technical Decision*

The purpose of this first stage is to determine a (possibly set of) robot(s) that most closely matches the task requirements. The starting point is the application area, or more specifically, the task itself for which the robot is needed. Thus, we need to determine in this stage the following:

1. The application area, e.g., assembly, welding, painting, etc.
2. The task within the application area, e.g., small parts assembly.
3. The task requirements, e.g., precision, speed, load.
4. The robots that most fully satisfy these requirements.
5. Whether human workers can perform the task.
6. Whether to go with robots or humans, if 5 above is true.

Identifying Application:

There is a wide range of applications, across various industries, for which industrial robots may be engaged. Both the application area and the narrow task within that application area should be identified. Thus, within welding, for example, we would identify spot welding and arc welding.

Identifying Task Characteristics:

This phase requires identification of all the task characteristics that influence the decision to employ humans or robots, and the selection among alternative, technically feasible robots. These character-

istics will include, for example, the required degree of precision and accuracy; whether it is too hazardous, dangerous, or difficult for humans; and whether significant increases in productivity and/or cost savings would result.

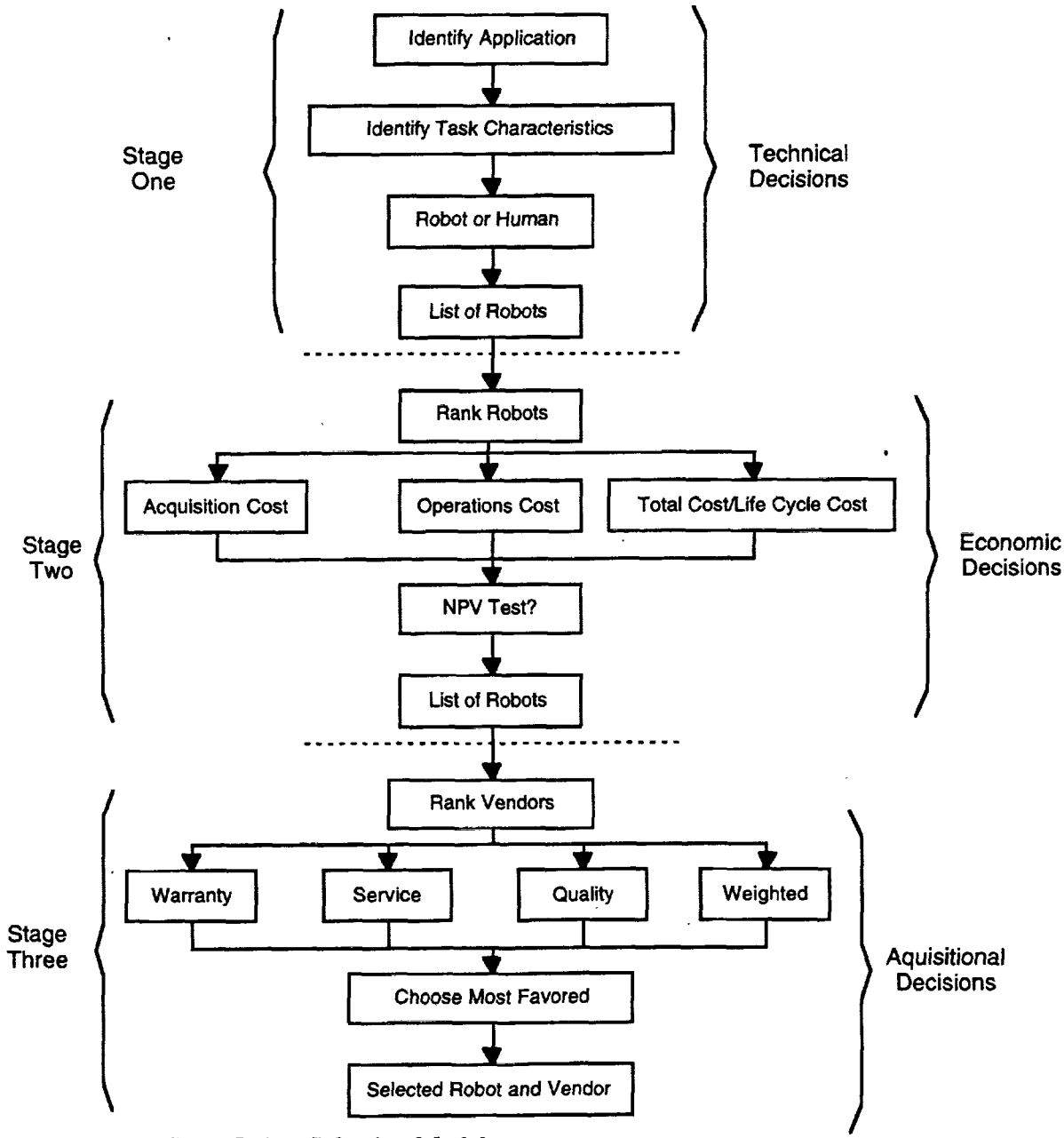


Figure 1: Three-Stage Robot Selection Model

A precise task definition might also require a task classification scheme, more fine-tuned than the one suggested by Ayres and Miller [2]. Since the desired robot is a function of the complexities of the task in question, we suggest the development of a task/robot grid (TRG) to associate specific task characteristics with relevant robot attributes. Let C_{ij} denote value j of task characteristic i, and A_{ij} denote value j of robot attribute i, i=1, 2, ..., m; j=1, 2, ..., n. Here C_{ij} is said to be compatible with A_{ij}, for particular values of i and j, if A_{ij} satisfies C_{ij}. For brevity, we denote this relationship

by $C_{ij} \in A_{ij}$. Thus specifying task characteristic C_{ij} would automatically call for a robot attribute A_{ij} such that A_{ij} at least satisfies the task requirement $C_{ij}, \in i,j$.

Stage Two: *Economic Decision*

This stage can be called a cost justification stage. It utilizes the output from the first stage, which is one or more robots suited to the task at hand. The primary role of this stage is the identification of those robots that make economic sense to invest in; a present value approach is followed by the knowledge-based system to exclude all robots with net present value (NPV) less than their net cost (i.e., purchase price and operating costs). A ranking on the basis of the cost factor is then applied to the remaining robots, if any, i.e., to those passing the economic viability test. Thus, net present value analysis is used to determine whether it is profitable to employ any robot, given its net cost and the economic benefits (e.g., incremental cash flows) expected to accrue as a result of employing the robot to perform the task.

By the end of Stage Two we will have identified a subset of robots that are the most favorable in terms of performance as well as cost. Since a large number of vendors may be available, it is important to be able to get the "best" deal possible. This implies not only a good competitive price, but also acceptable quality, warranties, and a promise of support services.

Stage Three: *Acquisitional Decision*

In Stage three we have to rank, for every vendor, every robot that meets the choice criteria in Stages 1 and 2. The factors that are involved in these rankings are many. For example, Hunt [6] indicates that a certain study revealed the following factors as critical in the purchasing decisions of robots: design, performance, cost, maintenance, warranties, financial terms, and delivery terms. These can conveniently be grouped into four categories: (1) cost (purchase price and operating costs), (2) warranties, (3) quality (performance and design), and (4) service (support, financial and delivery terms). Maintenance is part of operating cost which is accounted for in Stage Two. Quality, services, warranties, and purchase price are the relevant factors in vendor selection. Purchase price has also played a role in the economic decision to determine the viability of each robot. Here, prices are used to compare vendors and rank robots accordingly. Therefore, for each vendor we rank the relevant robots on the basis of purchase price and the other three criteria (quality, warranties, services) and choose the most favorable vendor/robot combination.

THE KNOWLEDGE-BASED SYSTEM

We implemented the prototype knowledge-based system using the CLIPS expert system tool [12]. CLIPS is a forward-chaining rule-based language that resembles OPS5 and ART, two other widely known expert system tools [5]. Developed by NASA, CLIPS has shown an increasing popularity and acceptance by end users [10]. The two main components of this prototype, the knowledge base and the database, are discussed below.

The Knowledge Base

The primary focus of the selection model is the task characteristics, since it is these characteristics that determine what type of robot is needed. This emphasis is reflected in the knowledge base (KB) (or rule base) which captures the knowledge about different task requirements that are needed to justify the use of robots or humans and to specify a particular matching robot or robots. Thus, given certain task characteristics or requirements, the expert system will specify the most suitable robot configurations for performing the task.

The KB also includes knowledge about robot costs and how to manipulate these costs to justify (economically) the use of the respective robots and rank those robots that are considered justifiable. Thus, given operating and purchase costs for each robot, the expert system ranks them on the combined cost criterion. Finally, the KB also includes knowledge to help vendor selection. Subjective multiple criteria are used to compare vendors with associated robots of interest.

As a rule-based shell, CLIPS stores the knowledge in rules, which are logic-based structures, as shown in Figure 2. Figure 3 is a natural English counterpart of the rule in Figure 2.

```
;Rule No 29.
(Defrule find-robots-3rd-pass  "Technical Features"
     ?f <- (robot-2 ?robot)
          (Features (Accuracy ?v11)
                    (Repeat ?v12)
                    (Velocity ?v13)
                    (Payload ?v14)
          )
          (Robot     (ID ?robot)
                    (Accuracy ?v21&:(<= ?v21 ?v11))
                    (Repeat ?v22&:(<= ?v22 ?v12))
                    (Velocity ?v23&:(<= ?v23 ?v13))
                    (Payload ?v24&:(<= ?v24 ?v14))
          )
          =>
          (retract ?f)
          (assert (robot-3 ?robot)
)
```

Figure 2: Example Rule in CLIPS

```
Rule No 29: Finds robots matching given technical
                features.

IF      There is a robot for the application with the
        required grippers,

AND     This robot meets (at a minimum) the following
        technical features: Accuracy, Repeatability,
        Velocity, and Payload as specified by user

THEN    Add this robot to the set of currently feasible
        robots.
```

Figure 3: Example Rule in Natural English

The Database

The database is a critical resource for the ES; all details for robot configurations are contained in the database. The type of information stored for each robot includes:

- Robot class or model
- Performance characteristics
- Physical attributes

- Power characteristics

A full-fledged system would include the following additional information to permit proper comparisons among competing robots.

- Environment requirements
- General characteristics
- Operating costs

Figure 4 shows the information stored in the database for a typical robot using CLIPS syntax. As the figure indicates, each robot has a set of physical and performance characteristics (e.g., type of end effectors, number of axes, repeatability, etc.) and a set of application tasks within its capability. All of this information (and more) is supplied by robot vendors.

```
(Robot (ID RT001)
        (Control S2)
        (Power E)
        (Axes 6)
        (Accuracy .2)
        (Repeat .05)
        (Velocity 145)
        (Payload 6)
        (Effectors adjust-jaw)
        (Jobs ML PT SA EA IN)
        (Vendor "IBM Corp.")
)

(Vendor (ID VD001)
         (Name "IBM Corp.")
         (Robot-Info RT001 28500)
            (Service .95)
            (Warranty .8)
            (Quality .83)
)
```

Figure 4: "Facts" Stored as Frames in a CLIPS Database

Also contained in the database are vendor attributes such as service record, warranties, quality, robots carried and purchase prices (see Figure 3). The first three attributes are represented in the database by subjective scores (ratings), on a scale of 0 to 10. A "0" may indicate, for example, an F rating, "10" an A++ rating. This information could come from industry analysts and experts in the robotics industry.

Illustration

In this section we shall provide the results of a consultation with the prototype expert system using actual robotics data obtained from Fisher [4]. The first step in Stage One is to describe the application. For lack of space, we skip the dialogue that allows the decision maker to describe the task and its suitability for robots or human workers. On the basis of the information provided in that dialogue advice will be given as to the choice between a robot solution or human workers for the task. Next, in Stage Two, economic analysis is performed, using information elicited through a similar dialogue as in Stage one, to determine the economic merit of each robot passing the technical test. This involves calculating a net present value (NPV) for the net incremental benefits resulting from employing robotics in the task under consideration. The NPV is then compared to the net cost

(price plus operating cost) of each robot, and only robots whose net cost is less than or equal to the NPV are chosen. If no robot is found to meet this test, the system offers two reasons for the failure:

1. that the task is not worth the required investments in robots, or
2. that the database includes insufficient number of robots.

In the last stage, Stage Three, the system elicits subjective input from the decision maker regarding the importance of vendor attributes, such as service or warranties. Again, the rating is on a scale of 0 to 10; "0" indicates unimportant and "10" maximally important. This information is then used to compute a subjective score for each vendor, by weighting the analyst's ratings of each vendor with the input from the decision maker. Now, robots can be ranked by both price and vendor weighted score. Figure 5 shows the final results of this consultation.

```
Indicate the importance of each of the following,
 on a scale from 0 to 10:
  1. Vendor service quality :7
  2. Vendor warranties :8
  3. Product quality :9
  4. Price :6
.....
Rank by Subjective score (S) or by Price (P)? p

Robot Index    Vendor Name              Price     Score
-----------    -----------              -----     ----
RT005          ASEA, Inc.               $60000     17
RT007          Bendix                   $70000     13
RT011          Cincinnati Milacron      $90000     17
RT006          Automatix, Inc.          $95000     18
RT008          Bendix                   $95000     13
RT016          Kuka                    $125000     13

The net present value of cashflows:    $129036.6
```

Figure 5: Subjective Values and Final Results

LIMITATIONS AND EXTENSIONS

The description of the task as allowed by the current prototype provides only a broad definition of the nature of the task to be performed; it does not provide specific details or "tolerances." For example, to increase the chances of a match, the user may be tempted to supply larger (less tighter) values for positional accuracy and repeatability. However, this may result in a large number of robots being selected and the prototype system allows ranking only through price and vendor attributes. To rank robots for each and every attribute, however, would probably be both unwieldy and unrealistic.

Moreover, a knowledge-based robot selection system should provide a friendly interface that allows the decision maker to input English phrases to describe a particular application; the system would then use the task definition thus provided by the user to suggest applicable robot(s). Therefore, the crucial task of the knowledge-based system would be to make sense out of the English phrases supplied by the decision maker to describe the task. This implies that the knowledge-based system would have to have some natural language processing capability to properly

associate the task description with meanings represented in the knowledge base. This, then, would pave the way for processing the applicable knowledge to reach a choice of a set of robots. Additionally, as mentioned earlier, the database component of the expert system needs to store a wealth of information about a wide range of robots and vendors. This information can be not only very detailed, but volatile as well, as the robot technology advances and as competition among vendor produces changes in vendor profiles. What all this amounts to is that the expert system needs to be able to survive the test of time and handle this voluminous data in a graceful manner. Current expert systems exhibit elementary data management capabilities which are inadequate for this inherently complex database. As Jarke and Vassiliou [7] indicate, a generalized Database Management System (DBMS) integrated in the ES may be necessary to deal with this database effectively. These authors sketch out four ways to extend expert systems with DBMS capabilities, not all of which are relevant in any given circumstances.

CONCLUSION

We presented in this paper a robot selection model based on a three-stage selection process; each stage feeds its output into the next stage until a final robot/vendor combination is selected. We implemented this model in a prototype knowledge-based system using the CLIPS expert system language. The prototype indicated that a full fledged expert system will be practical and can be extremely useful in providing a consistent and credible robot selection approach.

Further work is needed to improve the granularity and natural language processing capability of the system. Also needed is research into possibilities of extending the database management capabilities of the robot selection system by coupling it with a database management system.

REFERENCES

1. Anthony, Tery B. and Hauser, Richard D., "Expert Systems in Production and Operations Management: Research Directions in Assessing Overall Impact," INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH, Vol.29, No.12, 1991, pp. 2471-2482.

2. Ayres, Robert U. and Miller, Steven M., ROBOTICS: APPLICATIONS AND SOCIAL IMPLICATIONS, Ballinger Publishing Co., 1983.

3. Fard, Nasser S. and Sabuncuoglu, Ihsan, "An Expert System for Selecting Attribute Sampling Plans," INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH, Vol.3, No.6, 1991, pp. 364-372.

4. Fisher, E.L., "Industrial Robots Around the World," in Nof, S.Y., ed., HANDBOOK OF INDUSTRIAL ROBOTS, John Wiley & Sons, New York, 1985. pp. 1305-1332.

5. Giarratano, Joseph and Riley, Gary. EXPERT SYSTEMS : PRINCIPLES AND PROGRAMMING, PWS-KENT Publishing Company, Boston, 1989.

6. Hunt, V. Daniel, UNDERSTANDING ROBOTICS, Academic Press Inc., 1990.

7. Jarke, Matthias and Vassilliou, Yannis, "Coupling Expert Systems With Database Management Systems," EXPERT DATABASE SYSTEMS: PROCEEDINGS OF THE FIRST INTERNATIONAL WORKSHOP,, L. Kerschberg, (Ed.), Benjamin/Cummings, 1986, pp. 65-85.

8. Jones, M. S., Malmborg, C. J. and Agee, M. H., "Decision Support System Used for Robot Selection," INDUSTRIAL ENGINEERING, Vol. 17, No. 9, September, 1985, pp. 66-73.

9. Knott, Kenneth, and Getto, Robert D., "A model for evaluating alternative robot systems under uncertainty," INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH, Vol. 20, No. 2, 1982, pp. 155-165.

10. Martin, Linda and Taylor, Wendy, A BOOKLET ABOUT CLIPS APPLICATIONS, NASA, Lyndon B. Johnson Space Center, 1991.

11. Malmborg, C., Agee, M. H., Simons, G. R. and Choudhry, J.V., "A Prototype Expert System For Industrial Truck Selection," INDUSTRIAL ENGINEERING, Vol. 19, No. 3, March 1987, pp. 58-64.

12. NASA, Lyndon B. Johnson Space Center, CLIPS BASIC PROGRAMMING GUIDE, 1991.

13. Nof, S.Y., Knight, J. L., JR, and Salvendy, G., "Effective Utilization of Industrial Robots-A Job and Skills Analysis Approach," AIIE TRANSACTIONS, Vol. 12, No. 3, pp. 216-224.

14. Offodile, Felix O. and Johnson, Steven L., "Taxonomic System for Robot Selection in Flexible Manufacturing Cells," JOURNAL OF MANUFACTURING SYSTEMS, Vol. 9, No.1, 1987, pp. 77-80.

15. Offodile, Felix O., Lambert, B.K. and Dudek, R.A., "Development of a computer aided robot selection procedure (CARSP)," INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH, Vol. 25, No. 8, 1987, pp. 1109-1121.

16. Offodile, F. O., Marcy, W. M. and Johnson, S.L., "Knowledge base design for flexible robots," INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH, Vol. 29, No. 2, 1991, pp. 317-328.

17. Pham, D.T. and Tacgin, E., "DBGRIP: A learning expert system for detailed selection of robot grippers," INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH Vol. 29, No. 8, 1991, pp. 1549-1563.

18. Pluym, Ben Van Der, "Knowledge-based decision-making for job shop scheduling," INTERNATIONAL JOURNAL OF COMPUTER INTEGRATED MANUFACTURING, Vol. 3, No. 6, 1990, pp. 354-363.

19. Shah, V., Madey, G., and Mehrez, A., "A Methodology for Knowledge-based Scheduling Decision Support," OMEGA: INTERNATIONAL JOURNAL OF MANAGEMENT SCIENCES, Vol. 20, No. 5/6, 1992, pp. 679-703.

20. Trevino, J., Hurley, B.J., Clincy, V. and Jang, S.C., "Storage and industrial truck selection expert system (SITSES)," INTERNATIONAL JOURNAL OF COMPUTER INTEGRATED MANUFACTURING, Vol. 4, No. 3, 1991, pp. 187-194.

21. Wang, M-J., Singh, H. P. and Huang, W. V., "A decision support system for robot selection," DECISION SUPPORT SYSTEMS, Vol. , 1991, pp. 273-283.

omit

Session 3B: Knowledge Acquisition and CLIPS/External Software Integration

Session Chair: Keith Levi

ADDING INTELLIGENT SERVICES TO AN OBJECT ORIENTED SYSTEM.

Bret R. Robideaux and Dr. Theodore A. Metzler

LB&M Associates, Inc.
211 SW A Ave.
Lawton, OK 73505-4051

(405) 355-1471

robidb@lbm.com metzlert@lbm.com

Abstract

As today's software becomes increasingly complex, the need grows for intelligence of one sort or another to become part of the application- often an intelligence that does not readily fit the paradigm of one's software development.

There are many methods of developing software, but at this time, the most promising is the Object Oriented (OO) method. This method involves an analysis to abstract the problem into separate 'Objects' that are unique in the data that describe them and the behavior that they exhibit, and eventually to convert this analysis into computer code using a programming language that was designed (or retrofitted) for OO implementation.

This paper discusses the creation of three different applications that are analyzed, designed, and programmed using the Shlaer/Mellor method of OO development and C++ as the programming language. All three, however, require the use of an expert system to provide an intelligence that C++ (or any other 'traditional' language) is not directly suited to supply. The flexibility of CLIPS permitted us to make modifications to it that allow seamless integration with any of our applications that require an expert system.

We illustrate this integration with the following applications:

1. An After Action Review (AAR) station that assists a reviewer in watching a simulated tank battle and developing an AAR to critique the performance of the participants in the battle.
2. An embedded training system and over-the-shoulder coach for howitzer crewmen.
3. A system to identify various chemical compounds from their infrared absorption spectra.

Keywords - Object Oriented, CLIPS

INTRODUCTION

The goal of the project was to develop a company methodology of software development. The requirement was to take Object Oriented Analysis (done using the Shlaer-Mellor method) and efficiently convert it to C++ code. The result was a flexible system that supports reusability, portability and extensibility. The heart of this system is a software engine that provides the functionality the Shlaer/Mellor Method (Ref. 2 & 3) allows the analyst to assume is available. This includes message passing (inter-process communication), state machines and timers. CLIPS provides an excellent example of one facet of the engine. Its portability is well-known, and its extensibility allowed us to embed the appropriate portions of our engine into it. We could then modify its behavior to make it pretend it was a natural object or domain of objects. This allowed us to add expert system services to any piece of software developed using our method (and on any platform to which we have ported the engine).

OBJECT ORIENTED METHODS

The principal concept in an OO Method is the object. Many methods include the concept of logically combining groups of objects. The logic behind determining which objects belong to which grouping is as varied as the names used to identify the concept. For example, Booch (Ref. 4) uses categories and Rumbaugh (Ref. 5) uses aggregates. Shlaer/Mellor uses domains (Ref. 3). Objects are grouped into Domains by their functional relationships; that is, their behavior accomplishes complementary tasks.

Fig. 1 shows that the operating system is its own domain. It is possible the operating has very little to do with being OO, but it is useful to model it as part of the system. Another significant domain to note is the Software Architecture Domain. This is the software engine that provides the assumptions made in analysis. The arrows denote the dependence of the domain at the beginning of the arrow upon the domain at the end of the arrow. If the model is built properly, a change in any domain will only affect the domains immediately above it. Notice there are no arrows from the main application (Railroad Operation Domain) to the Operating System Domain. This means a change in operating systems (platforms) should not require any changes in the main application. However, changes will be required in the domains of Trend Recording, User Interface, Software Architecture and Network. The extent of those changes is dependent on the nature of the changes in the operating system. But this does stop the cascade effect of a minor change in the operating system from forcing a complete overhaul of the entire system.

While CLIPS could be considered an object, the definition of domains makes it seems more appropriate to call it a domain.

THE SOFTWARE ENGINE

The portion of the software engine that is pertinent to CLIPS is the message passing. The event structure that is passed is designed to model closely the Shlaer/Mellor event.

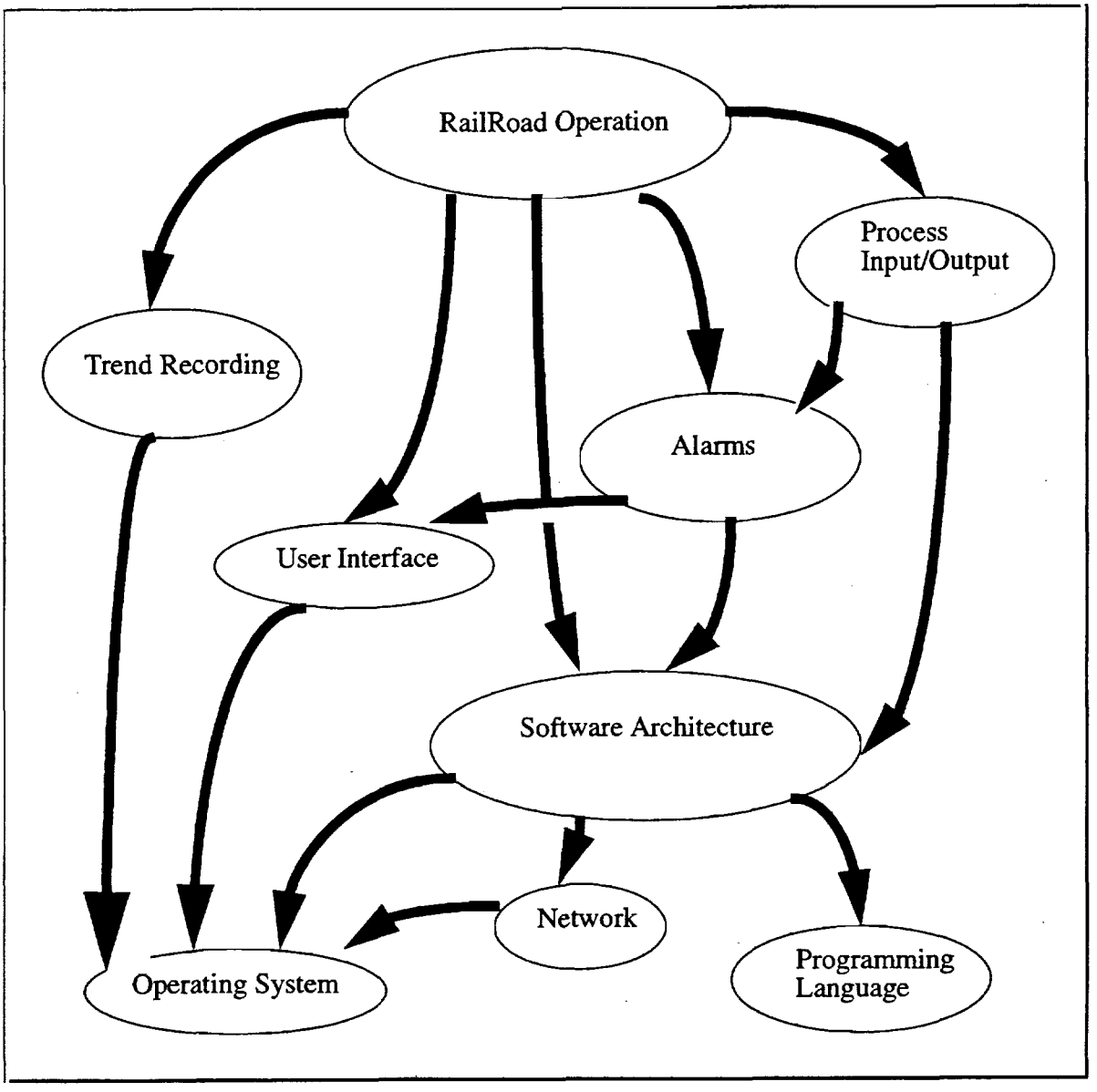


Figure. 1: Domain Chart for the Automated Railroad Management System
Copied from page 141 (Figure 7.4.1) of Ref 3

We have implemented five (5) kinds of events:

1. Standard Events: events passed to an instance of an object to transition the state model
2. Class Events: events dealing with more than one instance of an object
3. Remote Procedure Calls (RPCs): An object may provide functions that other objects can call. One object makes an RPC to another passing some parameters in, having some calculations performed and receiving a response.
4. Accesses: An object may not have direct access to data belonging to another object. The data other objects need must be requested via an accessor.
5. Return Events: the event that carries the result of an RPC or Access event

The event most commonly received by CLIPS is the Class Event. CLIPS is not a true object. Therefore it has no true instances, and Standard Events are not sent to it. CLIPS is more likely to make an RPC than to provide one. It is possible that an object may need to know about a certain fact in the fact base, so receiving an Access is possible. Should CLIPS use an RPC or Access, it will receive a Return Event. RPCs and Accesses are considered priority events so the Return Event could be handled as part of the function that makes the call.

The events themselves have all of the attributes of a Shlaer/Mellor event, plus a few to facilitate functionality. The C++ prototype for the function Send_Mesg is:

```
void Send_Mesg (char* Destination,
                EventTypes Type,
                unsigned long EventNumber,
                char* Data,
                unsigned int DataLength = 0,
                void* InstanceID = 0,
                unsigned long Priority = 0);
```

The engine is currently written on only two different platforms: a Sun 4 and an Amiga. CLIPS, with its modifications, was ported to the Amiga --along with some applications software that was written on the Sun-- and required only a recompile to work. This level of portability means that only the engine needs to be rewritten to move any application from one platform to another. Once the engine is ported, any applications written using the engine (including those with CLIPS embedded in them) need only be recompiled on the new platform and the port is complete. Companies are already building and selling GUI builders that generate multi-platform code so the GUI doesn't need to be rewritten.

The Unix version of the Inter-Process Communication (IPC) part of the engine was written using NASA's Simple Sockets Library. Since all versions of the engine, regardless of the platform, must behave the same way, we have the ability to bring a section of the software down in the middle of a run, modify that section and bring it back up without the rest of the system missing a beat. In the worst case, the system will not crash unrecoverably.

MODIFICATIONS TO CLIPS

Since the software engine is written in C++, at least some part of the CLIPS source code must be converted to C++. The most obvious choice is main.c, but this means the UserFunctions code must be moved elsewhere since they must remain written in C. I moved them to sysdep.c.

The input loop has been moved to main.c because all input will come in over the IPC as valid CLIPS instructions. The stdin input in CommandLoop has been removed, and

the function is no longer a loop. The base function added to CLIPS is:

(RETURN event_number destination data)

This function takes its three parameters and adds them to a linked list of structures (see Fig. 2). When CommandLoop has completed its pass and returns control to main.c, it checks the linked list and performs the IPC requests the CLIPS code made (see Fig. 3). RETURN handles the easiest of the possible event types that can be sent out: Class Events. To handle Standard Events, some way of managing the instance handles of the destination(s) needs to be implemented. To handle Accesses and RPCs some method of handling the Return events needs to be implemented. The easiest way is to write a manager program. Use *RETURN* to send messages to the Manager. The event number and possibly part of the data can be used to instruct the Manager on what needs to be done.

```
RETURN ()
  {
   extract parameters from call

   allocate a new element for the list

   fill in the structure

   add structure to the list

  }
```

Figure. 2: Pseudocode for function RETURN

```
main ()
  {
   init CLIPS

   begin loop

     Get Message

     set command string

     call CommandLoop

     check output list

     if there are elements in the structure
       make the appropriate send messages

   end loop
}
```

Figure 3: Pseudocode for main ()

Sending instructions and information into CLIPS requires a translator to takes Shlaer/Mellor events and convert them to CLIPS valid instructions. In Shlaer/Mellor terms this construct is called a Bridge.

With these changes CLIPS can pretend it is a natural part of our system.

EXAMPLES

This system has been successfully used with three different projects. An After Action Review Station for reviewing a simulated tank battle, an Embedded Training System to aid howitzer crewmen in the performance of their job and a chemical classifier based on a chemical's infrared absorbtion spectra.

The After Action Review (AAR) station is called the Automated Training Analysis Feedback System or ATAFS. See Figure 4 for its domain chart. Its job is to watch a simulated tank battle on a network and aid the reviewer in developing an AAR in a timely manner. It does so by feeding information about the locations and activities of the vehicles and other pertinent data about the exercise to the expert system. The expert system watches the situation and marks certain events in the exercise as potentially important. ATAFS will then use this information to automatically generate a skeleton AAR and provide tools for the reviewer to customize the AAR to highlight the events that really were important.

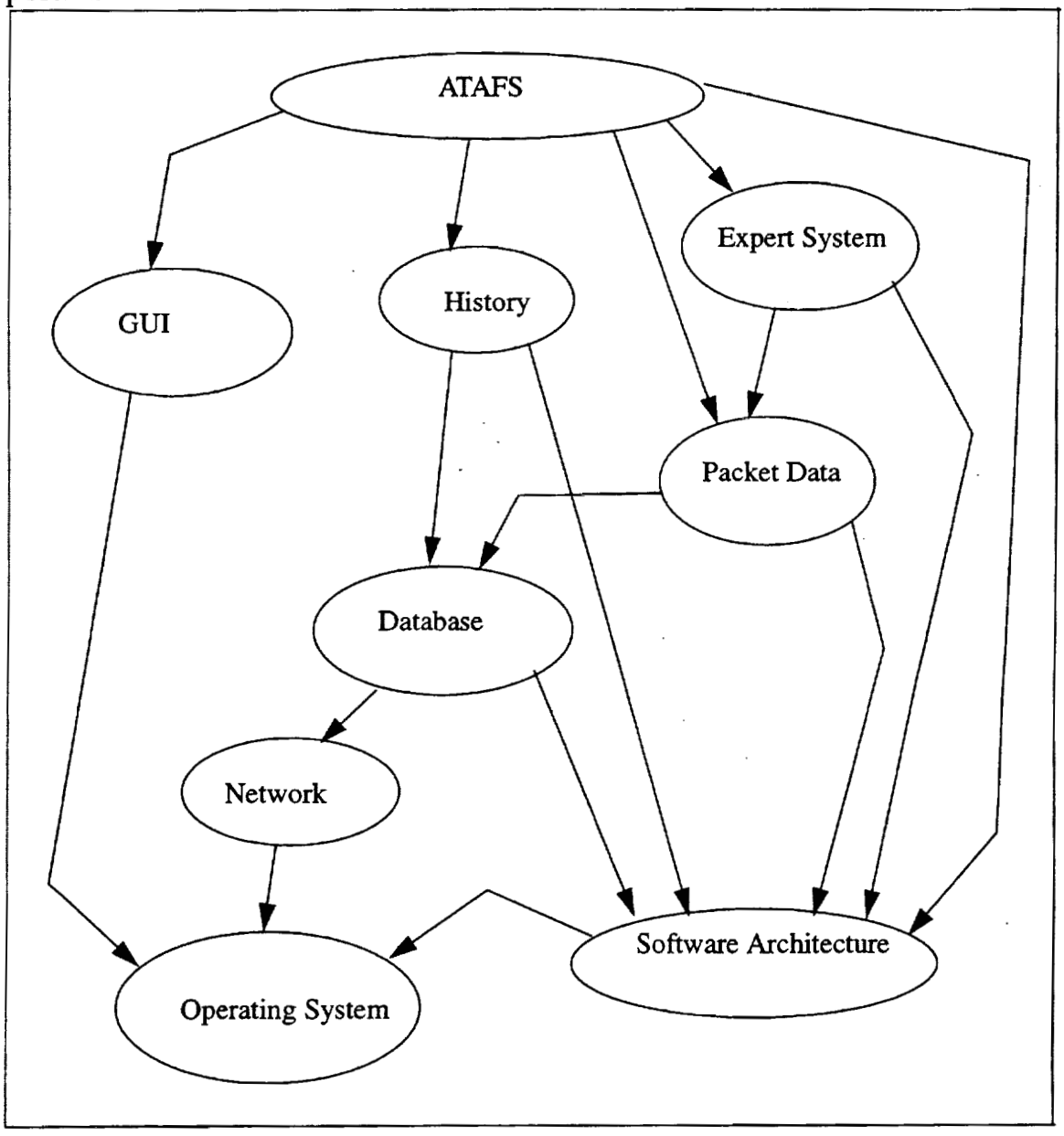


Fig. 4: ATAFS Domain Chart

In an early version of the system, ATAFS was given the location of a line-of-departure, a phase-line and an objective. During the simulated exercise of four vehicles moving to the objective, CLIPS was fed the current location of each vehicle. CLIPS updated each vehicle's position in its fact base and compared the new location with the previous location in relation to each of the control markings. When it found a vehicle on the opposite side of a control mark than it previously was, it *RETURN*ed an appropriate event to ATAFS. Upon being notified that a potentially interesting event had just occurred, ATAFS is left to do what ever needs to be done. CLIPS continues to watch for other events it is programmed to deem interesting.

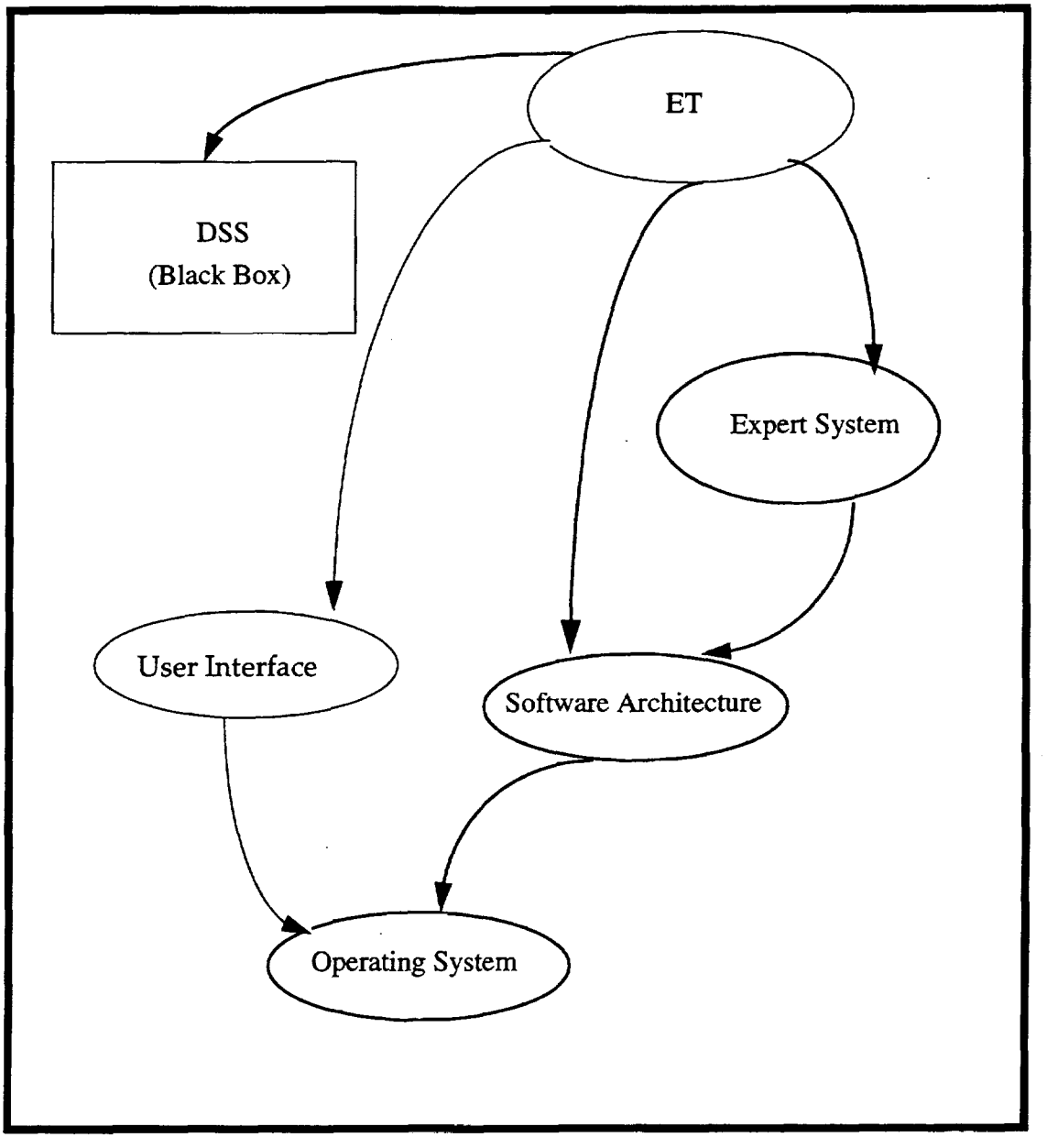


Fig 5: ET Domain Chart

Embedded Training (ET) is a coaching and training system for howitzer crewmen. See Figure 5 for its domain chart. It provides everything from full courseware to simple reminders/checklists. It also provides a scenario facility that emulates real world situations and includes an expert system that watches the trainee attempt to perform the required tasks. During the scenario the expert system monitors the trainee's activities and compares them to the activities that the scenario requires.

It adjusts the user's score according to his actions and uses his current score to predict whether the user will need help on a particular task. All of the user's actions on the terminal are passed along to CLIPS, which interprets them and determines what the he is currently doing and attempting to do. Having determined what the user is doing, CLIPS uses its *RETURN* in conjunction with a manager program to access data contained in various objects. This data is used by CLIPS to determine what level of help, if any, to issue the user. The decision is then *RETURN*ed to ET which supplies the specified help.

The chemical analysis system (Ref. 6) designed by Dr. Metzler and chemist Dr. Gore takes the infrared (IR) spectrum of a chemical compound and attempts to classify it using a trained neural net. In the course of their study, they determined the neural net performed significantly better when some sort of pre-processor was able to narrow down the identification of the compound. CLIPS was one of the methods selected. The IR spectrum was broken down into 52 bins. The value of each of these bins was abstracted into values of strong, medium and weak. Initially three rules were built: one to identify ethers, one for esters and one for aldehydes (Figure 6).

```
(defrule ester
  (bin 15 is strong)
  (or (bin 24 is strong)
      (or (bin 25 is strong)
          (bin 26 is strong))
      (and (or (bin 22 is strong)
               (bin 23 is strong)
               (bin 24 is strong))
           (or (bin 25 is strong)
               (bin 26 is strong)
               (bin 27 is strong)))
      (and (bin 28 is medium)
           (or (bin 22 is strong)
               (bin 23 is strong)
               (bin 24 is strong))))
=>
  (RETURN 1 "IR_Classifier" "compound is an ester"))
```

Figure 6. Example of a Chemical Rule

The results of this pre-processing were *RETURN*ed to the neural net which then was able to choose a module specific to the chosen functional group. Later, the output from another pre-processor (a nearest neighbor classifier) was also input into CLIPS and rules were added to resolve differences between CLIPS' initial choice and the nearest-neighbor's choice.

CONCLUSIONS

CLIPS proved to be a very useful tool when used in this way. In ET and ATAFS, the Expert System Domain could easily be expanded to use a hybrid system similar to that of the Chemical Analysis problem, or use multiple copies of CLIPS operating independently or coordinating their efforts using some kind of Blackboard. In many applications, intelligence is not the main concern. User interface concerns in both ET and ATAFS were more important than the intelligence. ET has the added burden of operating in conjuction with a black box system (software written by a partner company). ATAFS' biggest concern was capturing every packet off of the network where the exercise was occurring and storing them in an easily accessible manner. The flexibility of C++ is better able to handle these tasks than CLIPS, but not nearly as well suited for representing an expert coaching and grading a trainee howitzer crewman or reviewing a tank battle. While recognizing individual chemical compounds would be tedious task for CLIPS (and the programmer), recognizing the functional group the compound belongs to and passing the information along to a trained neural net is almost trivial.

Expert systems like CLIPS have both strengths and weaknesses, as do virtually all other methods of developing software. Object Oriented is becoming the most popular way to develop software, but it still has its shortfalls. Neural nets are gaining in popularity as the way to do Artificial Intelligence, especially in the media, but they too have their limits. By mixing different methods and technologies, modifying and using existing software to interact with each other, software can be pushed to solving greater and greater problems.

REFERENCES

1. Gary Riley, et al, CLIPS Reference Manual, Version 5.1 of CLIPS, Volume III Utilities and Interfaces Guide, Lyndon B. Johnson Space Center (September 10, 1991).

2. Sally Shlaer and Stephen Mellor, Object-Oriented Systems Analysis, Modeling the World in Data. (P T R Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1988).

3. Sally Shlaer and Stephen Mellor, Object Lifecycles, Modeling the World in States. (Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1992).

4. Grady Booch, Object Oriented Design with Applications. (The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991).

5. James Rumbaugh, et al, Object-Oriented Modeling and Design. (Prentice-Hall, Inc., Englewood Cliffs, 1991).

6. T.A. Metzler and R.H. Gore, Application of Hybrid AI to Identification of Chemical Compounds, Proceedings of TECOM Conference on AI and the Environment, Aberdeen Maryland, September 13-16, 1994

CLIPS, AppleEvents, and AppleScript: Integrating CLIPS with Commercial Software

Michael M. Compton
compton@ptolemy.arc.nasa.gov
(415) 604-6776

Shawn R. Wolfe
shawn@ptolemy.arc.nasa.gov
(415) 604-4760

AI Research Branch / Recom Technologies, Inc.
NASA Ames Research Center
Moffett Field, CA 94035-1000

Abstract:

Many of today's intelligent systems are comprised of several software modules, perhaps written in different tools and languages, that together help solve the users' problem. These systems often employ a knowledge-based component that is not accessed directly by the user, but instead operates "in the background" offering assistance to the user as necessary. In these types of modular systems, an efficient, flexible, and easy-to-use mechanism for sharing data between programs is crucial. To help permit transparent integration of CLIPS with other Macintosh applications, the AI Research Branch at NASA Ames Research Center has extended CLIPS to allow it to communicate transparently with other applications through two popular data-sharing mechanisms provided by the Macintosh operating system; Apple Events (a "high-level" event mechanism for program-to-program communication), and AppleScript, a recently-released scripting language for the Macintosh. This capability permits other applications (running on either the same or a remote machine) to send a command to CLIPS, which then responds as if the command were typed into the CLIPS dialog window. Any result returned by the command is then automatically returned to the program that sent it. Likewise, CLIPS can send several types of Apple Events directly to other local or remote applications. This CLIPS system has been successfully integrated with a variety of commercial applications, including data collection programs, electronic forms packages, DBMSs, and email programs. These mechanisms can permit transparent user access to the knowledge base from within a commercial application, and allow a single copy of the knowledge base to service multiple users in a networked environment.

Introduction

Over the past few years there has been a noticeable change in the way that "intelligent applications" have been developed and fielded. In the early and mid-1980s, these systems were typically large, monolithic systems, implemented in LISP or PROLOG, in which the inference engine and knowledge base were at the core of the system. In these "KB-centric" systems, other parts of the system (such as the user interface, file system routines, etc.) were added on once the inferential capabilities of the system were developed. These systems were often deployed as multi-megabyte "SYSOUTs" on special-purpose hardware that was difficult if not impossible to integrate into existing user environments.

Today, intelligent systems are more commonly being deployed as a collection of interacting software modules. In these systems, the knowledge base and inference engine serve as a subordinate and often "faceless" component that interacts with the user indirectly, if at all.

Several such systems have been deployed recently by the Artificial Intelligence Research Branch at NASA's Ames Research Center:

The Superfluid Helium On-Orbit Transfer system (or SHOOT) was a modular expert system that monitored and helped control a Space-Shuttle-based helium transfer experiment which flew

onboard the Endeavor orbiter during mission STS-57 in June, 1993. In that system, a CLIPS-based knowledge system running on a PC laptop on the Shuttle's Aft Flight Deck was embedded in a custom-developed C program that helped crew members (and ground controllers) perform transfer of cryogenic liquids in microgravity.

The Astronaut Science Advisor (also known as "PI-in-a-Box", or simply "[PI]") was another modular system that flew onboard the Space Shuttle in 1993 ([3]). [PI] ran on a Macintosh PowerBook 170 laptop computer and helped crew members perform an experiment in vestibular physiology in Spacelab. This system was comprised of a CLIPS-based knowledge system that interacted with HyperCard (a commercial user interface tool from Claris Corporation and Apple Computer) and LabVIEW (a commercial data acquisition tool from National Instruments) to monitor data being collected by the crew and offer advice on how to refine the experiment protocol based on collected data and, when necessary, troubleshoot the experiment apparatus. In that system, CLIPS ran as a separate application and communicated with HyperCard via specially-written external commands (XCMDs) that were linked into the HyperCard application.

The Prototype Electronic Purchase Request (PEPR) system ([1] and [2]) is yet another modular system that motivated the enhancements being described in this paper. The PEPR system uses a CLIPS-based knowledge system to validate and generate electronic routing slips for a variety of electronic forms used frequently at NASA Ames. A very important part of the PEPR system's capabilities is that it is able to work "seamlessly" with several commercial applications.

Interestingly, these applications represent a progression of integration techniques in which the knowledge-based component interacts with custom-developed programs. This progression starts with tightly-coupled, "linked" integration of components (in SHOOT), to integration via custom-built extensions to a specific commercial product (in [PI]), and finally to a general-purpose integration mechanism that can be used with a variety of commercial products (in PEPR).

Requirements for the PEPR system

A brief description of the PEPR system will help explain how we determined our approach to the problem of integrating CLIPS with other commercial applications.

Our primary goal in the development of the PEPR system was to demonstrate that knowledge-based techniques can improve the usability of automated workflow systems by helping validate and route electronic forms. In order to do this, we needed to develop a prototype system that was both usable and provided value in the context of a real-world problem. This meant that in addition to developing the knowledge base and tuning the inference engine, we also needed to provide the users with other tools. These included software to fill out electronic forms and send them between users, as well as mechanisms that could assure recipients of these forms that the senders were really who they appeared to be, and that the data hadn't been altered, either accidentally or maliciously, in transit. We also wanted to provide users with a way by which to find out the status of previously-sent forms, in terms of who had seen them and who hadn't.

Of course, we didn't want to have to develop all of these other tools ourselves. Thankfully, commercial solutions were emerging to help meet these additional needs, and we tried to use these commercial tools wherever possible. Our challenge became, then, how to best integrate the CLIPS-based knowledge system component with these other commercial tools.

Our first try at component integration: Keyboard macros

Our initial integration effort was aimed at developing a "seamless" interface between the knowledge system and the electronic forms package (which of course was the primary user interface). Our

first try was to use a popular keyboard macro package to simulate the keystrokes and mouse clicks that were necessary to transfer the data from the forms program to CLIPS. The keyboard macro would, with a single keystroke, export the data on the form to a disk file, switch to the CLIPS application, and then "type" the `(reset)` and `(run)` commands (and the requisite carriage returns) into the CLIPS dialog window. This would then cause CLIPS to read in and parse the data from the exported file.

This worked fairly well, and was reasonably easy to implement (that is, it did not require modifying any software). However, the solution had several serious drawbacks. First, it required that the user have the keyboard macro software (a commercial product) running on his or her machine. Also, the macro simply manipulated the programs' user interface, and was therefore very distracting for the user because of the flurry of pull down menus and dialog boxes that would automatically appear and disappear when the macro was run. Most importantly, however, was that the keyboard macro approach required each user to have their own copy of CLIPS and the knowledge base running locally on their machine. This, of course, would have presented numerous configuration and maintenance problems; the "loadup" files would have to be edited to run properly on each user's machine, and fixing bugs would have involved distributing new versions of the software to all users (and making sure they used it). Another drawback was that we found that the keyboard macros would often "break" midway through their execution and not complete the operation. This would, of course, have also been very frustrating for users. As a result, this early system, although somewhat useful for demonstrations, was never put into production.

Our second try: Apple Events

Our next attempt at integrating the electronic forms package with CLIPS was motivated by a demonstration we saw that showed that it was possible to integrate the forms package we had selected with a commercial DBMS. In this demo, a user was able to bring up an electronic order form, and enter information such as, say, a vendor number. When the user tabbed out of the Vendor Number field, the forms software automatically looked up the corresponding information in a "vendor table" in a commercial DB running on a remote machine. The vendor's name and address automatically appeared in other fields on the form.

Of course, this sort of data sharing is quite common in many data processing environments. However, we found this capability exciting because it involved two Macintosh applications (including our forms package of choice), from different vendors, cooperating across an AppleTalk network, with virtually no assistance from the user. This was exactly the sort of "seamless" behavior we were seeking.

This functionality was provided by what are known as "Apple Events", a "high-level event mechanism" that is built into "System 7", the current major release of the Macintosh operating system. We were encouraged by the fact that the forms package we had selected supported the use of Apple Events to communicate with external programs. We thought it might be possible to implement our desired inter-program interface by making CLIPS mimic the DBMS application with respect to the sending and receiving of Apple Events. However, the particular set of Apple Events supported by that version of the forms software was limited to interaction that particular DBMS product. This was problematic for several reasons. First, the set of Apple Events being used were tailored to interaction with this particular data base product, and it wasn't clear how to map the various DB constructs they supported into CLIPS. Second, it would have required a considerable amount of programming effort to develop what would have been the interface between a specific forms product and a specific data base product. We wanted to minimize our dependence on a particular product, in case something better came along in the way of a forms package or a DBMS.

So, we ended up not implementing the DB-specific Apple Event mechanism. However, we were still convinced that the Apple Event mechanism was one of the keys to our integration goals, and *did* implement several more general Apple Event handlers, which are described below.

What are Apple Events?

The following provides a brief description of the Apple Event mechanism. As mentioned above, Apple Events are a means of sharing data between to applications, the support for which is included in System 7. An Apple Event has four main parts. The first two components are a 4-character alphanumeric "event class" and a 4-character "event ID". The third component identifies the "target application" to which the event is to be sent. The actual data to be sent is a collection of keyword/data element pairs, and can range from a simple string to a complex, nested data structure. (In addition to these, there are several other parts of the event record which specify whether a reply is expected, a reply time-out value, and whether the target application can interact with a user.) From a programmer's perspective, these events are created, sent, received, and processed using a collection of so-called "ToolBox" calls that are available through system subroutines.

Although Apple Events are a "general purpose" mechanism for sharing data between programs, programs that exchange data must follow a pre-defined protocol that describes the format of the events. These protocols and record formats are described in the *Apple Event Registry*, published periodically by Apple Computer.

```
Event Class: MISC
   Event ID: DOSC
Target App: <ProcessID>
Event Data: "----"
            "(load \"MYKB.CLP\")"
```

Figure 1: An Example Apple Event

Figure 1 shows an example of a "do script" event that might be sent to CLIPS. The event class is "MISC", and the event ID is "dosc". The 4-dash "parameter keyword" identifies the string as a "direct object" parameter, and is used by the receiving program to extract the data in the event record. The protocol associated with the "do script" event in the Apple Event Registry calls for a single string that contains the "script" to be executed by the target application. In this example, it's a CLIPS command for loading a knowledge base file. This example shows the simplest data format. More complex Apple Events may require numerous (and even embedded) keyword/data pairs.

The Apple Event mechanism is very powerful. However, we wanted a more flexible solution to our needs of integrating CLIPS with the commercial forms application. It would have required considerable effort to implement a Apple-Event based programmatic interface between CLIPS and the other applications with which we needed to share data. That is, we didn't want to have to code

specific Apple Event handling routines for every other application we were using. This would have made it very difficult to "swap in" different applications should we have needed to. What we needed was a way by which we could use the power of Apple Events to communicate between CLIPS and other programs, but make it easier to code the actual events for a given program.

Our third try: AppleScript

Just as we were trying to figure out the best way to link CLIPS with the other programs we wanted to use, we learned about a new scripting technology for the Mac called AppleScript. This new scripting language provides most of the benefits of Apple Events (and is in fact based on the Apple Event mechanism) such as control of and interaction with Macintosh applications running either locally or remotely. In addition, it offers some other benefits that pure Apple Event programming does not. For example, using AppleScript, one is able to control and share data with an ever-growing array of commercial applications, without having to understand the details of the application's Apple Event protocol. AppleScript comes with a reasonably simple Script Editor that can be used to compose, check, run, and save scripts. In addition to providing all the constructs that one might expect in any programming language (variables, control, I/O primitives) it is also extendible and can even be embedded in many applications.

The thing that made AppleScript particularly appealing for our use was that it utilized the Apple Event handlers that we had already added to CLIPS. All that was necessary to permit the "scriptability" we desired was the addition of a new "Apple Event Terminology Extension" resource to our already-modified CLIPS application. This AETE resource simply provided the AppleScript Editor (and other applications) with a "dictionary" of commands that CLIPS could understand, and the underlying Apple Events that the AppleScript should generate.

Another very appealing aspect of integrating programs with AppleScript is more and more commercial software products are supporting AppleScript by becoming scriptable. That of course makes it much easier to take advantage of new software products as they come along. For example, we recently upgraded the forms tracking data base used by the PEPR system. We were able to replace a "flat-file" data base package with a more-powerful relational DBMS product with only minor modifications to the AppleScript code linking the DB to the other applications. This would have been far more difficult (if even possible) had we relied solely on integration with AppleEvents.

The main disadvantage to using AppleScript rather than Apple Events is that AppleScript is somewhat slower than sending Apple Events directly. However, the increased flexibility and power of AppleScript more than compensates for the comparative lack of speed.

```
tell application "CLIPS" of machine "My Mac"
      of zone "Engineering"
   do script "(reset)"
   do script "(run)"
   set myResult to evaluate "(send [wing-1] get-weight)"
   display dialog "Wing weight is " & myResult
end tell
```

Figure 2: An example AppleScript program

Figure 2 shows an example script that could be used to control CLIPS remotely. There are several things to note in this example. First, the commands are passed to CLIPS and executed as though they had been entered into the Dialog Window. The example shows both the "do script" command (which does not return a result) and the "evaluate" command (which does). The example also shows a "display dialog" command which is built in to AppleScript and displays the result in a modal dialog box. Of particular interest is that the CLIPS application is running on another Macintosh, which is even in another AppleTalk zone.

Specific CLIPS Extensions

The following paragraphs describe the actual CLIPS extensions that have been implemented to support the functionality described above. Note that some of these extensions were actually implemented by Robert Dominy, formerly of NASA Goddard Space Flight Center.

Receiving Apple Events

It's now possible to send two types of Apple Events to CLIPS. Each takes a string that is interpreted by CLIPS as though it were a command typed into the Dialog Window. The format of these Apple Events is dictated by the *Apple Event Registry*, and they are also supported by a variety of other applications. Note that CLIPS doesn't currently return any syntax or execution errors to the program that sent the Apple Events, so it is the sender's responsibility to ensure that the commands sent to CLIPS are syntactically correct.

The "do script" Event

The "do script" event (event class = MISC, event ID=DOSC) passes its data as a string which CLIPS interprets as if it were a command that were typed into the Dialog Window. It returns no value to the sending program.

The "evaluate" Event

The "evaluate" event (event class = MISC, event ID=EVAL) is very similar to the do script event, and also passes its data as a string which CLIPS interprets as if it were a command that were typed into the Dialog Window. However, it does return a value to the sending program. This value is always a string, and can be up to 1024 bytes in length.

Sending Apple Events from CLIPS

The two Apple Events described above can also be sent by CLIPS from within a knowledge base. Of course, the application to which the events are sent must support the events or an error will occur. However, as mentioned above, the "do script" and "evaluate" events are very common and supported by many Mac applications.

SendAEScript command

The SendAEScript command sends a "do script" event and can appear in a CLIPS function or in the right-hand-side of a rule. The syntax of the SendAEScript command is as follows:

```
(SendAEScript <target app> <command>)
```

In the above prototype, `<target app>` is an "application specification" and `<command>` is a valid command understandable by the target application. An application specification can have one of three forms; a simple application name, a combination application name, machine name and

AppleTalk Zone name, or a process identifier (as returned by `PPCBrowser`, described below). The `SendAEScript` command returns zero if the command is successfully sent to the remote application, and a variety of error codes if it was not. Note that a return code of zero does *not* guarantee that the command was successfully executed by the remote application; only that it was sent successfully .

The following examples show each of the application specification types.

```
CLIPS>(SendAEScript "HyperCard" "put \"hello\" into msg")
0
CLIPS>
```

The above example sends a "do script" Apple Event to HyperCard running on the local machine, and causes it to put "hello" into the HyperCard message box.

```
CLIPS>(SendAEScript "HyperCard" "John's Mac" "R&D" "put \"hello\" into msg")
0
CLIPS>
```

The above example sends a similar "do script" Apple Event to HyperCard running on a computer called "John's Mac" in an AppleTalk zone named "R&D". Note that it is necessary to "escape" the quote characters surrounding the string "hello" to avoid them being interpreted by the CLIPS reader.

SendAEEval command

The SendAEEval command is very similar to the SendAEScript command, differing only in that it returns the value that results from the target application evaluating the command.

```
(SendAEEval <target app> <command>)
```

The following examples show CLIPS sending a simple command to HyperCard running on the local machine:

```
CLIPS> (SendAEEval  "HyperCard"  "word 2 of \"my dog has fleas\" ")
"dog"
CLIPS>
```

Note that the result returned by SendAEEval is always a string, e.g.:

```
CLIPS> (SendAEEval  "HyperCard"  "3 + 6")
"9"
CLIPS>
```

The `SendAEEval` command does not currently support commands that require the target application to interact with its user. For example, one could not use `SendAEEval` to send an "ask" command to HyperCard.

PPCBrowser command

The `PPCBrowser` function permits the CLIPS user to select an AppleEvent-aware program that is currently running locally or on a remote Mac. This command brings up a dialog box from which the user can click on various AppleTalk zones, machine names and "high-level event aware" applications. It returns a pointer to a "process ID" which can be bound to a CLIPS variable and used in the previously-described "send" commands.

```
CLIPS>(defglobal ?*myapp* = (PPCBrowser))
CLIPS> ?*myapp*
<Pointer: 00FF85E8>
```

The above example doesn't show the user's interaction with the dialog box.

GetAEAddress command

The `GetAEAddress` function is similar to `PPCBrowser` in that it returns a pointer to a high-level aware application that can then be bound to a variable that's used to specify the target of one of the "SendAE" commands described earlier. Rather than presenting a dialog box to the user, however, it instead takes a "target app" parameter similar to that described above.

```
(GetAEAddress <target app>)
```

The following example shows the GetAEAddress function being used to specify the target of a `SendAEEval` function call.

```
CLIPS> (defglobal ?*myapp* = (GetAEAddress "HyperCard" "Jack's Mac" "R&D"))
CLIPS> (SendAEEval ?*myapp*  "8 + 9")
"17"
CLIPS>
```

TimeStamp command

Another extension we've made is unrelated to inter-program communication. We have added a `TimeStamp` command to CLIPS. It returns the current system date and time as a string:

```
CLIPS>(TimeStamp)
"Wed Sep 7 12:34:56 1994"
CLIPS>
```

Possible Future Extensions

In addition to the CLIPS extensions described above, we are also looking into the possibility of implementing several other enhancements. First, we want to generalize the current Apple Event sending mechanisms to permit the CLIPS programmer to specify the event class and event ID of the events to be sent. This is a relatively straightforward extension if we limit the event data to a string passed as the "direct object" parameter. It would be somewhat harder to allow the CLIPS programmer to specify more complex data structures, because we would have to design and implement a mechanism that allows the CLIPS programmer to construct these more complex combinations of keywords, parameters, and attributes. We will probably implement these extensions in stages.

Another extension we're considering is to make CLIPS "attachable". This would permit the CLIPS programmer to include pieces of AppleScript code in the knowledge base itself. This would significantly enhance the power of CLIPS, as it would make it possible to compose, compile, and execute AppleScript programs from within the CLIPS environment, and save these programs as part of a CLIPS knowledge base.

Acknowledgments

Some the extensions described in this paper were designed and implemented by Robert Dominy, formerly of NASA's Goddard Space Flight Facility in Greenbelt, Maryland.

Also, Jeff Shapiro, formerly of Ames, ported many of the enhancements described herein to CLIPS version 6.0.

References

[1] Compton, M., Wolfe, S. 1993 *Intelligent Validation and Routing of Electronic Forms in a Distributed Workflow Environment.*. Proceedings of the Tenth IEEE Conference on AI and Applications.

[2] Compton, M., Wolfe, S. 1994 *AI and Workflow Automation: The Prototype Electronic Purchase Request System.* Proceedings of the Third Conference on CLIPS.

[3] Frainier, R., Groleau, N., Hazelton, L., Colombano, S., Compton, M., Statler, I., Szolovits, P., and Young, L., 1994 *PI-in-a-Box, A knowledge-based system for space science experimentation*, AI Magazine, Volume 15, No. 1, Spring 1994, pp. 39-51.

S19-82
34079
P-7

N95-19644

TARGET'S ROLE IN KNOWLEDGE ACQUISITION, ENGINEERING, VALIDATION, AND DOCUMENTATION

Keith R. Levi, Ph.D.
Department of Computer Science
Maharishi International University
1000 North Fourth Street, FB1044
Fairfield, Iowa 52557
email: levi@miu.edu

ABSTRACT

We investigated the use of the TARGET task analysis tool for use in the development of rule-based expert systems. We found TARGET to be very helpful in the knowledge acquisition process. It enabled us to perform knowledge acquisition with one knowledge engineer rather than two. In addition, it improved communication between the domain expert and knowledge engineer. We also found it to be useful for both the rule development and refinement phases of the knowledge engineering process. Using the network in these phases required us to develop guidelines that enabled us to easily translate the network into production rules. A significant requirement for TARGET remaining useful throughout the knowledge engineering process was the need to carefully maintain consistency between the network and the rule representations. Maintaining consistency not only benefited the knowledge engineering process, but also had significant payoffs in the areas of validation of the expert system and documentation of the knowledge in the system.

INTRODUCTION

Developing an expert system involves processes of knowledge acquisition, creation of a prototype rule base, creating a series of refinements to the prototype system, validating the system, and maintaining the system. Although these processes are conceptually separable, in actual practice they interact a great deal. The knowledge acquisition phase of this development process has long been recognized as the greatest challenge to building an expert system (Hayes-Roth, Waterman and Lenat, 1983).

The TARGET (NASA, 1993) task analysis tool appears to be well-suited to assisting in the knowledge acquisition process in a way that should directly facilitate all the other phases of expert system creation. TARGET (Task Analysis Report Generation Tool) is a graphical network creation tool intended for modeling tasks that are primarily procedural in nature. An example of a TARGET generated network is shown in Figure 1.

TARGET has mainly been used for knowledge acquisition, both as the initial phase of expert system development and also as a self-contained model of a given procedure. It was our expectation that in addition to facilitating knowledge acquisition for an expert system, TARGET would benefit the other processes of expert system development. First, it should benefit the initial creation of rules by acquiring knowledge in a form that is easily translatable into rules. Second, having a graphical representation of knowledge should assist in examining the rule base for areas needing further development and refinement. Third, the graphical representation should also facilitate validation by providing an easily navigable map of the entire knowledge base. And finally, it should aid system maintenance by providing documentation of the knowledge in the expert system in a manner complementary to that provided by the rule base.

APPLICATION DOMAIN

The application domain was a help desk for a commercial software product. Expert systems are becoming quite successful in this domain (Rewari, 1993). Perhaps the most prominent example is Compaq Computer, who have developed and fielded systems that recently received on of the American Association of Artificial Intelligence's awards for innovative applications of AI (Hedberg, 1993). Compaq estimates that their system saves them $10 - $20 million dollars annually in customer service costs. Hewlett-Packard, Digital Equipment Corporation, and Color Tile are other examples of companies that have developed help-desk expert systems (Arnold, 1993). Color Tile's system has reportedly enabled them to expand the services of their help desk, reduce average call time from ten to two minutes, and simultaneously reduce staff from twelve to seven people.

The software product for our expert system, ClearAccess (ClearAccess Corporation, 1993), provides a common end-user interface to over 60 databases, including Oracle, Sybase, Ingres, DB2, IMS, Paradox, DBASE, and others. The system works in client-server architectures, allows database queries to be constructed using a graphical interface, and also interfaces with spreadsheet, wordprocessor, and graphics programs. It can be used in conjunction with a spreadsheet program such as Microsoft Excel to create a spreadsheet front-end to a database.

Technical support for ClearAccess is a signficant challenge for its help desk. In typical software companies, technical support personnel are often junior programmers or even non-programmers. There is high turnover since personnel are often promoted to development positions when they become more knowledgeable about the product. Technical support for ClearAccess is particularly challenging because of the breadth of knowledge required. Technical support personnel must be knowledgeable not only about their own product, but also about the many databases, spreadsheets, and other packages their system interacts with. Since ClearAccess usually works in a client-server network they must also be familiar with client-server and networking software and hardware.

The help-desk expert system we developed for ClearAccess presently encompasses about 200 rules and covers the most critical and common issues encountered by the technical support personnel. The expert system is a rule-based data-driven reasoning system for diagnosis, testing, and remediation of customer problems. The system is presently undergoing validation testing by the help-desk personnel.

KNOWLEDGE ACQUISITION

The first stage in developing an expert system is the process of knowledge acquisition. This process traditionally involves two knowledge engineers interviewing a subject matter expert. One knowledge engineer has lead responsibility for conducting the interview. The other has lead responsibility for taking notes during the interview. In contrast to the traditional knowledge acquisition process we found that only a single knowledge engineer was needed when using TARGET.

During a knowledge acquisition session the domain expert articulated tasks and procedures used by ClearAccess help-desk personnel for a given problem. As each task was articulated the knowledge engineer immediatly recorded these as nodes and links in a TARGET task network. The domain expert observed the network as the knowledge engineer entered it. The knowledge that was being communicated (or not communicated, in some cases) was immediatly obvious. In addition to facilitating knowledge acquisition by making it immediatly obvious what was being communicated, this procedure also provided an explicit visual trace of the knowledge in the system as the session proceeded. This was valuable because it allowed the domain expert to

easily recall any earlier assumptions that she or he made in the course of a long line of reasoning.

Figure 1 shows a fragment of the TARGET task network for problems associated with linking databases with spreadsheets such as Microsoft's Excel. One traces a problem solving session starting with the leftmost node and the following a path according to the comments in the nodes and the labels on the directed links between nodes. The hexagon nodes are decision nodes. There are always two or more arcs leaving a decision node. The text in the decision node poses some question. The arcs leaving the node are labeled with the possible answers to the question. The rectangular nodes are required tasks. They describe a task or procedure that must be performed. Required tasks with heavy borders are terminator tasks. They indicate the completion of a process.

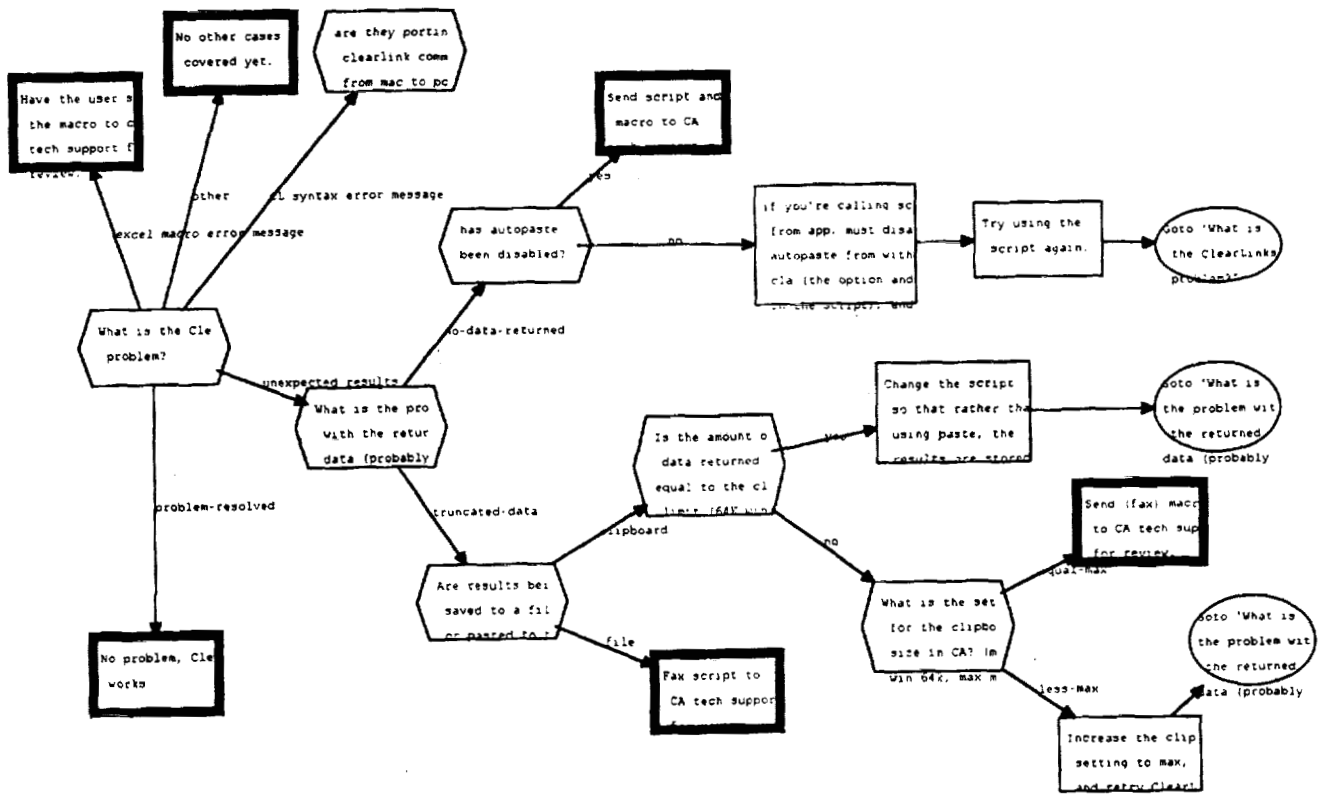


Figure 1. A fragment of the TARGET task network for the ClearAccess technical support expert system.

The network fragment shown in Figure 1 is only a portion of the network for dealing with problems involving ClearAccess and Excel. The total network for dealing with these problems presently has 46 nodes. The fragment pictured has 21 nodes. There are four other problem areas that the expert system currently covers. These areas have their own task networks, each containing from 40 to 50 nodes.

KNOWLEDGE ENGINEERING

The knowledge engineering process involves taking the information gained from the knowledge acquisition process and implementing it in a computer executable form. In this case we are implementing the information as forward-chaining production rules for the CLIPS expert system shell. Typically, the knowledge acquisition process produces textual notes which may or may not be accompanied by graphs, charts, or pseudo-code rules. In the present case, the knowledge acquisition process produces the TARGET task network. We found that translating the task networks into CLIPS rules was straightforward once we established guidelines for creating the TARGET representation.

Rule Creation Phase

Figure 2 shows two rules developed from the partial network shown in Figure 1. The first rule, CLask-clipboard-limit, comes from a hexagonal decision node. The second rule, CLsave-results-to-file comes from the rectangular required-task node that immediatly follows the decision node of the first rule. As illustrated in the first rule, decision nodes become queries when translated to CLIPS rules. The links leaving decision nodes become a menu of possible responses to the query. In this rule, the menu presented to the end user will simply have alternatives of "yes" or "no". This yes-no menu is produced by the *yes* function that appears in the assertion of the rule. The rest of the consequent has the effect of creating an attribute, data-returned-equals-clipboard-limit, and then assigning the appropriate "yes" or "no" value (selected by the end user) to the attribute. This attribute-value pair is then asserted to the fact list. The antecedents of the rule are derived from all the assertions along the path leading to the decision node.

```
(defrule CLask-clipboard-limit
  (phase clinks)
  (subproblem unexpected-results)
  (unexpected-result-type truncated-data)
  (saving-results-file-or-clipbd clipboard)
  =>
  (assert (data-returned-equals-clipboard-limit
    =(yes "Is the amount of data being
          returned equal to the clipboard limit
          (64K: Windows, 32K: Mac)?"))))
```

```
(defrule CLsave-results-to-file
  (phase clinks)
  (subproblem unexpected-results)
  (unexpected-result-type truncated data)
  (data-returned-equals-clipboard-limit yes)
  (saving-results-file-or-clipbd clipboard)
  =>
  (msg "Change the script so that rather than
        using paste, the results are stored to a
        file; then open the file in the application.")
  (assert (saving-results-file-or-clipbd file)))
```

Figure 2. Two CLIPS rules derived from task network displayed in Figure 1.

It is important that all the nodes along the path to the decision node are involved in the antecedent of the rule. This serves to make the rule a self-contained piece of knowledge. This self-containment property is fundamental to having a comprehensible and well-structured rule-based expert system (Brownston, et. al., 1985; Ignizio, 1991; Pedersen, 1989).

The second rule in Figure 2 illustrates that required-task nodes will typically produce some notification message telling the user to perform some action. (Or, in a more automated environment the expert system might perform the action itself). The consequent of the rule must assert a fact onto the fact list to represent that the indicated action has been performed. In this rule a message is given to the user to change the spreadsheet macro-script such that the results are stored to a file rather than using the paste function. A corresponding fact is then asserted to

the fact list. Analogous to the case for decision nodes, the antecedents for this rule are derived from all the assertions along the path leading to the required-task node.

Rule Refinement Phase

A greater challenge than creating a set of rules from the initial TARGET network was keeping the TARGET representation consistent with the CLIPS rules as we refined the system. Such refinements are a major part of the expert system development process. The usual case in developing an expert system is to quickly develop an initial prototype system consisting of a first-pass set of rules. Most of the work in developing the system is then performed as a process of trying out the prototype, adding new rules, and modifying rules as necessary.

The difficulty in maintaining consistency arises because the expert system's behavior results from the knowledge as implemented in CLIPS rules and executed by the CLIPS inference engine. The knowledge underlying the system, however, is most easily analyzed in terms of the graphical TARGET representation. We found that it was quite convenient and informative to use the TARGET network as a guide to which parts of the expert system were incomplete or needed further refinements.

For this knowledge engineering phase we used two PCs operating side-by-side. One was displaying the TARGET network and the other was executing the CLIPS expert system. We used the TARGET network as a navigation guide for testing the behavior of the expert system. We made only minor changes to the CLIPS rules during these sessions. We found it was much more efficient to make changes to the TARGET network during the session with the domain expert rather than to the CLIPS rules. We added new nodes to the network in some cases and in others modified existing nodes. We clearly marked any changes to the network so that we could return to them at a later time and make the necessary modifications to old rules or additions of new rules to the CLIPS rule set.

This procedure was heavily dependent on maintaining a close correspondence between the network and rule representations. If there was not a close correspondence then we could not easily locate the network nodes to modify in response to desired changes in the behavior of the expert system. Or, we might make changes to network nodes, but if there is a lack of correspondence then it becomes quite difficult to locate the CLIPS rules that need to be changed in response to the network changes. We found that unless we were very careful and conscientious about maintaining this consistency the two representations quickly diverged and it became impossible to relate them.

VALIDATION AND DOCUMENTATION

Maintaining consistency between the network and rule representations was a lot of work. It required a significant amount of discipline and effort on our part. This was not surprising since the task network represents a type of external documentation for the expert system code. Most software developers tend to avoid careful documentation of their code unless they are required to do so. Such writing takes time and effort, and developers almost always feel that their code is clear and comprehensible--at least at the time they are writing it. This natural tendency to avoid documentation is even stronger in developing expert systems where one uses a fast prototyping approach and code is typically written, tested, and then revised or discarded. It is almost always the case, however, that the code is never so clear and comprehensible to other individuals who might have to maintain the code at a later date. Further, it is common experience that the code becomes difficult to understand even for the original developer after a few weeks or ever a few days away from it.

Thus, one significant benefit of maintaining consistency of the network and rule representations beyond its immediate benefit of aiding the rule refinement process is that it results in effective documentation of the knowledge in the expert system. Indeed, the two representations document the knowledge in the system in complementary ways. Rule-based representations are well-suited to documenting self-contained chunks of declarative knowledge (Brownston et. al., 1985; Ignizio, 1991; Pedersen, 1989). But they are not so appropriate for documenting the procedural flow-of-control among the chunks of knowledge. This information is implicit in the interaction between the entire rule set and the expert system inference engine. The task network representation, on the other hand, is well-suited to documenting the flow of control of the knowledge in the system. It is not so strong at representing the self-contained chunks of knowledge that are well-captured by production rules.

This documentation of corporate knowledge is by itself of significant value to a company. It is also quite useful for validating the knowledge in the implemented version of the system. Given a task network representation that is closely related to the rule-based implementation of the same knowledge, it is quite straightforward to use the network to guide a domain expert in testing the system such that he or she evaluates at least one path to every final state in the system. Referring to the network fragment in Figure 1, one can see that it would be straightforward to use the network to direct answers to the expert system queries such that the network is navigated to each terminal node. After a terminal node is encountered the knowledge engineer (or domain expert) marks that node and its incoming link as having already been traversed. In this way it is easy to ensure that every terminal node has been visited at least once.

Unless the task network has a simple tree topology, this procedure does not guarantee that all possible paths through the expert system have been examined. This would require a more sophisticated graph traversal procedure. However, even this simple procedure should guarantee a much more systematic and thorough validation check of an expert system than is often done for rule-based expert systems.

SUMMARY AND CONCLUSION

First, we found TARGET to be very helpful in the knowledge acquisition process. It enabled us to perform knowledge acquisition with one knowledge engineer rather than two, and in addition it improved communication between the domain expert and knowledge engineer. We also found it to be useful for both the rule development and refinement phases of the knowledge engineering process. Its usefulness in this process required that we adopt guidelines for developing the network. These guidelines enabled us to easily translate the network into production rules. A significant requirement for TARGET remaining useful throughout the knowledge engineering process was the need to carefully maintain consistency between the network and the rule representations. Maintaining such a consistency not only benefited the knowledge engineering process, but also had significant payoffs in the areas of validation of the expert system and documentation of the knowledge in the system. We conclude that TARGET is a very useful tool for aiding the development of rule-based expert systems such as are typically developed with the CLIPS expert system shell.

Finally, we propose that a promising area for future work is to automate the process of generating CLIPS rules from a TARGET task network. Such a facility would simultaneously eliminate the need to hand-code CLIPS rules and the problem of maintaining consistency between the task network and the production rules. The generality of such a rule generator will depend on the specific guidelines and restrictions that are placed on the construction of the task networks.

ACKNOWLEDGMENTS

Ahto Jarve provided very able assistance in performing much of the knowledge acquisition and knowledge engineering for the expert system described in the paper. Melinda Thomas and Patrick Geurin of Fairfield Software provided the domain expertise encoded in the expert system. I would also like to acknowledge the initial work on the Clear Access technical support expert system carried out by Dr. Ralph Bunker, Patrick Daley, and Slobodan Dumuzliski.

REFERENCES

Arnold, B., "Expert System Tools Optimizing Help Desks," SOFTWARE MAGAZINE, January, 1993, pp. 56 - 64.

Brownston, L., Farrell, R., Kant, E., and Martin, N., PROGRAMMING EXPERT SYSTEMS IN OPS5, Addison-Wesley, Reading, MA, 1985.

ClearAccess, "User's Guide," ClearAccess Corporation, Fairfield, Iowa, 1994.

Hayes-Roth, F. Waterman, D.A., and Lenat, D., BUILDING EXPERT SYSTEMS, Addison-Wesley, Reading, MA, 1983.

Hedberg, S. "The Evolution of Applied AI: A Report on the Fifth Innovative Applications of AI Conference," AI MAGAZINE, Fall, 1993.

Ignizio, J., INTRODUCTION TO EXPERT SYSTEMS: THE DEVELOPMENT AND IMPLEMENTATION OF RULE-BASED EXPERT SYSTEMS, McGraw-Hill, 1991.

Ortiz, C.J., Ly, H.V., Saito, T., and Loftin, R.B. "TARGET: Rapid Capture of Process Knowledge," PROCEEDINGS OF TECHNOLOGY 2002: THE THIRD NATIONAL TECHNOLOGY TRANSFER CONFERENCE AND EXPOSITION, Vol 1, NASA, Washington, February 1993, pp 279-288.

Pedersen, K., EXPERT SYSTEMS PROGRAMMING: PRACTICAL TECHNIQUES FOR RULE-BASED SYSTEMS, Wiley, 1989.

NASA, "TARGET Reference Manual," Software Technology Branch, NASA, Johnson Space Center, Houston, Texas, November, 1993.

Rewari, A., "AI in Corporate Service and Support," IEEE EXPERT, Vol 8:6, 1993, pp 5 - 40.

OMIT

Session 4A: Aerospace Applications

Session Chair: Melissa Mahoney

An Expert System for Configuring a Network for a Milstar Terminal

Melissa J. Mahoney
Dr. Elizabeth J. Wilson

Raytheon Company
1001 Boston Post Road
Marlboro, MA 01572

melissa@tif352.ed.ray.com
bwilson@sud2.ed.ray.com

Abstract

This paper describes a rule-based expert system which assists the user in configuring a network for Air Force terminals using the Milstar satellite system. The network configuration expert system approach uses CLIPS. The complexity of network configuration is discussed, and the methods used to model it are described.

1. Introduction

Milstar COMPLEXITY

Milstar is a flexible, robust satellite communications system which provides world-wide communication capability to a variety of users for a wide range of tactical and strategic applications. Milstar is interoperable requiring simultaneous access by Air Force, Navy and Army terminals. The transmit/receive capability of these terminals ranges from small airborne units with limited transmission power to stationary sites with large fixed antennas.

The wide range of applications of the Milstar system includes services capable of supporting status reporting, critical mission commands, point-to-point calls, and networks. The same system that provides routine weather reports must also allow top-level mission coordination among all services. This diversity results in traffic which is unpredictable and variable. The system must accommodate both voice and data services which can be transferred at different rates simultaneously. In order to protect itself from jamming and interference, the Milstar waveform includes a number of signal processing features, such as frequency hopping, symbol hop redundancy, and interleaving, some of which are among the network parameters defined at service setup.

The configuration of each terminal is accomplished through the loading of adaptation data. This complex data set provides all the terminal specific and mission specific parameters to allow the terminal to access the satellite under the mission conditions with the necessary resources to establish and maintain communication. Network definitions are a subset of this adaptation data.

Defining a network for a terminal is a challenging task requiring a large body of knowledge, much of it heuristic. The systems engineers who have been on the Milstar program for many years have amassed this knowledge; many of them have been reassigned to other programs. Since the recent launch of the first Milstar satellite, testing activity has been very high and expertise in this area is not readily available. When testing ends and Milstar becomes fully operational, the terminals will be operated by personnel whose Milstar knowledge is considerably less than that of the systems engineers. To solve the problem of having a user with limited knowledge take advantage of the many network configuration capabilities of the Milstar system, one of the systems engineers proposed developing an expert system to encapsulate the vast amount of knowledge required to successfully design network definitions accurately and reliably.

NETWORK COMPLEXITY

A Milstar network consists of 82 parameters which are interrelated. Some relationships are well-understood, but many are obscure. The few existing domain experts spend a significant amount of time constructing network definitions starting from a high-level mission description usually provided by a commanding authority. Not only should the network allow communication, but it should do so in a manner which makes optimal use of satellite resources. The expert, then, is faced with an optimization problem: to define a network whose parameters are consistent with each other while minimizing the use of payload resources.

The complexity of network configuration is compounded by the number and locations of the terminals which will participate in the network. Additional terminals add more constraints to the problem space. The resource requirements of different terminals may vary, and their locations may be dynamic, as in the case of airborne terminals.

The complexity further increases when configuring a terminal to participate in several networks simultaneously. Certain resources cannot be shared among a terminal's active networks.

KNOWLEDGE ACQUISITION

Several system engineers were available as knowledge sources. Initially, they were asked to review each of the network parameters and to describe obvious relationships between them. Remarks about rules, exceptions, and values were recorded and coded into prototype rules. These rules were reviewed with the experts and modified as needed to confirm that the information had been accurately translated.

In addition to interviewing the domain experts, scores of documents were read. Tables and charts were constructed to focus on the relationships between only several parameters at a time. Gradually, a hierarchy of knowledge was formed.

The knowledge base was partitioned into functional areas (e.g., hardware parameters, mission-level parameters, terminal-specific parameters, and service-specific parameters). A knowledge engineer was assigned to each one. When it was deemed that most of the knowledge had been acquired in one of these areas, the entire knowledge engineering team of four would check the functional area's rules with respect to each network type (of which there are 11). This cross-checking resulted in more rules which captured the exceptions and special cases associated with the more general rules belonging to the functional area.

In-house operations and maintenance training courses provided the knowledge engineers with an opportunity to learn Milstar as an Air Force terminal operator. This provided the hands-on training that taught lessons not documented in specifications.

2. The Expert System

SCOPE

The scope of the task was to develop an expert system to assist the user in configuring a network definition for a single terminal. This scope represents the necessary first step in later building an extended system which will address the issues of multiple terminals and multiple networks.

SYSTEM COMPONENTS

The expert system consists of three major components: a graphical user interface (GUI), a rule and fact base, and an interface between those two components. The GUI performed the more trivial consistency checks between the input parameters. For example, if a voice network is being configured, then the only valid data rate for the network is 2400 bps. The GUI ghosts out all other choices. The GUI also checks the syntax and ranges of user-entered data.

The interface code controls the flow of the expert system. It reads in the user-supplied input items, translates them into facts, loads the appropriate rules; when the rules have finished firing, the interface code passes their results to the GUI. This process continues as the user moves between the screens.

Rules and facts were used to check the more complicated relationships between the parameters. Rules were not used to perform trivial tasks (e.g., syntax checking) nor inappropriate tasks (e.g., controlling the flow).

SOFTWARE TOOLS

CLIPS was chosen for this project because of its forward-chaining inference engine, and its ability to run as both a standalone and an embedded application. Other benefits were its object-oriented design techniques and its level of customer support.

The rules and facts were developed, tested and refined on a Sun workstation under UNIX. They were ported to an IBM 486/66, the target environment. A point-and-click graphical user interface was developed for the PC using Visual Basic. Borland C++ was used to write the interface code between the GUI and the CLIPS modules.

Currently, the expert system contains about 100 rules and 1000 facts.

OBJECTS

CLIPS Object-Oriented Language (COOL) was used where appropriate.

In the problem domain, terminals are represented as objects having attributes such as a terminal id, an antenna size, a latitude/longitude location, and a port usage table. The port usage table is made up of 34 ports. A port is an object which has attributes such as data rate and i/o hardware type. Terminals are objects to prepare for future expansion to the analysis of a network definition for multiple terminals.

CLIPS offers a query system to match instances and perform actions on a set of instances. For example, CLIPS can quickly find the terminal requiring the most robust downlink modulation by keying off each terminal's antenna size and satellite downlink antenna type. Satisfying the needs of this terminal would be a constraint to ensure reliable communication among a population of terminals.

TEMPLATES

Valid parameter relationships are defined using templates and asserted as facts during system initialization. For example, only a subset of uplink time slots are valid with each combination of data rate, uplink modulation, and uplink mode. Figure 1a shows a table relating these parameters. Figure 1b shows a template modeled after this table. Figure 1c shows a deffacts structure which will assert the facts contained in the table by using the template.

RULES

Several rules exist for most of the 82 parameters. The left-hand side of each rule specifies a certain combination of patterns which contain some bounded values (i.e., network parameters that already have values assigned). The right-hand side of each rule performs three actions when it fires. First, it assigns a suggested parameter value. Second, it creates a multifield containing other valid, but less optimal, parameter values. And, third, it creates an explanation associated with the parameter. The GUI highlights the suggested value, but allows the user to change it to one of the other valid choices. Figure 2 shows a rule which fires when the expert system does not find any valid uplink time slots. [* The problem of refiring needed to be addressed. Refiring occurred because when a rule modifies a fact, the fact is implicitly retracted and then reasserted. Since the fact is "new", the rule fires again immediately. This problem was solved by including a check in the antecedent whether the valid choices multifield is currently the same as what the rule will set it to be. If the fields are identical, then the rule will not refire.]

EXPLANATIONS

The explanation associated with each parameter reflects the left-hand side of the rule which ultimately assigned the parameter's value. An explanation template contains the text, and a flag to indicate whether a positive rule fired (i.e., a valid parameter value was suggested), or a negative rule fired (i.e., the expert system could find no valid value for the parameter based upon preconditions). In the latter case, the GUI immediately notifies the user that the expert system is unable to proceed, and it states a recommendation to fix the problem. Normally the user is instructed to back up to a certain parameter whose current value is imposing the unsatisfiable constraint. The explanation facility is invoked by pointing the mouse on the parameter and clicking the right button.

OUTPUT

After the user has proceeded through the screens and values have been assigned to all network parameters, s/he may choose various output media and formats. In production, the completed network will be converted to hexadecimal format and downloaded into the terminal's database which is then distributed to the field.

FUTURE EXTENSIONS

As stated earlier, a terminal class was established to allow for future expansion to the analysis of a network with respect to multiple terminals. Taking this idea one step further would involve creating a network class, and making a network definition an instance of it. Multiple networks could then be considered simultaneously when writing rules to suggest values for certain parameters. For example, a terminal cannot participate in two networks which use the same uplink time slot. Advancing the expert system to this level of consistency checking would improve its utility significantly. CLIPS promotes a modular approach to design which allows the developer to plan for future extensions like these.

3. Conclusions

An expert system has proven to be a valuable tool in the configuration of Milstar terminal network parameter sets. The generation of these parameter sets is complex and primarily heuristic lending itself to an expert system approach. CLIPS and COOL has provided an effective and efficient development environment to capture the knowledge associated with the application domain and translate these relationships into a modular set of rules and facts. The decision to use rules vs. facts, CLIPS vs. C++, rules vs. GUI was made after careful consideration of the overall implementation strategy and the most efficient means to provide the expertise.

The complexity of the problem domain, variety of sources of expertise (design engineers, operational support, and program documentation), and broad and varying scope of the applications has made this project a significant knowledge acquisition challenge. The knowledge engineering team approach has been successful in translating the vast and subtle nuances of the implementation strategies and in developing a synergetic composite of the available expertise.

The combined use of rules, facts, objects, C++ code, and a graphical interface has provided a rich platform to capture the Milstar knowledge and transform this knowledge into a modular tool staged for expansion and improvement.

References

1. Basel, J., D'Atri, J. and Reed, R., "Air Force Agile Beam Management", Raytheon Company, Equipment Division, Marlborough, Massachusetts, June 15, 1989.

2. Giarratano, J. and Riley, G., EXPERT SYSTEMS: PRINCIPLES AND PROGRAMMING, ed. 2, PWS Publishing Company, Boston, Massachusetts, 1994.

3. Vachula, G., "Milstar System Data Management Concept", GMV:92:05, Raytheon Company, Equipment Division, Communication Systems Laboratory, Marlborough, Massachusetts, February 7, 1992.

Primary U/L Channels

U/L Modulation		Data Rate (bps)					
		2400	1200	600	300	150	75
HHR FSK	2	1,2,3,4	--	--	--	--	--
	4	5,6	1,2,3,4	--	--	--	--
	8	7	5,6	1,2,3,4	--	--	--
	16	--	7	5,6	1,2,3,4	--	--
	32	--	--	7	5,6	1,2,3,4	--
	64	--	--	--	7	5,6	1,2,3,4
	128	--	--	--	--	7	5,6
	256	--	--	--	--	--	7
LHR FSK	10	--	--	--	7	5,6	1,2,3,4
	20	--	--	--	--	7	5,6
	40	--	--	--	--	--	7

Secondary U/L Channels

U/L Modulation		Data Rate (bps)		
		300	150	75
HHR FSK	2	1,2,3,4	--	--
	4	5,6	1,2,3,4	--
	8	7	5,6	1,2,3,4
	16	--	7	5,6
	32	--	--	7

Figure 1a: Table Relating Uplink Time Slot, Uplink Mode, Uplink Modulation and Data Rate

```
; valid-dr-ts

; this template sets up associations of uplink mode,

; uplink modulation, and data rate/timeslot pairs

(deftemplate valid-dr-ts)
        (field ul-mode
                (type SYMBOL)
                (default ?NONE))
        (field ul-modulation
                (type SYMBOL)
                (default ?NONE))
        (multifield dr-ts
                (type ?VARIABLE)
                (default ?NONE))
```

Figure 1b: Template Corresponding to Table in Figure 1a

```
; this deffacts defines valid data rate and timeslot pairs

; for each combination of uplink mode and uplink modulation

(deffacts set-valid-dr-ts
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK2)
                        (dr-ts 2400 bps with time slots 1-4))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK4)
                        (dr-ts 2400 bps with time slots 5-6
                        1200 bps with time slots 1-4))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK8)
                        (dr-ts 2400 bps with time slot 7
                        1200 bps with time slots 5-6))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK16)
                        (dr-ts 1200 bps with time slot 7
                        600 bps with time slots 5-6
                        150 bps with time slots 1-4))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK32)
                        (dr-ts 600 bps with time slot 7
                        300 bps with time slots 5-6
                        150 bps with time slots 1-4))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK64)
                        (dr-ts 300 bps with time slot 7
                        150 bps with time slots 5-6
                         75 bps with time slots 1-4))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK128)
                        (dr-ts 150 bps with time slot 7
                        75 bps with time slots 5-6))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation HHRFSK256)
                        (dr-ts 75 bps with time slot 7))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation LHRFSK10)
                        (dr-ts 300 bps with time slot 7
                        150 bps with time slots 5-6
                         75 bps with time slots 1-4))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation LHRFSK20)
                        (dr-ts 150 bps with time slot 7
                        75 bps with time slots 5-6))
        (valid-dr-ts    (u1-mode PRIMARY)
                        (u1-modulation LHRFSK40)
                        (dr-ts 75 bps with time slot 7))
        (valid-dr-ts    (u1-mode SECONDARY)
                        (u1-modulation HHRFSK2)
                        (dr-ts 300 bps with time slots 1-4))
```

Figure 1c: Deffacts Representation of Table in Figure 1a

```
(valid-dr-ts      (ul-mode SECONDARY)
                  (ul-modulation HHRFSK4)
                  (dr-ts 300 bps with time slots 5-6
(valid-dr-ts      (ul-mode SECONDARY)
                  (ul-modulation HHRFSK8)
                  (dr-ts 300 bps with time slot 7
                  150 bps with time slots 5-6
                   75 bps with time slots 1-4))
(valid-dr-ts      (ul-mode SECONDARY)
                  (ul-modulation HHRFSK16)
                  (dr-ts 150 bps with time slot 7
                   75 bps with time slots 5-6))
(valid-dr-ts      (ul-mode SECONDARY)
                  (ul-modulation HHRFSK32)
                  (dr-ts 75 bps with time slot 7))
```

```
(defrule 12bults-009
"   this rule fires when the terminal's required u/1 modulation is more robust than the implied u/1 modulation
    using any u/1 time slot."
      (outnet  (name data-rate)
               (value ?dr-value))          ;   get value of data-rate
      (outnet  (name u1-mode)
               (value ?u1-mode))           ;   get value of u1-mode
      (outnet  (name u1-antenna-type)
               (value ?u1-ant-type))       ;   get value of u1-ant-type
      (outnet  (name net-type)
               (value ?net-type))          ;   get value of net-type
      ?fact <- (outnet   (name u1-time-slot)   ;   get address
                         (choices $?oldchoices))   ;   of time slot
      (test    (neq $?oldchoices =(mv-append nil)))   ;   don't refire (* see note)
      (terminal ?term-id)                  ;   get the terminal id
      (u1modmap        (ant-size =(get-ant-size  ?term-id)
                       (u1-ant ?ul-ant-type)
                       (u1-modulation ?ul-mod-terminal))
                                                    ;   get the minimum
                                                    ;   u1 mod this terminal
                                                    ;   can use given its antenna size
                                                    ;   and the u1 antenna type
      (outnet  (name u1-modulation)
               (value ?u1-mod-type))               ;   get the u/1 modulation type
      (valid-u1-quad   (data-rate ?dr-value)       ;   get all valid modulations
                       (u1-mode ?u1-mode)          ;   for this data rate
                       (u1-mod-type ?u1-mod-type)
                       (u1-mods $?all-valid-mods))
      (most-to-least u1 ?u1-mode $?allmods)        ;   get all modulations
      (test      (more-robustp ?u1-mod-terminal   (first $?all-valid-mods)    $?allmods))
                                                    ;   if the terminal's
                                                    ;   required robustness is
                                                    ;   more robust than the
                                                    ;   most robust implied
                                                    ;   modulation
      (valid-dr-ts       (u1-mode ?u1-mode)
                         (u1-modulation ?u1-mod-terminal)
                         (dr-ts $?all-dr-ts))      ;   get all pairs of valid data rates
                                                    ;   and timeslots for this modulation
      ?explain <- (explanation (param-name ul-time-slot)   ;   get previous explanation
                               (text ?oldtext))    ;   associated with this
                                                    ;   parameter
```

Figure 2: Rule Using the Facts Shown in Figure 1c

```
=>

(modify ?fact     (suggest nil)                    ;   no suggested value
                  (choices =(mv-append nil)))      ;   no valid choices
(if (neq ?net-type MCE)                            ;   tailor the explanation
then                                               ;   to the network type
(bind ?newtext
(str-implode
(mv-append
(str-explode "This terminal cannot reliably support this service's data rate u sing the current u/1 antenna
type. U sing any time slot with this data rate would imply an insufficient u/1 modulation for this
terminal. Either use a stronger u /1 antenna type, or lower the data rate. To satisfy this terminal's u/1
modulation requirements, the valid data rate/time slot pairs are:"
$?all-dr-ts
)         ;end append
)         ;end implode
)         ;end bind

else
(bind ?newtext
(str-implode
(mv-append
(str-explode "This terminal cannot reliably support this service's data rate u sing the current u/1 antenna
type. U sing any time slot with this data rate would imply an insufficient u/1 modulation for this
terminal. Either use a stronger u /1 antenna type, or lower the data rate. To satisfy this terminal's u/1
modulation requirements, the valid data rate/time slot pairs are:"
$?all-dr-ts
(str-explode". Since this is an MCE network, time slot 4 should be used.")
)         ;end append
)         ;end implode
)         ;end bind
)         ;end if

(explain ?explain  ?old text  ?newtext YES)

)
```

Figure 2: Rule Using the Facts Shown in Figure 1c (cont'd.)

S21-82
34681
P-11

N95-19646

Expert System Technologies for Space Shuttle Decision Support: Two Case Studies

Christopher J. Ortiz
Workstations Branch (PT3)
Johnson Space Center
Houston, TX 77058
chris.ortiz@jsc.nasa.gov

David A. Hasan
LinCom Corporation
1020 Bay Area Blvd., Suite 200
Houston, TX 77058
hasan@gothamcity.jsc.nasa.gov

Abstract

This paper addresses the issue of integrating the C Language Integrated Production System (CLIPS) into distributed data acquisition environments. In particular, it presents preliminary results of some ongoing software development projects aimed at exploiting CLIPS technology in the new Mission Control Center (MCC) being built at NASA Johnson Space Center. One interesting aspect of the control center is its distributed architecture; it consists of networked workstations which acquire and share data through the NASA/JSC-developed Information Sharing Protocol (ISP). This paper outlines some approaches taken to integrate CLIPS and ISP in order to permit the development of intelligent data analysis applications which can be used in the MCC.

Three approaches to CLIPS/ISP integration are discussed. The initial approach involves clearly separating CLIPS from ISP using *user-defined functions* for gathering and sending data to and from a local storage buffer. Memory and performance drawbacks of this design are summarized. The second approach involves taking full advantage of CLIPS and the CLIPS Object-Oriented Language (COOL) by using *objects* to directly transmit data and state changes from ISP to COOL. Any changes within the object slots eliminate the need for both a data structure and external function call thus taking advantage of the object matching capabilities within CLIPS 6.0. The final approach is to treat CLIPS and ISP as *peer toolkits*. Neither is embedded in the other, rather the application interweaves calls to each directly in the application source code.

Introduction

A new control center is being built at the NASA Johnson Space Center. The consolidated Mission Control Center (MCC) will eventually replace the existing control center which has been the location of manned spaceflight flight control operations since the Gemini program in the 1960s. This paper presents some preliminary results of projects aimed at incorporating knowledge-based

applications into the MCC. In particular, it discusses different approaches being taken to integrate CLIPS with the MCC Information Sharing Protocol (ISP) system service.

MCC and ISP

The new control center architecture differs significantly from the current one. The most prominent difference is its departure from the mainframe-based design of the existing control center. The new MCC architecture is aggressively distributed. It consists of UNIX workstations (primarily DEC Alpha/OSF1 machines at present) connected by a set of networks running TCP/IP protocols. Exploitation of commercially available products which will be relatively easy to replace and upgrade has been a prime motivating factor in this change. Particular attention has been paid to the use of standard hardware, and commercial off-the-shelf (COTS) software is being used wherever possible.

With a mainframe computer at the center of all flight support computation, the process of presenting telemetry and computational results to flight controllers was really a matter of "pushing" the relevant data out of the mainframe onto the appropriate flight control terminals. In recent years, the task of acquiring telemetry on the system of MCC upgrade workstations has been a matter of requesting the telemetry data from the mainframe. The central computer has served as the one true broker for telemetry data and computations.

In the distributed MCC design, the notion of a central telemetry and computation broker has disappeared. In fact, terminals have disappeared. The flight control consoles consist of UNIX workstations which have access to telemetry streams on the network but must share data instead of relying on the mainframe for common computations.

Software applications running on the MCC workstations will obtain telemetry data from a system service called the Information Sharing Protocol (ISP). ISP is a client/server service. Distributed clients may request ISP support from a set of "peer" ISP servers. The servers are responsible for extracting data from the network. Client applications needing those data initiate a session with the servers (possibly from different machines), using the ISP client application program interface (API). These clients *subscribe* to data from the ISP servers, which deliver the data asynchronously to the clients as the data change. Parenthetically, the ISP API provides *data source independence*. Thus, ISP client applications may be driven by test data for validation and verification, playback data for training or live telemetry for flight support.

It is the intent of the new MCC architecture that hardware (workstations, routers, network cabling, etc...) will take a backseat to the "software platform". ISP is a prominent element of this platform, since it is the data sharing component of the system services in addition to providing a telemetry acquisition service. Indeed, the architecture presupposes that many of the functions previously handled in the mainframe program (e.g., display management, limit sensing, fault summary

messages, "comm fault" detection) will now be carved up into smaller, easier to maintain application programs.

ISP supports the integration of these applications by allowing clients to receive data that have been *published* by other clients. These shared data will in many instances be computations which were previously handled internally to the mainframe, e.g., spacecraft trajectories and attitude data, but with greater frequency, the shared data are expected to be "higher order information" derived from analyses of telemetry data.

Some of the data analysis necessary for the generation of this higher order information will be derived from rule-based pattern matching of telemetry. One example of this is the Bus Loss Smart System which is used by EGIL flight controllers to identify power bus failures. Another example discussed later in this paper is the Operational Instrumentation Monitor (oimon) which assesses the health of electronic components involved in the transmission of Shuttle sensor data down to Earth.

Rule-based applications can capture the often heuristic procedures used by flight controllers to perform their duties. A number of such applications are currently under development by flight control groups where the knowledge-based approach contributes to getting the job done "better, faster and cheaper". CLIPS is being used in a number of these. In the discussion that follows, methods for combining the telemetry acquisition and data sharing functionality of ISP with the pattern matching capabilities of CLIPS are discussed.

An Embedded Approach

CLIPS is a data-driven language because it uses data in the form of facts to activate and fire if-then rules which result in a change of execution state for the computer. CLIPS was designed to be embedded in other applications thus allowing them the ability to perform complex tasks without the use of complex programming on the part of the rule developer. Since ISP is a transport protocol for both receiving and transmitting information, it only makes sense to find a way to integrate the two toolkits to provide CLIPS developers with a reliable transport mechanism for data.

In our work using ISP and CLIPS we have taken two approaches in integrating the two toolkits. The first was to directly embed ISP within CLIPS and provide a simple set of user-defined functions which allow the CLIPS developer to communicate via ISP. The second approach was to use both toolkits as peers allowing the developer to have complete control over the integration.

The first method of embedding ISP into CLIPS grew out of three separate attempts to balance data, speed and ease of use concerns for the application developer. The first attempt at integration (Figure 1) consisted of creating a separate *data layer* with which ISP and CLIPS could communicate. This layer provides a storage location for all the change only data that ISP receives as well

as hiding the low level ISP implementation. This approach provided several challenges in how ISP and CLIPS would communicate with the data layer.

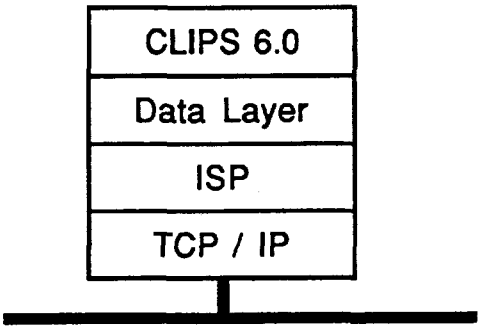


Figure 1: The first attempt at integration

A simple set of commands was added to CLIPS to aid in the communication with ISP.

```
(connect-server [CLIPS_TRUE / CLIPS_FALSE] )
(subscribe-symbol [symbol_name] [CLIPS_TRUE / CLIPS_FALSE] )
(enable-symbols [CLIPS_TRUE / CLIPS_FALSE] )
```

The ***connect-server*** command establishes or disconnects a link with the ISP server and registers the application as requesting ISP data. Likewise, ***subscribe-symbol***, informs the ISP server of which data elements are requested or no longer needed by the application. The ***enable-symbols*** command begins and ends the flow of data to the application.

Other API calls were added to CLIPS to enable an application to publish data to the ISP server for use in other CLIPS or ISP only applications.

```
(publish-symbol [symbol-name] [EXCLUSIVE])
(publish [VALUE/LIMIT/STATUS/Message] [symbol-name] [data]
[time])
(unpublish-symbol [symbol-name] )
```

The ***publish-symbol*** command informs the ISP server that the CLIPS application wants permission to send data under a given symbol-name. The optional EXCLUSIVE parameter informs the server that the requesting application should be the only application allowed to change the value.

The final set of APIs allows ISP to communicate with CLIPS in the form of facts.

```
(want-isp-event [CYCLE/LIMIT] [CLIPS_TRUE/CLIPS_FALSE])
```

The ***want-isp-event*** command will activate or deactivate cycle or limit facts to be published in the fact list.

This first attempt provided a solid foundation for communication between the two tool kits. However, the use of a separate data layer proved to be cumbersome to

implement. The data layer needed to have a dynamic array and a quick table look-up mechanism as well as a set of functions to provide CLIPS with the ability to check values. Each time a rule needed a data value, an external function would have to be called. This proved to be a costly option in the amount of time needed to call the external function. A better method was clearly needed.

Our second attempt at integration ISP and CLIPS consisted of the elimination of the data layer (Figure 2). All of the ISP data would be fed to CLIPS via facts. This method had several advantages. First, it reused the ISP functions developed earlier. Second, it allowed CLIPS to store all ISP data as facts. Finally, CLIPS application developers could do direct pattern matching on the facts to retrieve the data.

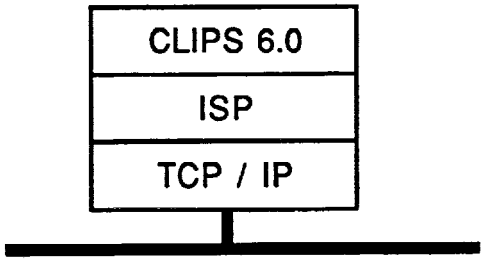


Figure 2: Removal of the data layer

The use of facts provided a few smaller challenges in the update and management of the fact list. There needed to be a way to find an old fact that had not been used and retract it when new data arrived. This could be done at the rule level or in the integration level. At the rule level, the application programmer would be required to create new rules to seek out and remove obsolete facts. At the integration level, time would be spent looking for matching facts with similar IDs to be removed.

The final approach at integration of ISP with CLIPS involved the use of the CLIPS Object-Oriented Language (COOL). Each data packet would be described as an instance of a class called MSID. The MSID class would provide a data storage mechanism for storing the data name, value, status, time tag, and server acceptance information.

```
(defclass MSID
   (is-a USER)
   (role concrete)
   (pattern-match reactive)
   (slot value ( create-accessor read-write) (default 0.0))
   (slot status ( create-accessor read-write)(default 0.0))
   (slot time (create-accessor read-write)(default 0.0))
   (slot accepted (create-accessor read-write)
     (default CLIPS_TRUE))
)
```

A second advantage of using the object implementation consists of inheriting constructors and destructors. When an instance of the MSID class is created, a

constructor is activated and the symbol is automatically subscribed. On the other hand, when an symbol is no longer needed a destructor is activated and the symbol is automatically unsubscribed. Constructors and destructors free the application programmer from worrying about calling the appropriate ISP APIs for creating and deleting symbols.

```
(defmessage-handler MSID init after()
   (subscribe-symbol (instance-name ?self) CLIPS_TRUE)
)

(defmessage-handler MSID delete before()
   (enable-symbols CLIPS_FALSE)
   (subscribe-symbol (instance-name ?self) CLIPS_FALSE)
   (enable-symbols CLIPS_TRUE)
)

( definstances MSIDS
   ( S02K6405Y of MSID )
   ( S02K6205Y of MSID )
   ( S02K6026Y of MSID )
   ...
   ( S02K6078Y of MSID )
)
```

Creating and subscribing symbols are automatically handled by COOL. One of the only ISP implementation details that the application programmer needs to be concerned is to enable the symbols and to schedule which ISP events are to be handled.

```
(defrule connect
   ?fact<- (initial-fact)
=>
   (retract ?fact)
   (enable-symbols CLIPS_TRUE )
   (want-isp-event LIMIT CLIPS_FALSE )
   (want-isp-event CYCLE CLIPS_TRUE )
)
```

Another clear advantage of using objects, like facts, is the ability to do direct pattern matching. As the ISP data changes, a low level routine updates the value in the affected slot. CLIPS could then activate any rule which needed data from the changed slot and work with this information on.

```
(defrule ValueChanges
   ?msid <- (object ( is-a MSID ) ( value ?value))
=>
   (update-interface ( instance-name-to-symbol (
      instance-name ?msid)) ?value )
)
```

After working with the integration of CLIPS and ISP there is an advantage that CLIPS/COOL bring to bear on the ease of use for linking CLIPS with external

'real time' data. One such application that used this integrated technology was a prototype to display switch positions from on-board systems to Space Shuttle ground controllers. The prototype was up and running within three days. The display technology had already been developed as part of a training tool to help astronauts learn procedures for the Space Habitation Module. The display technology was then reused and combined with CLIPS and ISP to monitor telemetry and react whenever subscribed data were detected. CLIPS was needed deduce single switch settings based on the downlinked telemetry data. For example, several parameters may contain measurements of pressure across a line. If most of the pressure sensors begin to fall low, then a pressure switch might have been turned off.

An Open Toolkit Approach

So far, the discussion has focused on integration of CLIPS and ISP by embedding ISP into CLIPS. This has the advantage of hiding the details of the ISP client API from developers of CLIPS applications; however, it is easy to imagine applications which are no more "CLIPS applications" per se than they are "ISP clients". CLIPS and ISP provide distinct services, so it is not immediately obvious which ought to be embedded in the other, or whether embedding is even necessary.

The third approach we used to integrating CLIPS and ISP was implemented into the oimon application, discussed below. Instead of embedding ISP inside CLIPS, the two APIs are treated as "peers" within the application. Consider the following definitions:

- **open system**: a system which makes its services available through a callable API,

- **open toolkit**: an open system which permits the caller to always remain in control of execution.

The primary distinction between these two is that open systems may require that the caller turn over complete control of the application (e.g., by calling a `MainLoop()` function).

Open toolkits permit the caller to exploit the systems' functionality while maintaining control of the application; it real-time applications, this can be critical. Examples of open toolkits include the X-toolkit, the ISIS distributed communications system, and Tcl/Tk. Figure 3 depicts a number of the CLIPS and ISP API functions. Although both have functions which will take control of the application (i.e., `Run(-1)` and `ItMainLoop()`), the use of these functions is not required; lower level primitives are available to fine-tune execution of the two systems (i.e., `Run(1)` and `ItNextEvent()`, `ItDispatchEvent()`). By the definitions above, both CLIPS and ISP are open toolkits. As a result, they may both be embedded in a single application which chooses how and when to dispatch to each.

```
• ISP:
    - ItInitialize()
    - ItPublish(), ItSubscribe()
    - ItConnectServer(), ItDisconnectServer()
    - ItNextEvent(), ItDispatchEvent()
• CLIPS:
    - InitializeCLIPS()
    - AssertString()
    - Reset()
    - Run()
```

Figure 3: Elements of the CLIPS and ISP appication program interfaces

OIMON

The Operational Instrumentation Monitor (oimon) is being developed as a MCC application run at Instrumentation/Integrated Communications Officer (INCO) console workstations. The program is useful to other non-INCO flight controllers, since it publishes (via ISP) the status of certain electronic components which may affect the validity of data in the telemetry stream. The paragraphs which follow outline oimon and discuss how ISP and CLIPS have been integrated into it as peer open toolkits.

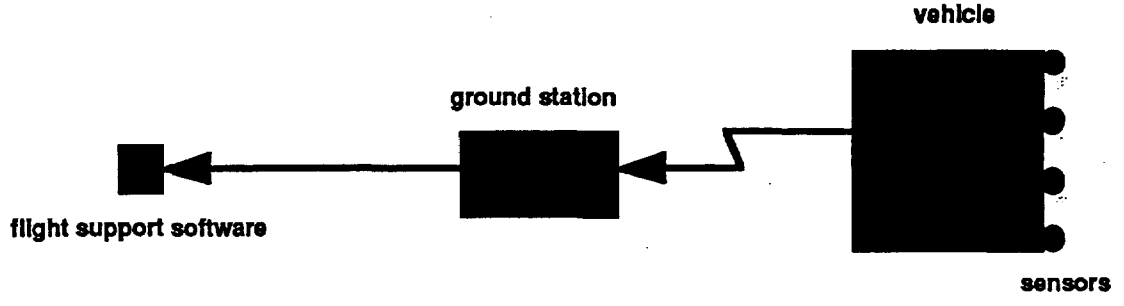


Figure 4: Shuttle sensor "channelization"

Figure 4 presents a summary of the data flow originating at sensors on board the Shuttle (e.g., temperatures, pressures, voltages, current) and ending up on a flight controller's display or in some computation used by a flight control application. The significant aspect of this figure is that there are a number of black boxes (multiplexer/demultiplexers -- MDMs and discrete signal conditioners -- DSCs) which sit in the data path from sensor to flight controller. A number of these black boxes (the so-called operational instrumentation (OI) MDMs and DSCs) are the responsibility of the INCO flight control discipline. There are other black boxes managed by other disciplines, for example the "flight critical" MDMs. The oimon application is concerned with the OI black boxes.

Failure of an OI MDM or DSC can corrupt the telemetry data for a number of on-board sensors. As a result, most flight controllers and many flight control applications are interested in the status of MDMs and DSCs. Instead of requiring that each consumer of telemetry data individually implement the logic necessary to assess MDM/DSC status, the INCO discipline will run oimon as an ISP client which publishes the status of the OI MDMs and DSCs. Any consumer of data affected by a particular OI black box may subscribe to its status through ISP and thus effectively defer OI MDM/DSC to the INCO discipline. This deferral of responsibility to the appropriate discipline is one of the benefits of a software architecture which promotes data sharing between different applications on different workstations.

The oimon application is a C-language program which makes calls to the ISP API to obtain its input data, the CLIPS API to execute the pattern matching necessary to infer the MDM/DSC statuses, and the ISP API to publish its conclusions. There are no explicit parameters in the downlist which unambiguously indicate OI black box status, and this is the reason CLIPS is needed. The major elements of oimon of relevance here are

- a set of CLIPS rules which implement the pattern matching,
- callback functions invoked whenever a telemetry event occurs,
- assertions of CLIPS facts from within the callback functions, and
- a main loop which coordinates CLIPS rule firing and ISP event dispatching.

A template of oimon is shown in Figure 5.

```
isp_event_callback(){
    ...
    sprintf( fact, ...);
    AssertString(fact);
    ...
}

main(){
    ...
    ItAddCallback( ...isp_events... );
    while(True){
          while( moreISP() ){
                ItNextEvent( ... );
                ItDispatchEvent(...);
          }

          while( moreCLIPS() ){
                Run(1);
          }
    }
}
```

Figure 5: oimon code template

The CLIPS rule base is constructed so as to implement a number of tests currently used by INCO flight controllers to manually assess MDM/DSC status and to enable/disable some tests based on the results of others. In particular, these tests are (1) the MDM wrap test, (2) MDM and DSC built-in test equipment tests, (3) power bus assessment, and (4) a heuristic test developed by INCO flight controllers to deduce OI DSC status based on a handful of telemetry parameters which are connected to the DSCs through each of the DSC cards and channels.

The oimon application is still under development. However, preliminary experience with it suggests that the integration of CLIPS and ISP as peer toolkits called from a main application is not only feasible but easily implemented. Preliminary experience with this approach to CLIPS/ISP integration has revealed one advantage of it over embedding ISP in CLIPS. Under certain circumstances, the invocation of CLIPS functions can invoke the CLIPS "periodic action". When CLIPS is embedded in ISP, this can cause ISP events to "interrupt" the execution of the consequent of an ISP rule. In the peer toolkit approach, ISP functionality is not invoked using the CLIPS periodic action and thus this behavior does not exist. Subsequent testing of oimon will focus on tuning the event-dispatching/rule-firing balance to ensure that oimon is neither starved of telemetry nor prevented from reasoning due to high data rates.

Summary

This paper has outlined three approaches we took to integrating the CLIPS inference engine and the ISP client API into single applications. A summary of a "data layer" approach was given, but this approach was not actually implemented. A similar method was also described in which ISP API calls are embedded in CLIPS, and ISP event processing is handled as an ISP "periodic action". The CLIPS syntax for this approach was presented. A quick prototype was developed based on this second approach, and the prototype demonstrates the soundness of the technique. In particular, it permitted very rapid development of the application. Unlike the first two approaches, the third approach we discussed did not embed ISP in CLIPS. Rather the CLIPS and ISP APIs are invoked as "peer toolkits" in the C-based oimon application. This application is currently being tested against STS-68 flight data, but additional development is expected. Preliminary results from oimon suggest that the peer toolkit approach is also sound. The possibility of ISP events interrupting the firing of ISP rules is eliminated in the oimon approach, since ISP is invoked directly from the application instead of being called as a CLIPS periodic action.

References

Paul J. Asente and Ralph R. Swick, **X Window System Toolkit**, Digital Press, 1990.

Joseph C. Giarratano and Gary Riley, **Expert Systems Principles and Programming**, PWS Pub. Co.

Joseph C. Giarratano , **CLIPS User's Guide**, NASA JSC-25013

John K. Ousterhout, **Tcl and the Tk Toolkit**, Addison-Wesley, 1994.

G. Riley, *CLIPS: An Expert System Building Tool*, Proceedings of the Technology 2001 Conference, San Jose, CA, December 1991.

The ISIS Distributed Toolkit Version 3.0 User Reference Manual, Isis Distributed Systems, 1992.

The Meteorological Monitoring System for the Kennedy Space Center/Cape Canaveral Air Station

Allan V. Dianic
(alland@fisher.css.gov)
ENSCO, Inc. Applied Research and Systems Division
Melbourne, Florida

Abstract

The Kennedy Space Center (KSC) and Cape Canaveral Air Station (CCAS) are involved in many weather-sensitive operations. Manned and unmanned vehicle launches, which occur several times each year, are obvious examples of operations whose success and safety are dependent upon favorable meteorological conditions. Other operations involving NASA, Air Force and contractor personnel -- including daily operations to maintain facilities, refurbish launch structures, prepare vehicles for launch and handle hazardous materials -- are less publicized but are no less weather-sensitive.

The Meteorological Monitoring System (MMS) is a computer network which acquires, processes, disseminates and monitors near real-time and forecast meteorological information to assist operational personnel and weather forecasters with the task of minimizing the risk to personnel, materials and the surrounding population. CLIPS has been integrated into the MMS to provide quality control analysis and data monitoring. This paper describes aspects of the MMS relevant to CLIPS including requirements, actual implementation details and results of performance testing.

1.0 Introduction

The Meteorological Monitoring System (MMS) is designed to operate in a networked environment for the support of meteorological and operational personnel (see Figure A.1 on page 7). The MMS consists of a Preprocessor (PPR) and one or more Monitoring and Display Station (MDS). The MMS Preprocessor acquires data from sensors located in and around the KSC/CCAS area, and places them into one of the MMS data classes. These classes categorize data into common types for uniform processing (see Figure A.2 on page 7). The data are then subjected to a quality-control check by an embedded CLIPS implementation and disseminated to the network of MMS Monitoring and Display Stations (MDS). The MDS, using a second MMS implementation of CLIPS, provides the end user with a tool to monitor weather and generate warnings and alerts when weather conditions violate published criteria. The system maintains a database of operations and associated weather criteria for the user to select. Once activated, the meteorological constraints for an operation are transformed into a series of CLIPS rules which, along with a current stream of near real-time and forecast data, are used to trigger alarms notifying the user of a potentially hazardous weather condition. Several user-defined functions have been added to CLIPS, giving it the ability to access MMS resources and alarms directly.

The use of CLIPS for this effort was in large part a research effort: we were interested in making use of a tool that could add a great deal of flexibility and power to the QC and monitoring tasks. However, we were concerned about its implementation and the extent to which performance would have to be sacrificed in exchange for that flexibility. This paper describes aspects of the MMS relevant to CLIPS including requirements, actual implementation details and results of performance testing. The conclusion will discuss the overall success of the effort and future expan-

sion possibilities.

The MMS was tested on a DECstation 5000/125 uses a MIPS R3000A processor running at 25 MHz with 32 megabytes of memory. This was a capable mid-level workstation in 1991, but is slower than comparably priced systems currently available. The system was developed under Ultrix using C, CLIPS version 5.0, X windows and SL-GMS, a tool for dynamic graphical screen management.

2.0 Quality Control

Quality control is a function that attempts to identify and tag erroneous weather measurements and observations. Missing or invalid data may be introduced by hardware or software failures and must be identified so that they do not negatively affect the operation of the MMS.

The MMS Preprocessor is responsible for acquisition, quality control analysis and dissemination of meteorological data. CLIPS has been embedded in the quality control (Data_QC) module, and analyzes each data point processed by the system. The Preprocessor currently ingests a minimum of 1000 weather measurements each minute; currently these data are contained in approximately 11 kilobytes of ASCII text.

2.1 Implementation

Data_QC runs as a background process monitored by the system health facility. During initialization, the data templates representing MMS data classes are loaded into the shell. Data_QC then waits for a message from the data ingest routine indicating that a set of data is ready to be analyzed. The process executes a Unix 'fork' after identifying the data class of the incoming data. While the parent process waits for the child to complete, the child loads raw data into the shell using the proper data template. QC rules are loaded next, and the shell decision process is initiated.

Quality control rules for each data class are structured to operate in two layers. The first layer of rules performs the actual analysis of the data while the second provides the mechanism for saving the data following the operation. The order of execution is controlled by setting the salience of the rules such that the second layer follows the first. For example, 'windspeedqc' is a first level rule which evaluates wind facts for a specific condition (a value outside of a range). When the condition is satisfied (wind speed in error), the fact being examined is retracted and reasserted with the appropriate QC tag. The second level rule 'windsavefacts' exists only to save each fact using the user defined function 'record_met_fact.' After the shell concludes execution, the child process saves the list of recorded facts to a disk file and sends a control message to trigger distribution of the data to the MMS network. The child process then terminates, and the parent waits for the next message.

The current set of rules implements simple range checking. A more sophisticated set of analysis rules could be easily developed and implemented. Quality control rule files are stored in a standard location (a library directory) and are reloaded for each execution, thus allowing for changes to be made while the Preprocessor continues to operate. Each data class has its own rule file, and modified rules will take effect during the next execution cycle for the affected data class.

2.2 Performance

The use of CLIPS for the QC function adds to the power and flexibility of this facility; however, the capabilities of the shell must also be considered in terms of its ability to complete all tasks within specified time requirements. If it cannot, then an alternate method would have to be implemented.

The CLIPS/QC processing times were obtained by inserting time checkpoints at different locations in the code and performing a series of test executions. The data obtained from these tests were used to calculate the maximum, minimum and mean performance times for each data class currently processed. The data, summarized in Table 2.1, expresses both a base processing estimate and three total processing estimates. The base estimate expresses the maximum, minimum and mean time for the quantity of data expected to arrive each minute. The total processing estimates are based on the average performance of the shell's processing of lightning data. The minimum, maximum and average times are multiplied by the number of lightning strikes (20, 60 or 100) and added to the base processing estimate listed in the first section of the table.

Table 2.1 CLIPS/QC Performance (all times are in seconds)

Class	*Iterations*	*Data Pts per Iteration*	*Maximum*	*Minimum*	*Average*
Wind	37	62	6.870	2.150	3.936
Temperature	37	69	4.850	1.470	2.378
Elec. Potential	31	625	22.430	7.540	11.457
General Met	37	1	0.340	0.180	0.241
			---	---	---
Base Processing Estimate			34.490	11.340	18.012
Single Lightning Strike	220	1	0.380*	0.180	0.240
Total processing estimates including lightning:					
Total/20 strikes	42.090	14.940	22.815		
Total/60 strikes	57.290	22.140	32.420		
Total/100 strikes	72.490	29.340	42.026		

* The maximum value within 3-standard deviations was used, above which there were only four points: 0.410, 0.430, 0.560, 0.590. The sample standard deviation was 0.053.

The performance of the shell in this implementation is well within the time requirement for handling expected data loads. The minimum one-minute data load had an average processing time of 18.012 seconds, which is only 30 percent of the maximum available time. In the MMS's current environment (Florida), lightning activity is a daily issue during the summer months. Data loads in excess of one strike per second are experienced frequently and such loads increased the average processing time to 32.420 seconds (54 percent of maximum); 100 strikes per minute required 42.026 seconds (70%). These results confirm that CLIPS can handle such a task in a near real-time environment. Admittedly the current rules are very simple, but this was necessary to validate a minimum ability to operate in this environment. The unanswered question remaining to be explored concerns enhancements to the complexity of QC rules and the effect such enhancements would have on processing times. While such a discussion is not within the scope of this paper, it

seems that enhancements are possible. Average idle time for the QC function was 18 seconds (30%) out of each minute, which would indicate available capacity for more sophisticated analysis techniques. Additional consideration must be given to the fact that newer, faster hardware would certainly provide faster processing of shell tasks.

3.0 Monitoring

The MMS Monitoring and Display Station (MDS) is responsible for receiving, monitoring and displaying meteorological and forecast data. CLIPS has been embedded in the monitoring (Data_Monitor) module and controls the activation and deactivation of alarm conditions based upon data and constraints. The MDS provides the user with all the necessary user interfaces for controlling and observing Data_Monitor activity.

3.1 Implementation

Data_Monitor consists of two separate processes; the Monitoring_Controller and Monitoring_CLIPS. Monitoring_Controller runs as a background process monitored by the system health facility. Monitoring_CLIPS is a module containing both CLIPS and user defined functions.

Monitoring_Controller waits for a message from the another MDS process which would indicate a change in status (i.e. new meteorological data received, operation activation, monitoring pause or resume). The controller uses a simple scheduling control mechanism to record such events and trigger shell executions. When a shell is to be executed, the controller retrieves all active meteorological constraints and builds CLIPS rules to search the fact base. Rules for each data class are written into separate disk files. Monitoring_CLIPS is started by Monitoring_Controller through a Unix 'fork' to process rules and data for a specific data class. One Monitoring_CLIPS is started for each data class scheduled to be processed. The Monitoring_Controller then waits for the children to complete their executions.

Monitoring_CLIPS initializes the shell, loads the data templates representing MMS data classes and then loads data for the class specified. Rules built by the controller are loaded last, and then the shell is executed through a 'RunCLIPS' function call. The monitoring rules are built differently based upon the current violation state of each constraint. An unviolated constraint is translated into a 'normal' rule set consisting of two rules. A constraint currently in violation is translated into a 'deactivation' rule set consisting of three rules.

The first rule of the 'normal' set verifies the existence of at least one applicable data point and asserts an enabling fact if such a point is found. The left-hand side (LHS) of the main constraint rule checks the existence of the enabling fact and examines the appropriate meteorological facts and QC tags. If the LHS is satisfied, the RHS will trigger an alarm signal using the user defined functions 'mtu_set_violation' and 'send_activation_message' so that the user interface will activate a visual and audible alarm.

The first rule of the 'deactivation' set is the same as that in the 'normal' set, as is the LHS of the main rule; however, the RHS differs in its structure and function. Since the goal of the 'deactivation' set is the opposite of the 'normal' set, its function is to search for a counterexample to the premise that no violation exists. If the counterexample exists in the fact base, the RHS will retract the enabling fact asserted in the first rule and undefine the third rule of the 'deactivation' set. No additional alarm is generated, and the screen icon in the user interface will continue to indicate a

violation. Without the existence of a counter example, the RHS of the third rule will clear the violation by using the user defined functions 'mtu_set_violation' and 'send_deactivation_message.'

A practical example of the monitoring function may be found by considering the following scenario: a payload canister is to be hoisted at KSC launch complex 39A. The meteorological constraints for that operation include a steady state wind limit of 17.2 knots and the absence of actual or forecast lightning activity within a 5 nautical mile radius. An MMS user could activate the 'Hoisting' operation from the list of those available, which would retrieve the associated constraints, load them into the monitoring table and notify the Monitoring_Controller process. Rules for wind and lightning strike data will be built from the constraints using the 'normal' form, and the two shells will be scheduled to execute. For wind class data two rule sets would be built: Hoist_1, which will alert when wind reported by sensor 9 exceeds of 17.2 knots, and Hoist_2, which does the same using sensor 10. The execution sequence for the wind class rules are depicted in Figure B.1 on page 8. Notice that the Hoist_1 rule violates after finding the sensor 9 wind speed of 19 knots at time=0. The violation continues as the measurement for time=1 still exceeds the threshold. Finally at time=2 the sensor 9 wind speed has passed below 17.2 knots. Notice that from time=0 through time=2 that sensor 10 was reporting invalid wind speed datum, and the Hoist_2_enable rule didn't fire; only when a valid measurement arrived at time=3 did Hoist_2_enable find a valid wind speed, and therefore assert the enabling fact (enable_Hoist_2).

3.2 Performance

The monitoring of meteorological measurements and observations can be greatly enhanced by the use of an expert system shell. In addition to finding violations of individual meteorological parameters, special sets of rules could search the fact base for a number of different conditions which may signal the onset of severe or threatening weather. The concern in using CLIPS, or any other shell, is one of performance. While the monitoring task has a less stringent time requirement than that of CLIPS/QC, it is difficult to express that value as a specific number. Processing certainly must be completed within the time period of the data, which is generally one minute. Occasional spikes above that level are acceptable so long as the overall performance does not exceed that level.

The CLIPS/Monitoring processing times were obtained by inserting time checkpoints at different locations in the code and performing a series of test executions. Tests were performed using the full MDS software with a test driver providing a full set of meteorological data at normal intervals. The number of constraints and facts were varied to test the performance characteristics of this implementation. Time samples were taken for constraint levels of 1, 10, 20, 30,...,100 and for fact levels of 0, 100, 200, 300,...,1000.

The performance of the shell was well within the time requirements stated above. Results from the test sample data set, consisting of 1038 time samples and summarized in Table 3.1, indicated that average performance for a heavy load will require about 15 seconds for setup (i.e. fact assertion, rule construction and loading) and about 30 seconds for the shell to complete processing. The resulting total of approximately 45 seconds is only 75% of the one minute target. Such results bode well not only for the ability of CLIPS to handle the current workload, but its ability to handle more complex tasks involving more a greater number of facts and more complex rules.

Additional performance analysis has been provided in Appendix C on page 10. Included in this appendix are three graphs that describe the manner in which actual shell processing time was

observed to vary in response to different combinations of values for the number of constraints and the number of facts. (Note: These times exclude the amount of time required to build rules and assert meteorological facts.)

Table 3.1 Monitoring Performance Tables (all times in seconds)

Setup Times			Facts			
Constraints	100		500		1000	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
1	2.03	0.03	7.92	0.48	15.40	0.95
10	2.27	0.27	7.70	0.39	15.16	1.04
30	2.18	0.18	7.83	0.02	15.00	0.55
50	2.17	0.15	7.96	0.48	15.44	1.52
Processing Times			Facts			
Constraints	100		500		1000	
	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
1	0.22	0.04	1.04	0.38	1.84	0.32
10	1.52	0.03	7.04	0.70	13.99	1.14
30	3.59	0.01	16.58	0.13	28.01	0.86
50	4.13	0.23	15.85	0.20	29.82	0.49

An interesting CLIPS performance characteristic is the apparent "levelling-off" of the processing time around 30 constraints. This can be seen in the bottom figure of Appendix B, which shows "crunch time" (i.e. shell processing time) as a function of the number of constraints for selected levels of the number of facts. In each of the curves in this figure, the observed processing time increases in a roughly linear fashion when the number of constraints was less than about 30, and the slopes of these curves increase as the number of facts increases. After 30 constraints, however, these curves level-off to be nearly horizontal. At this point, the effect of increases in the number of constraints on processing time is quite minimal. This is significant when we consider that the MMS monitoring table is currently capable of holding 1000 constraints. Since the performance times seem to level-off above a relatively modest number of constraints, activation of a much larger group should not result in excessively large execution times.

4.0 Conclusion

The CLIPS expert system shell performs well in both MMS implementations. Although neither the quality control nor the monitoring implementation constitutes a complex use of expert systems technology, this application of CLIPS is useful in several ways. First, it is currently functional and useful, and data are processed and monitored in an effective manner within all time requirements. Second, enhancements are possible that would more fully exploit CLIPS capabilities. Third, the system has potential for growth through the utilization of a more powerful set of rules and the use of another CLIPS tool known as COOL (CLIPS Object Oriented Language). The primary objective of the MMS project was to create a system that would be useful to weather forecasters and operational personnel. We believed that CLIPS could greatly enhance the system and help us to reach our objective, but we also had reservations concerning implementation and performance. It is now clear that the inclusion of CLIPS was a proper decision based upon its performance and capabilities.

Appendix A MMS Architecture

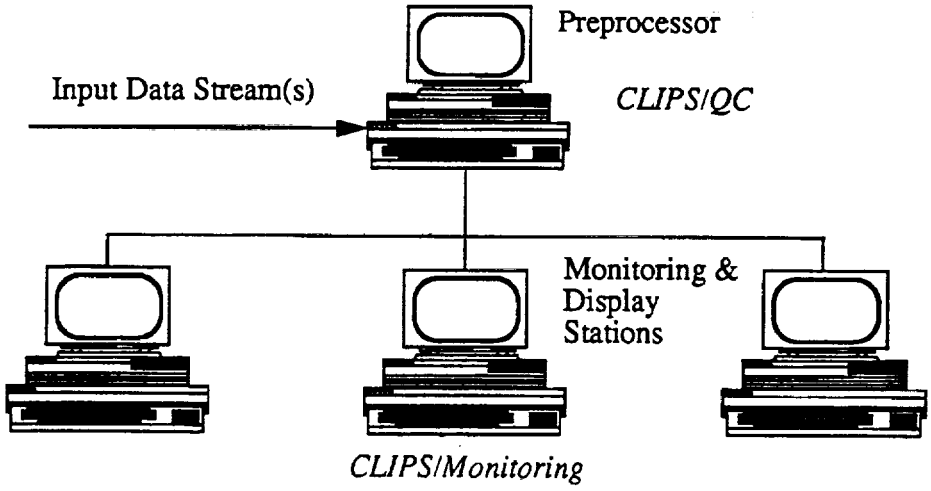


Figure A.1 MMS network overview

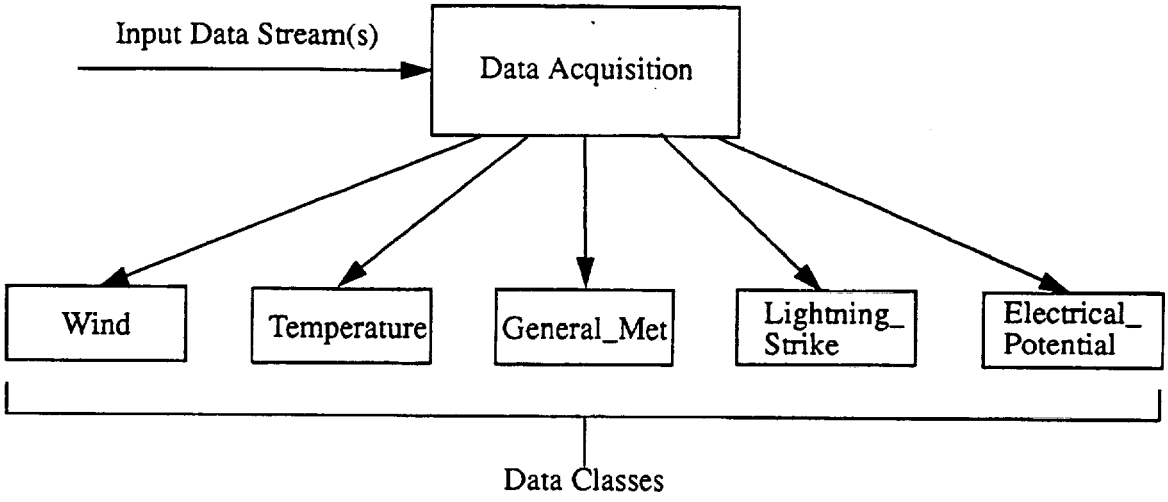


Figure A.2 MMS data classes

Appendix B Sample Monitoring Scenario

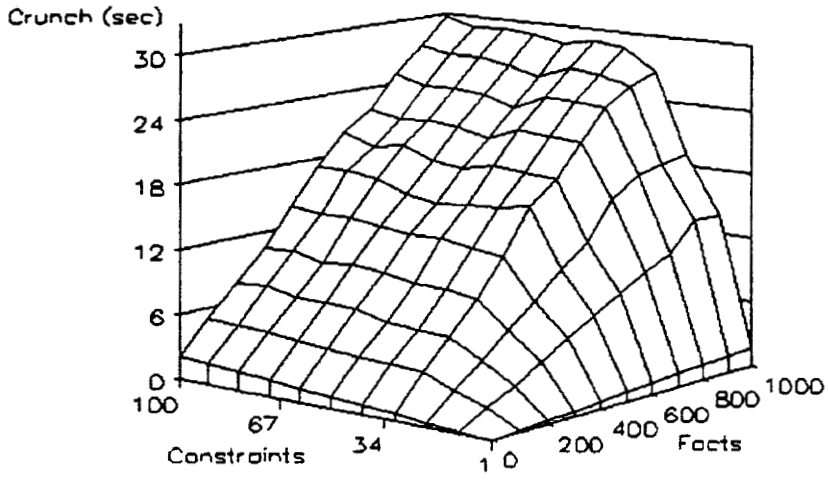
Fact Base	Rules	Alarms	Time
Sen 9 Spd 19 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 16 QC-Ok Sen 12 Spd 15 QC-Ok Sen 13 Spd 16 QC-Ok	Hoist_1_enable Hoist_2_enable Hoist_1_main Hoist_2_main *New data received, rules built and loaded.*	None	0
Sen 9 Spd 19 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 16 QC-Ok Sen 12 Spd 15 QC-Ok Sen 13 Spd 16 QC-Ok enable_Hoist_1	Hoist_2_enable Hoist_1_main Hoist_2_main *Hoist_1_enable finds data, asserts enabling fact, and undefines itself. Hoist_2_enable finds no data (sensor 10 QC is bad).*	None	0
Sen 9 Spd 19 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 16 QC-Ok Sen 12 Spd 15 QC-Ok Sen 13 Spd 16 QC-Ok	Hoist_2_enable Hoist_1_main Hoist_2_main *Hoist_1_main finds wind violation, triggers alarm and retracts enabling fact.*	Hoist WIND	0
Sen 9 Spd 18 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 15 QC-Ok Sen 12 Spd 13 QC-Ok Sen 13 Spd 16 QC-Ok	Hoist_1_enable Hoist_2_enable Hoist_1_main Hoist_2_main Hoist_1_Deactivate *New data received, rules built and loaded.*	Hoist WIND	1
Sen 9 Spd 18 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 15 QC-Ok Sen 12 Spd 13 QC-Ok Sen 13 Spd 16 QC-Ok enable_Hoist_1	Hoist_2_enable Hoist_1_main Hoist_2_main Hoist_1_Deactivate *Hoist_1_enable finds data, asserts enabling fact, and undefines itself. Hoist_2_enable finds no data (sensor 10 QC is bad).*	Hoist WIND	1
Sen 9 Spd 18 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 15 QC-Ok Sen 12 Spd 13 QC-Ok Sen 13 Spd 16 QC-Ok	Hoist_2_enable Hoist_1_main Hoist_2_main *Hoist_1_main finds wind violation, undefines deactivation rule and retracts enabling fact.*	Hoist WIND	1

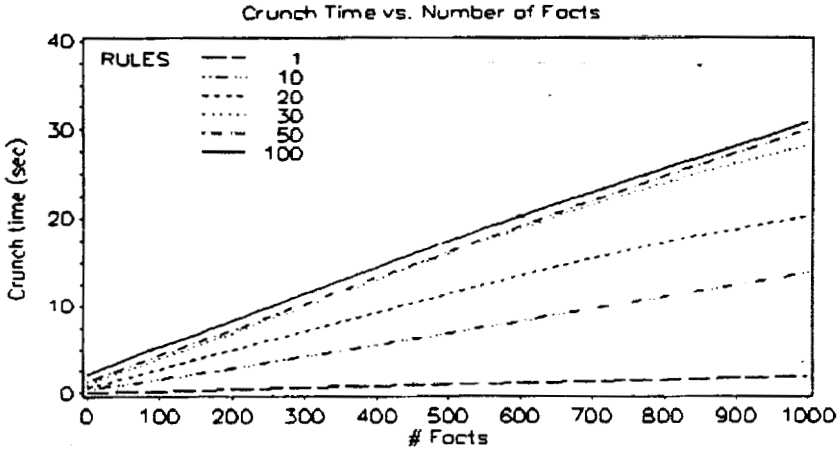
Figure B.1 Monitoring Scenario

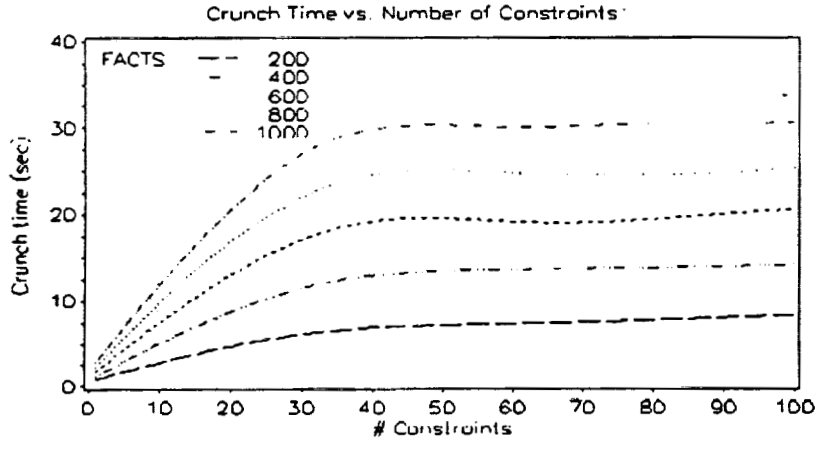
Fact Base	Rules	Alarms	Time
Sen 9 Spd 17 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 13 QC-Ok Sen 12 Spd 12 QC-Ok Sen 13 Spd 13 QC-Ok	Hoist_1_enable Hoist_2_enable Hoist_1_main Hoist_2_main Hoist_1_Deactivate *New data received, rules built and loaded.*	Hoist WIND	2
Sen 9 Spd 17 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 13 QC-Ok Sen 12 Spd 12 QC-Ok Sen 13 Spd 13 QC-Ok enable_Hoist_1	Hoist_2_enable Hoist_1_main Hoist_2_main Hoist_1_Deactivate *Hoist_1_enable finds data, asserts enabling fact and undefines itself. Hoist_2_enable finds no data (sensor 10 QC is bad).*	Hoist WIND	2
Sen 9 Spd 17 QC-Ok Sen 10 Spd 99 QC-Bad Sen 11 Spd 13 QC-Ok Sen 12 Spd 12 QC-Ok Sen 13 Spd 13 QC-Ok	Hoist_2_enable Hoist_1_main Hoist_2_main Hoist_1_Deactivate *Hoist_1_main finds no violation. Deactivation rule clears indicator, retracts enabling fact.*	None	2
Sen 9 Spd 17 QC-Ok Sen 10 Spd 19 QC-Ok Sen 11 Spd 15 QC-Ok Sen 12 Spd 14 QC-Ok Sen 13 Spd 16 QC-Ok	Hoist_1_enable Hoist_2_enable Hoist_1_main Hoist_2_main *New data received, rules built and loaded.*	None	3
Sen 9 Spd 17 QC-Ok Sen 10 Spd 19 QC-Ok Sen 11 Spd 15 QC-Ok Sen 12 Spd 14 QC-Ok Sen 13 Spd 16 QC-Ok enable_Hoist_1 enable_Hoist_2	Hoist_1_main Hoist_2_main *Hoist_1_enable finds data, asserts enabling fact and undefines itself. Hoist_2_enable finds data, asserts enabling fact and undefines itself.*	None	3
Sen 9 Spd 17 QC-Ok Sen 10 Spd 19 QC-Ok Sen 11 Spd 15 QC-Ok Sen 12 Spd 14 QC-Ok Sen 13 Spd 16 QC-Ok	Hoist_1_main Hoist_2_main *Hoist_1_main finds no violation. Hoist_2_main finds wind violation, triggers alarm and retracts enabling fact.*	Hoist WIND	3

Figure B.2 Monitoring Scenario (continued)

Appendix C CLIPS/Monitoring Performance Graphs







REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave Blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	Nov/94	Conference Proceedings, September 1994

4. TITLE AND SUBTITLE

Third CLIPS Conference Proceedings - Volumes I and II

5. FUNDING NUMBERS

6. AUTHOR(S)

Gary Riley, editor

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Lyndon B. Johnson Space Center
Houston, Texas 77058
I-NET, Inc.
Houston, Texas 77058

8. PERFORMING ORGANIZATION REPORT NUMBERS

S-785

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

NASA CP-10162

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified/Unlimited
Available from the NASA Center for AeroSpace Information
800 Elkridge Landing Road
Linthicum Heights, MD 21090-2934
(301) 621-0390 Subject Category: 61

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Expert systems are computer programs which emulate human expertise in well defined problem domains. The potential payoff from expert systems is high: valuable expertise can be captured and preserved, repetitive and/or mundane tasks requiring human expertise can be automated, and uniformity can be applied in decision making processes. The C Language Integrated Production System (CLIPS) is an expert system building tool, developed at the Johnson Space Center, which provides a complete environment for the development and delivery of rule and/or object based expert systems. CLIPS was specifically designed to provide a low cost option for developing and deploying expert system applications across a wide range of hardware platforms. The development of CLIPS has helped to improve the ability to deliver expert system technology throughout the public and private sectors for a wide range of applications and diverse computing environments. The Third Conference on CLIPS provided a forum for CLIPS users to present and discuss papers relating to CLIPS applications, uses, and extensions.

14. SUBJECT TERMS

Expert Systems, Programming Languages, Computer Techniques

15. NUMBER OF PAGES
401

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited