*NASA Conference Publication 10049*, Vol. 1

NASA-CP-10049-VOL-1
19960001796

# First CLIPS Conference Proceedings

## Volume I

*Proceedings of a conference hosted by
the NASA-Lyndon B. Johnson Space Center and held at
the NASA-Johnson Space Center
Houston, Texas
August 13-15, 1990*

NASA

# Final Agenda
# First CLIPS Conference

*August 13–15, 1990*
*Gilruth Center, Johnson Space Center*

## Monday August 13

| | | | | | |
|---|---|---|---|---|---|
| 7:30–6:00 | pm | | Late Registration | | |
| 8:30–9:00 | am | | Welcoming Address: Lawrence Herbolsheimer, Deputy Administrator for Code C, NASA | | |

### *PARALLEL SESSIONS*

| | | | | | |
|---|---|---|---|---|---|
| 9:00–10:00 | am | A1 | Engineering Applications | B1 | Intelligent Tutors and Training |
| 10:10–11:10 | am | A2 | Intelligent Software Engineering | B2 | Automated Knowledge Acquisition I |
| 11:20–12:20 | pm | A3 | Network Applications | B3 | Automated Knowledge Acquisition II |
| 12:20–1:10 | pm | | *Lunch* | | |
| 1:15–1:45 | pm | | *Panel Discussion* "The Future of CLIPS," Robert Savely, Chris Culbert, Gary Riley, Brian Donnell | | |
| 1:50–2:50 | pm | A4 | Verification and Validation | B4 | Enhancements to CLIPS–General I |
| 3:00–4:00 | pm | A5 | Space Shuttle Quality Control/Diagnosis | B5 | Enhancements to CLIPS–General II |
| 4:10–5:50 | pm | A6 | Space Shuttle and Real-time Applications | B6 | Medical, Biological, and Agricultural Applications |
| 6:00–7:00 | pm | | *Reception (Cash Bar)* | | |
| 7:00–9:00 | pm | | *Banquet and Keynote Speaker: Dr. F. Story Musgrave, NASA Astronaut* | | |

## Tuesday August 14

| | | | | | |
|---|---|---|---|---|---|
| 8:00–5:00 | pm | | Late Registration | | |

### *PARALLEL SESSIONS*

| | | | | | |
|---|---|---|---|---|---|
| 9:00–10:00 | am | A7 | Quality Control Applications | B7 | Intelligent Databases, and Networks |
| 10:10–11:10 | am | A8 | Space Station Freedom Applications | B8 | User Interface I |
| 11:20–12:20 | pm | A9 | Space Shuttle and Satellite Applications | B9 | User Interface II |
| 12:20–1:30 | pm | | *Lunch* | | |
| 1:30–2:30 | pm | A10 | Artificial Neural Systems & Fuzzy Logic | B10 | Enhancements to CLIPS: Reasoning/Representation |
| 2:40–4:00 | pm | A11 | Parallel and Distributed Processing I | B11 | Enhancements to CLIPS: Object Oriented |
| 4:10–5:30 | pm | A12 | Parallel and Distributed Processing II | B12 | Enhancements to CLIPS: Graphics/X-Windows |

## Wednesday August 15

| | | | | | |
|---|---|---|---|---|---|
| 8:00–12:00 | pm | | Late Registration | | |

### *PARALLEL SESSIONS*

| | | | | | |
|---|---|---|---|---|---|
| 8:30–9:30 | am | A13 | Aerospace Applications | B13 | Advisory Systems I |
| 9:40–10:40 | am | A14 | Simulation and Defense | B14 | Advisory Systems, and Intelligent Tutors |
| 10:50–12:10 | am | A15 | Intelligent Control | | |
| 11:50–1:00 | pm | | *Lunch* | | |

## CLIPS Users Group Organizational Meeting

1:00– 5:00 pm (1) Discussion and approval of a constitution for the CLIPS Users Group
(2) Nomination and election of officers

## MESSAGE FROM THE GENERAL CHAIR FIRST CLIPS USERS CONFERENCE

These *Proceedings of the First CLIPS Users Conference* mark an important milestone in the evolution of CLIPS. Until this time, CLIPS has been used mainly by individuals who had little knowledge of what others were doing. Now with over 3000 sites using CLIPS in government, industry, and academia, this Conference and its Proceedings offers people the opportunity to learn how many others are using CLIPS. This dissemination of the state-of-the-art work offers many potential benefits to all CLIPS users. As more people hear about CLIPS, we expect further growth in the user community. Conferences like this are an important mechanism for increasing the growth of artificial intelligence technology. We welcome your comments concerning CLIPS and how it can be improved.

Joseph C. Giarratano
University of Houston at Clear Lake

## MESSAGE FROM THE BRANCH CHIEF SOFTWARE TECHNOLOGY BRANCH

Since the first general distribution of CLIPS by my section in 1986, CLIPS has been continuously improved. The new version 5.0 of CLIPS that we will release in Fall 1990 has major enhancements by the addition of objects, generic functions, defglobals, integer data type support, and deftemplate type checking. All new versions of CLIPS are thoroughly tested to insure its performance in mission critical applications.

CLIPS comes with a full set of documentation manuals, all the source code in C, a cross reference style and verification tool (CRSV), an intelligent tutor for learning CLIPS, an on-line help facility, and telephone support for technical questions. In addition to the version of CLIPS written in C, a version written in Ada is also available. An executable of CLIPS for the IBM PC and compatibles is available with pulldown menus and mouse support. Another version for the Macintosh designed for the Mac style interface is also supported. Since the complete source code for CLIPS is supplied, it can be compiled to run on any machine with a C compiler. CLIPS has been successfully ported to virtually every brand of hardware including Cray, Next, Connection machine, VAX, HP, and Sun. CLIPS is free to NASA and USAF users, and their contractors. The price to others is $312, and just $187 for universities. There are no royalties or fees for unlimited copies of CLIPS, thus making it a very cost-effective tool for distributing expert systems, and for education.

Our success with CLIPS has led us to develop other advanced technology tools such as *Nets* for the development and delivery of artificial neural networks, *Costmodl* for estimating software development costs, and *Compass* for planning and scheduling. For additional information, please contact the Help Desk at (713) 280-2233.

I am proud to have directed the development of CLIPS and seen it grow from a small internal NASA project to over 3000 sites in just four years. CLIPS is an excellent example of the spin-off benefits of the Space Program in maintaining the competitive edge of this nation in an increasingly competitive world.

Robert T. Savely
NASA/ Johnson Space Center

## MESSAGE FROM COSMIC

COSMIC, NASA's Computer Software Management and Information Center, is proud to cosponsor the First CLIPS Users Group Conference.

Since 1966 COSMIC has been the one central marketing and distribution center for computer software created under NASA funding. The COSMIC inventory presently contains over 1,200 computer programs. In addition to artificial intelligence, application areas include: aerodynamics; composite analysis; computational fluid dynamics; control systems; heat transfer; image processing; optics; project management; reliability; satellite communications; scientific visualization; and UNIX utilities.

While this is not a complete list, it does indicate the breadth of applications. Most programs are sold without licensing restrictions so programs can be used as supplied or incorporated into commercial and/or in-house programming operations.

As part of NASA's Technology Utilization Program, our goal is to save time and money for industry, other government agencies, and academic institutions. The COSMIC inventory, a valuable national resource, is maintained to help the U.S. economy keep its competitive edge.

For assistance in locating appropriate software, you can write, call, FAX, or E-mail to explain your requirements. We will review our database of NASA software and send descriptive abstracts of each appropriate program. There is no charge for this service.

COSMIC
The University of Georgia
382 East Broad Street
Athens, GA 30602

phone (404) 542-3265
FAX (404) 542-4807
E-mail SERVICE@COSSACK.COSMIC.UGA.EDU

## CONTENTS

## A1 Session:
## Engineering Applications

# THREE CLIPS-BASED EXPERT SYSTEMS FOR SOLVING ENGINEERING PROBLEMS

by

**W. J. Parkinson**
**Los Alamos National Laboratory**
**Los Alamos, NM 87545**

**G. F. Luger**
**Department of Computer Science**
**University of New Mexico**
**Albuquerque, NM 87131**

**R. E. Bretz**
**Department of Petroleum Engineering**
**New Mexico Institute of Mining and Technology**
**Socorro, NM 87801**

## ABSTRACT

We have written three expert systems, using the CLIPS PC-based expert system shell. These three expert systems are rule based and are relatively small, with the largest containing slightly less than 200 rules. The first expert system is an expert assistant that was written to help users of the ASPEN computer code choose the proper thermodynamic package to use with their particular vapor-liquid equilibrium problem.

The second expert system was designed to help petroleum engineers choose the proper enhanced oil recovery method to be used with a given reservoir. The effectiveness of each technique is highly dependent upon the reservoir conditions.

The third expert system is a combination consultant and control system. This system was designed specifically for silicon carbide whisker growth. Silicon carbide whiskers are an extremely strong product used to make ceramic and metal composites. The manufacture of whiskers is a very complicated process, which to date, has defied a good mathematical model. The process was run by experts who had gained their expertise by trial and error. A system of rules was devised by these experts both for procedure setup and for the process control. In this paper we discuss the three problem areas of the design, development, and evaluation of the CLIPS-based programs.

## INTRODUCTION

Our goal for these projects was to develop small computer-based expert assistants to help some of the less-experienced engineers at Los Alamos National Laboratory make expert decisions about particular tasks for which they may not otherwise have the expertise.

Although we had available a LISP machine and the sophisticated hybrid expert system shell, KEE, we felt that the "frame" or "object-based" simulation expert system designs were unnecessary and overly complex for our application. We further felt that any usable expert system should be executable with an inexpensive shell and an easily available computer. We therefore focused our search on rule-based shells for the PC family of computers. But we were not sure that the PC-based shells available to us would be adequate for the expert systems that we envisioned. We were pleasantly surprised when we found that CLIPS[1] could handle our tasks.

We have written three expert systems using the CLIPS expert system shell. These three expert systems are rule-based and relatively small. The largest is slightly less than 200 rules. Their size, however, does not preclude them from being useful. The first expert system is an expert assistant that was written to help users of the ASPEN computer code pick the proper thermodynamic package to be used with their particular vapor-liquid equilibrium problem. ASPEN is a large computer code used to design chemical plants and refineries. Vapor-liquid equilibrium and associated energy balance problems are an important part of chemical plant and refinery design. There are 144 possible correlations for vapor-liquid equilibrium calculations that can be obtained using combinations of the built-in ASPEN options. Picking the proper combination requires some expertise in thermodynamics, which may not be a strong area for the plant designer. If the designer uses the wrong correlation, the plant design will be wrong.

The second expert system was designed to help petroleum engineers pick the proper enhanced oil recovery method to be used with a given reservoir. There are several different techniques used for enhanced oil recovery. The effectiveness of each one is highly dependent upon the reservoir conditions. Ultimately, the choice of an enhanced oil recovery technique will be based upon economics. However, there are many techniques and many variations of conditions within each technique. It is therefore impractical to do an economic analysis of each case. This expert system helps the user screen the methods and reduce the required number of economic calculations to just a few.

The third expert system is a combination consultant and control system. This system was designed specifically for the silicon carbide whisker growth. Silicon carbide whiskers are an extremely strong product that are used to make ceramic and metal composites. The manufacture of the whiskers is a very complicated process, which to date has defied a good mathematical model. The process was run by experts who had gained their expertise by trial and error. A system of rules was devised by these experts both for procedure set up and for process control. These rules make up the expert system used for this process. This expert system is especially useful because the silicon carbide whisker process is a candidate for technology transfer. With the expert system, we can transfer complicated technology that is not well enough understood to model mathematically without transferring the experts as well.

## THE ASPEN COMPUTER CODE

One goal was to develop an expert system to help engineers design refinery and chemical processes. The modern design of refinery or chemical plant operations requires a chemical process simulation computer code. The simulation code used by Los Alamos is ASPEN[2] (Advanced System for Process ENgineering). It was developed for the United States Department of Energy at the Massachusetts Institute of Technology. Similar software products are available in the commercial marketplace, but because of ASPEN's solids handling capability and its wide choice of built-in thermodynamic packages, it was considered to be the only product suitable for the design of coal and oil shale processing units. Thus, ASPEN was the product of choice for use in LANL's recent coal and shale oil projects.

To demonstrate why an expert system is useful, it is first necessary to give a brief description of how ASPEN works. Then the descriptions of thermodynamic packages and of our expert system for determining the proper thermodynamic package for use with ASPEN will demonstrate why and how this program is useful.

With ASPEN, a chemical plant is represented by modules that are tied together by material flow streams. The modules may, in turn, be represented by building blocks that are unit operations common to many different types of processing modules. The ASPEN package also includes a large thermodynamic database and a code that allows mass and energy balances to be performed to aid the designer in sizing, specifying, and selecting equipment and/or determining optimum processing conditions.

For example, Fig. 1 is a block flow diagram of an oil shale processing plant. Each block represents a major plant module, including modules for a retort, a gas-liquid separation and cleanup unit, a hydrogen recovery unit, a hydrogen plant, and a hydrotreater unit. The solid lines represent material flow streams in the plant. In developing a block flow diagram like that in Fig. 1, the designer supplies instructions to ASPEN that specify the compositions, conditions and rates of raw materials, and operating conditions, and identify the module and the flow streams between modules. Output from ASPEN contains information on the rates, temperatures, pressures, and compositions of the flow streams between and from the modules.

Figure 2 is the building block, or unit operation, flow diagram for the retort module of Fig. 1. In Fig. 2, the unit operation blocks, pumps, mixers, heaters, reactors, etc., are displayed in a logical design sequence. The solid lines represent mass flow, and the dotted lines represent energy flow. When the flow stream information and conditions are specified and calculated for the block diagram of Fig. 1 and additional instructions are supplied for each building block, calculations can be performed in ASPEN to yield rate, temperature, pressure, composition, and energy requirements for each building block and for the flow streams between the building blocks. This output is used by the designer to make economic calculations, to specify equipment, or to set the optimum operating parameters for the unit.

In the retort module, the oil shale unit undergoes processes of heating, thermal decomposition, combustion, and vaporization to name a few. Calculations involving these processes require information on enthalpy, heat capacities, heats of reaction, and vaporization for materials that contain so many components that the compositions must often be expressed in terms of fractions containing components with nearly like-properties. Measured thermodynamic data are often sparse over the range of compositions and conditions encountered, and even if available, would be clumsy or time consuming for use with computer codes. Thus,

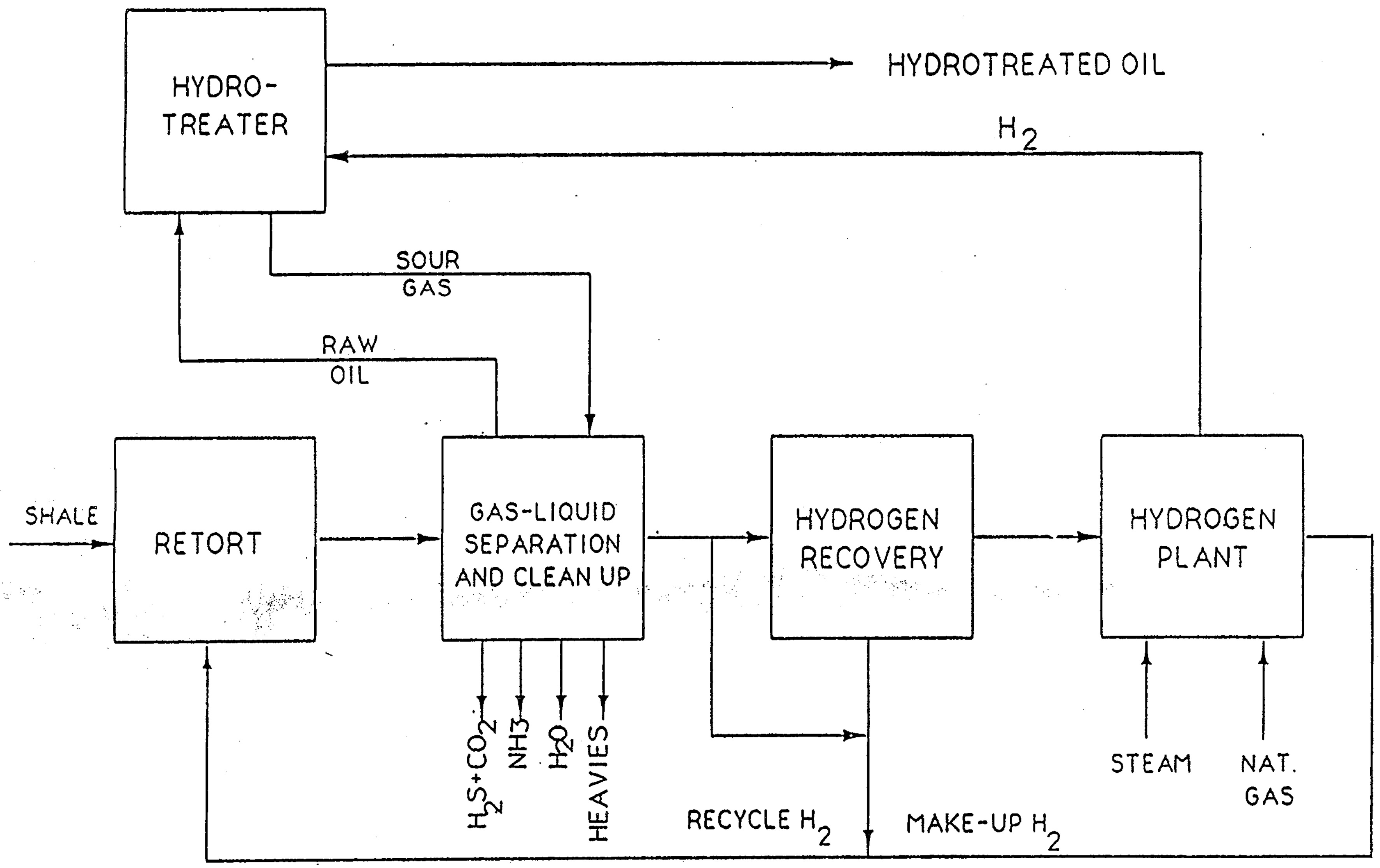


**Fig. 1. ASPEN block flow diagram for an oil shale plant.**

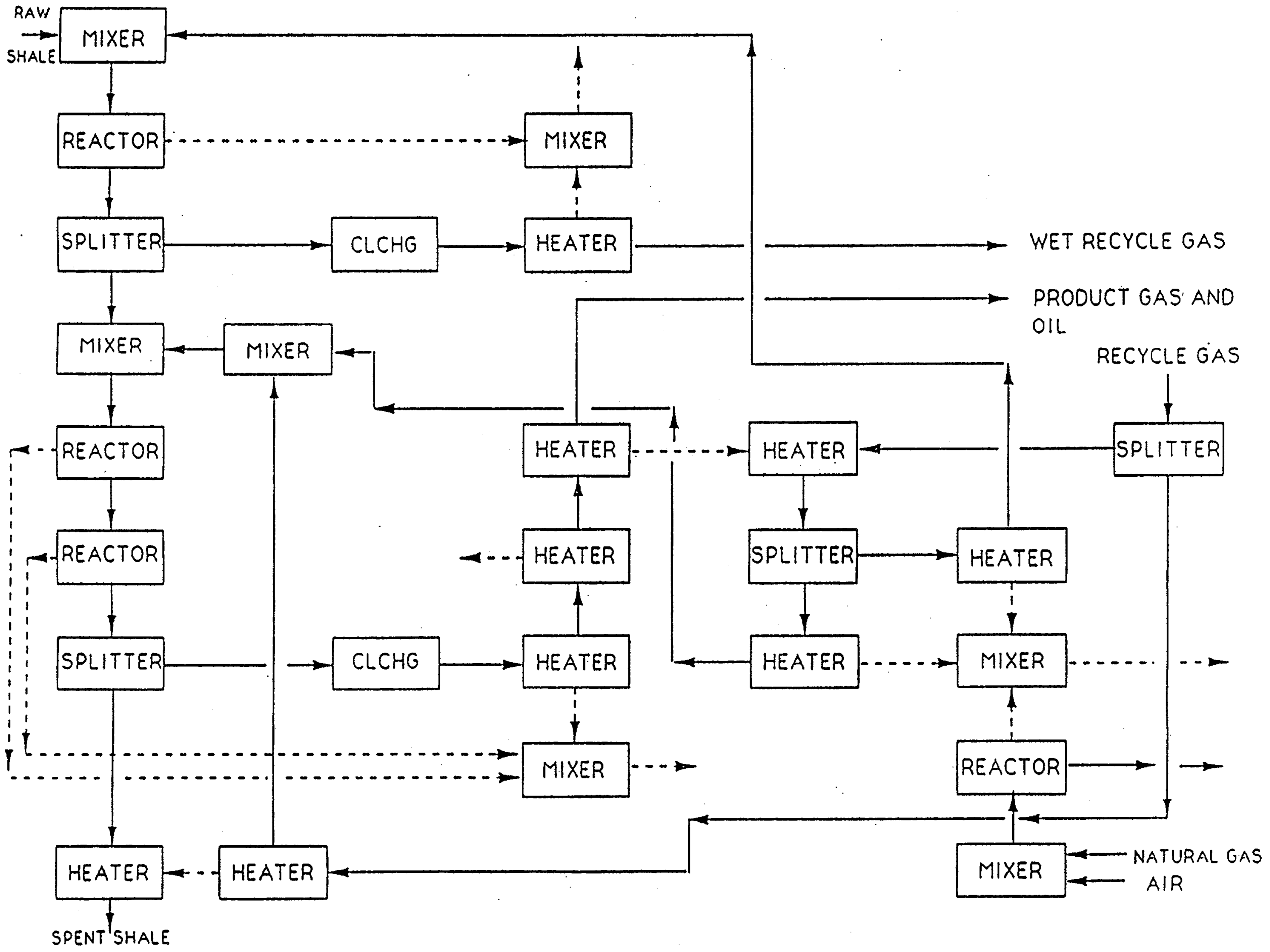


Fig. 2. ASPEN block flow diagram for an oil shale retort.

the ASPEN code provides the designer with several thermodynamic models, or packages, that provide data for the calculations described above.

Figure 3 is the ASPEN building block diagram for the gas-liquid separation and cleanup module. The diagram requires less equipment than the retort, but, more importantly, it differs from the diagram for retort (Fig. 2) in the type of process being carried out. As shown in Fig. 3, like the retort module there are heating, pumping, and mixing blocks, but in addition, here there are water separation, hydrocarbon phase separations, and impurity removal processes. Although these separations also require thermodynamic data for modeling, the nature of these processes differs significantly from those in the retort module.

Not surprisingly, the thermodynamic package that is appropriate for use with calculations in the retort module is likely very different from the appropriate package for use with the gas-liquid and cleanup module. Even within the gas-liquid separation and cleanup module, the best thermodynamic package for use in performing calculations on the different unit operations may differ. For instance, the best package to use in separating impurities, which may be highly polar compounds, might be different from that used with the flash unit in which the liquid-vapor phase behavior of relatively nonpolar compounds is described.

Also not surprisingly, many designers, when first assigned the task of using ASPEN, do not have the necessary experience and training to know the strengths and weaknesses of the different thermodynamic packages available to them. Those neophytes often must rely on the advice of more experienced designers until they gain their own knowledge and experience.

As an example of the computations involved, ASPEN contains two basic thermodynamic model types for predicting equilibrium K-values, that is,

$$K_i = y_i/x_i = \phi_i^L/\phi_i^V \quad , \qquad (1)$$

where $y_i$ = the mole fraction of component i in the vapor phase,
$x_i$ = the mole fraction of component i in the liquid phase,
$\phi_i^L$ = the fugacity coefficient of component i in the liquid phase, and
$\phi_i^V$ = the fugacity coefficient of component i in the vapor phase.

In K-value calculations, the vapor phase fugacity coefficients are always calculated using an equation of state (EOS) thermodynamic model. For liquids that are ideal mixtures or nearly ideal mixtures, such as mixtures of hydrocarbons, EOS are also used to predict liquid fugacities. When both the liquid and vapor phase fugacities are predicted using an EOS, the model is called an EOS model. Several such EOS models are available with ASPEN. Many EOS are based, at least in part, on fundamental concepts of the physical properties of matter, while others may be purely empirical. And each different EOS offers an advantage over

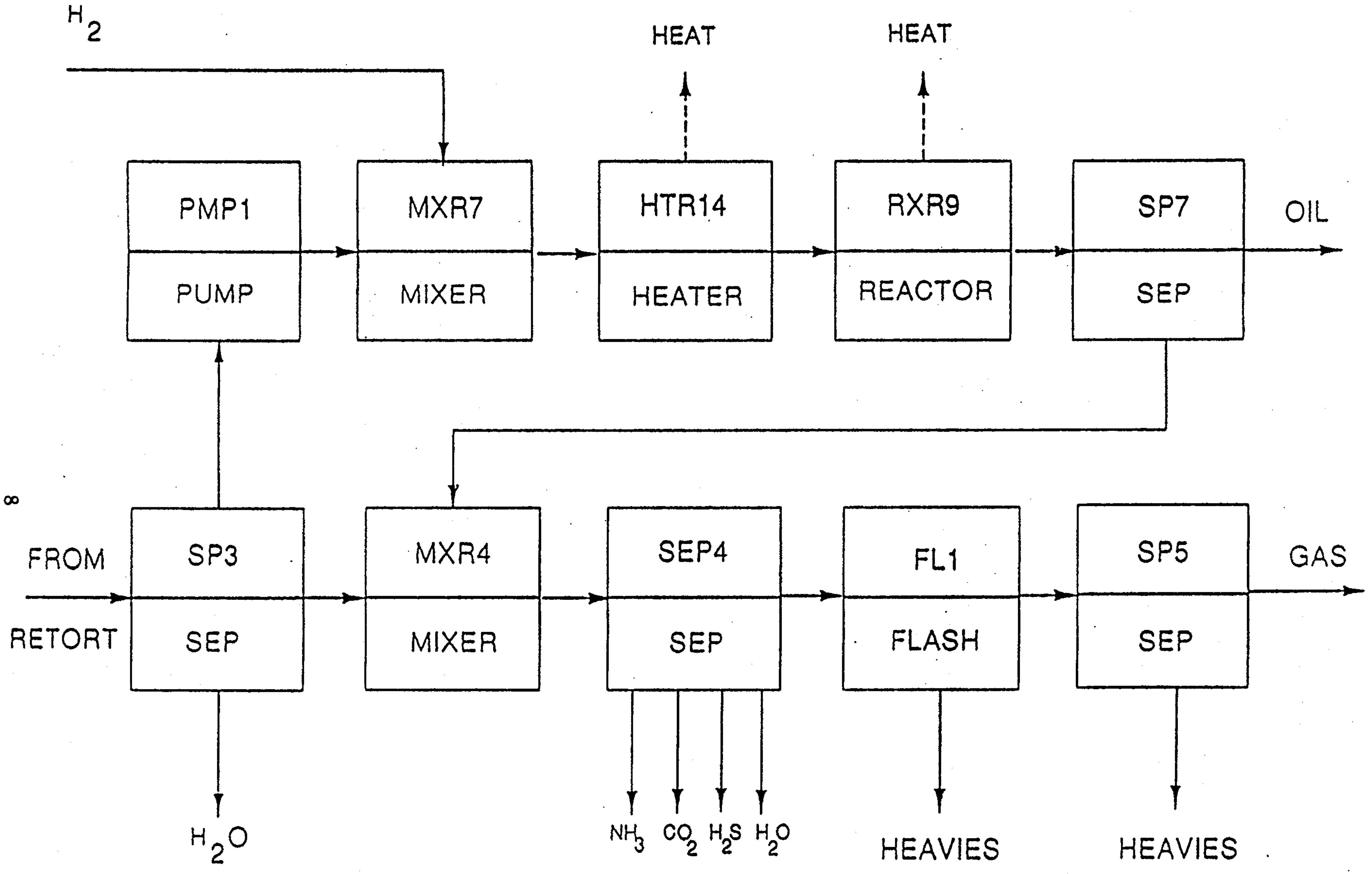


**Fig. 3.** **ASPEN block flow diagram of a gas-liquid separation and clean-up unit.**

the other EOS for one (or more) type of prediction. But no EOS has been demonstrated to be superior to all other EOS for every design situation.

For nonideal liquid mixtures, such as aqueous electrolyte solutions, a type of thermodynamic model called an activity coefficient model is often used to predict liquid phase fugacity coefficients. An example of this type of model is Eq. (2), the famous Chao-Seader Eq. (3),

$$K_i = \nu_i^o \gamma_i / \phi_i^V \quad , \tag{2}$$

where $\nu_i^o$ = the liquid fugacity coefficient of the pure component (this coefficient is different from $\phi_i^L$, which is the liquid fugacity coefficient of component i in the liquid mixture), and

$\gamma_i$ = the liquid phase activity coefficient.

Typically, with an EOS model, both the liquid- and vapor-phase fugacity coefficients are calculated using the same EOS. With the activity coefficient model, the pure component liquid fugacity coefficient model is usually matched with a particular activity coefficient model. In this model, however, one may choose the calculation for the vapor-phase fugacity coefficient from many EOS, independently from the activity coefficient calculation. Thus, many models and many combinations of models are available from which to choose, and in a given situation, one choice is often better than any other.

In addition to the equilibrium K-values, other important thermodynamic functions are also used in ASPEN. For example, the Gibbs free-energy function is related to the equilibrium K-value by Eq. (3), as

$$-\Delta G_i = RT\ln K_i \quad , \tag{3}$$

where $\Delta G_i$ = the Gibbs free energy of component i at a given state relative to the reference state,
$R$ = the gas law constant, and
$T$ = the absolute temperature.

The enthalpy is related to the entropy and the Gibbs free energy by Eq. (4), as

$$\Delta H_i = \Delta G_i + T\Delta S_i \quad , \tag{4}$$

where $\Delta H_i$ = the enthalpy of component i relative to a reference state,
$\Delta S_i$ = the entropy of component i relative to a reference state, and where enthalpy, entropy, and Gibbs free energy are all defined for a given state relative to a reference state.

Correct enthalpy values are necessary for proper energy balances. Therefore, it is important to pick the correct K-value model and to pick thermodynamic models that are consistent with the K-value model. This is necessary for the proper use of ASPEN because the major tasks performed by ASPEN are to obtain energy balances and mass balances (phase compositions) at all points in a chemical plant or refinery. The expert assistant helps the ASPEN user to pick the correct K-value and thermodynamic models.

Figure 4 shows the search tree for the expert assistant. The leaves on the tree are the ASPEN system options, or SYSOPs, which are the models that can be used for predicting K-values. In addition to the steam tables for pure water (SYSOP12) and the ideal gas and liquid model (SYSOPO), ASPEN contains two additional major model types: EOS and activity coefficient models. These major model types are further subdivided. The EOS models vary from the empirical Benedict-type equations to the more theoretical-type van der Waals equations. The activity coefficient models are divided into two groups: the models for which the ASPEN user is not required to supply his own system data and the models for which the user is required to supply the system data. The expert assistant asks the user questions about the situation in question. On the basis of the answers given, the assistant recommends the most appropriate model for that situation. The rules for determining the correct model were obtained from the literature[4-10] and and personal experience. References 4 and 5 are exceptionally good.

Each ASPEN SYSOP has provisions for calculating the following thermodynamic properties for the solid, liquid, and vapor phases in a thermodynamically consistent manner.

Fugacity coefficient
Enthalpy
Gibbs Free Energy
Entropy
Volume (density)
Viscosity (no solid correlation)
Thermal conductivity
Diffusivity (no solid correlation)
Surface tension (liquid correlation only)

Within each SYSOP, the code allows changing of thermodynamic models used to calculate the individual phase properties. We have compiled rules in our expert assistant that will allow the user to choose changes to the vapor and liquid fugacity coefficients. This is the most important correlation because it leads directly to the plant mass balance. In a later paper, it would be worthwhile to explore different correlations for all of the other properties. The ASPEN-supplied alternate thermodynamic models for calculating liquid and vapor fugacity coefficients that may be substituted into the system options are listed in Table I. Also listed in Table are the model numbers, which are related to SYSOP numbers.

Table II lists the possible substitutions for which we have compiled rules.

There are 144 possible combinations of models in Table II that may be suggested by using our compiled rules. Each model will be better than any other in a given situation.

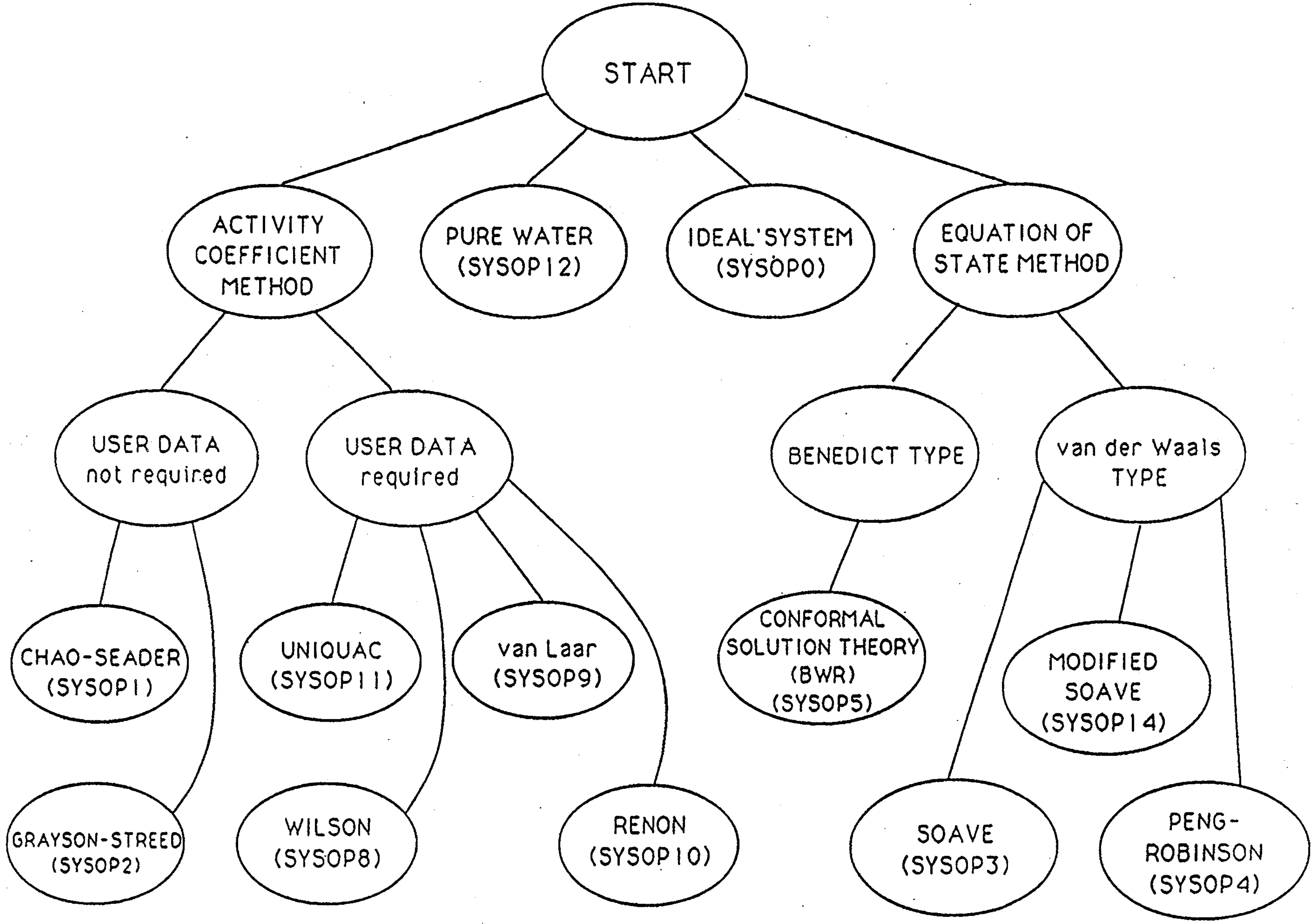


**Fig. 4. Search tree for the plant design expert assistant.**

**TABLE I**

**ASPEN FUGACITY COEFFICIENT MODELS**

| Model Number | Name |
|---|---|
| 2 | Hayden-O'Connell |
| 3 | Redlich-Kwong-Soave |
| 4 | Peng-Robinson |
| 5 | Conformal Solution Theory (BWR) |
| 6 | Perturbed Hard Chain (nonpolar) |
| 7 | Perturbed Hard Chain |
| 14 | Modified Redlich-Kwong-Soave |
| 21 | Scatchard-Hildebrand (temperature dependent) |
| 22 | Wilson (temperature dependent) |
| 23 | van Laar (temperature dependent) |
| 24 | Renon (temperature dependent) |
| 25 | UNIQUAC (temperature dependent) |
| 26 | Electrolytes |

**TABLE II**

**POSSIBLE FUGACITY COEFFICIENT SUBSTITUTIONS FOR THE PROJECT TWO EXPERT SYSTEM**

| SYSOP Number | SYSOP Name | Vapor Fugacity Coefficient Model Number | Liquid Fugacity Coefficient Model Number |
|---|---|---|---|
| 0 | Ideal solutions | | 26 |
| 1 | Chao-Seader | 2,3,4,5,6,7,14 | 21 |
| 2 | Grayson-Streed | 2,3,4,5,6,7,14 | 21 |
| 3 | Soave | 2,6[a],7[a] | 6[a],7[a] |
| 4 | Peng-Robinson | 2,6[a],7[a] | 6[a],7[a] |
| 5 | BWR | 2,6[a],7[a] | 6[a],7[a] |
| 8 | Wilson | 2,3,4,5,6,7,14 | 22,26 |
| 9 | van Laar | 2,3,4,5,6,7,14 | 23,24,26 |
| 10 | Renon (NRTL) | 2,3,4,5,6,7,14 | 26 |
| 11 | UNIQUAC | 2,3,4,5,6,7,14 | 25,26 |
| 12 | Water | | 26 |
| 14 | Modified Soave | 2,6[a],7[a] | 6[a],7[a] |

[a]If models 6 or 7 are used for the vapor phase, the same model must be used for the liquid phase and vice-versa.

After the ASPEN user has chosen the best thermodynamic models for each situation in the plant, he can run the code. Unfortunately, since the temperature, pressures, and compositions of many of the internal flow streams were initially only estimates, the ASPEN computation will change these values. These changes may be great enough to require model changes for subsequent ASPEN iterations. Typically, several ASPEN runs are required to obtain the desired product. Figure 5 characterizes how a design engineer can interactively use an expert system with the ASPEN computer code.

## EXAMPLE--HOW CLIPS WORKS

We have picked a simple example to demonstrate how CLIPS works with this expert system. In the example, the ASPEN user wants to know which model to use for a desalination plant. Although this is a trivial problem for the expert system it demonstrates well how CLIPS solves a problem. With the answers to only three questions, the expert system picks the steam tables (SYSOP12) and suggests using the electrolyte model for the liquid phase fugacity coefficient (model 26 in Table I). A portion of the search space used by the CLIPS expert assistant that contains this example is shown in Fig. 6. The solution to the example problem is shown on the figure in bold face.

Figure 7 shows the dialogue between a user and CLIPS. The reader can follow in Fig. 6, which is a portion of the search space used by CLIPS.

With this system, instead of just picking the best solution for the more complicated problems, we use a scoring technique and supply the user with a list of rated solutions. To do this, simply score the answer to each question asked by CLIPS.

The dialogue with CLIPS begins with the following questions: "*Are the system conditions low pressure and high temperature?*" "*Is your system almost all steam and water?*" and "*Are electrolytes present in the mixture?*" The user responds with *No, Yes, Yes.* In this case, this is enough information to allow CLIPS to make a final decision: "*Use SYSOP12 for steam, the score is 5.0. Use liquid phase fugacity coefficient option number 26 for electrolytes.*" The score of 5.0 is high. In this example, we really have little choice. With most problems, at this point CLIPS would give a list of possible choices with their relative scores. These scores will be more meaningful as one uses the system more often.

## EXPERT ASSISTANT FOR ENHANCED OIL RECOVERY

Some of the reasons to study enhanced oil recovery (EOR) are listed in a 1986 paper by Stosur (11). At the time of the printing of his paper only 27% of the oil ever discovered in the United States had been produced. About 6% more will be produced using existing technology; under current economic conditions. This leaves the remaining 67% as a target for EOR. Currently, only about 6 % of our daily oil production comes from EOR. These figures indicate, even in these times of reduced awareness of an impending energy crisis, that the study of EOR can be rewarding because of the high potential pay-off.

EOR is expensive. It is necessary for engineers to pick the best EOR recovery method for the reservoir in question in order to optimize or even make

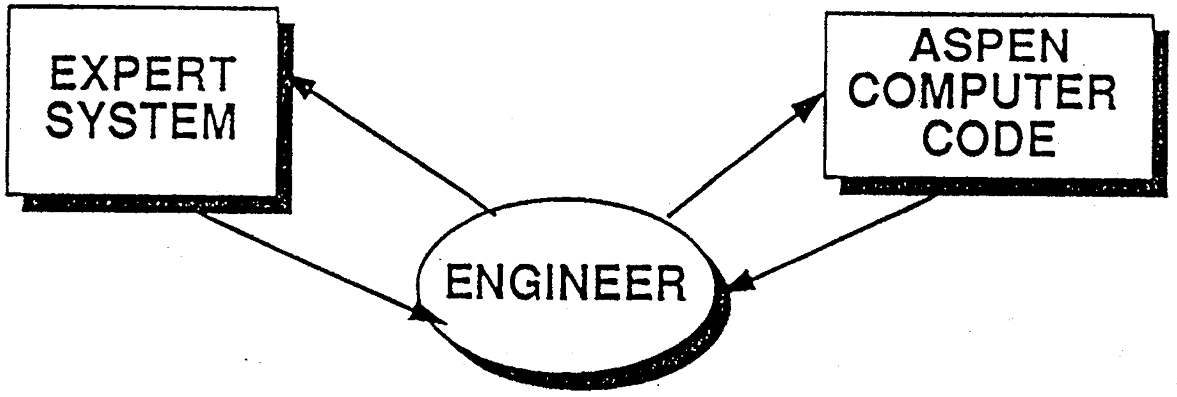


**Fig. 5. The engineers use of the expert assistant with the ASPEN computer code.**

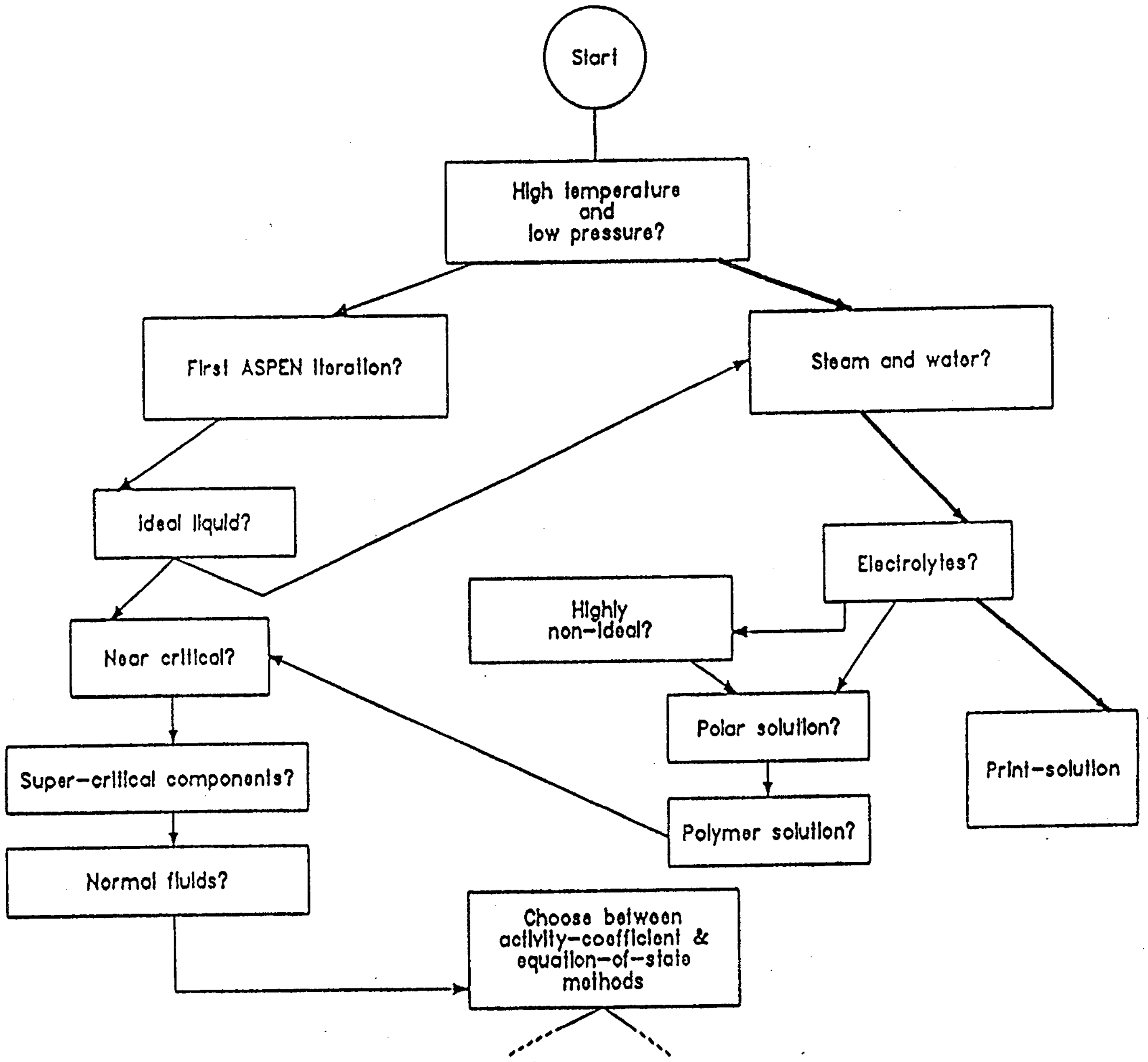


**Fig. 6. A portion of the search space of the plant design expert assistant using CLIPS.**

Are the system conditions, low pressure and high temperature?

(yes/no/"don't know")

**no**

Is your system almost all steam and water ? (yes/no)

**yes**

Are electrolytes present in the mixture?

(yes/no/"don't know")

**yes**

Use SYSOP12 for steam, the score is 5.0

Use liquid-phase fugacity coefficient option number 26 for electrolytes

**Fig. 7. Dialogue with CLIPS.**

profits. The entire screening method itself is expensive. It typically involves many steps. The first step is to consult a technical screening guide. Screening guides consist of a table or several charts that list rules of thumb for picking the proper EOR technique as a function of reservoir and crude oil properties. The candidate techniques are often subjected to laboratory flow studies. Data from these studies are then often used in computer simulations of the reservoir. Finally, a pilot project may be used to demonstrate the viability of the selected technique. Economic evaluations are usually carried out throughout the screening process.

Our expert assistant was developed to replace the table and the graphs used in the technical screening guides or to replace step one in the screening process. It provides essentially the same information as the old table and graph method. but it is more comprehensive than graphs and easier to use than the tables. It provides the user with a weighted list of potential techniques at the end of the run, which is quite hard to do with the tables. The expert assistant is user friendly in that it asks all the questions and leads the user through the first stage of the screening process. It is understood that the final choice of a technique will be based upon economics, but it is obvious that the first screening step is quite important because of the high cost of the entire screening process and the absolute necessity of choosing the most economically optimum EOR technique.

For this study we define EOR as any technique that goes beyond water flooding or gas recycling to increase oil well production. This includes only the injection of material not usually found in the reservoir. The program we have developed relies mainly on the work of Taber and Martin (12) and Goodlet et al. (13) for its rules.

EOR techniques can be divided into four general categories: thermal, gas injection, chemical flooding, and microbial. Thermal techniques usually require reservoirs with fairly high permeability. Steam flooding is a thermal technique that has traditionally been the most used EOR method. In the past it has been applied only for relatively shallow reservoirs containing viscous oils. This is one area where screening criteria are changing because improved injection methods now allow us to go deeper, and because new studies have pointed out that steam temperatures affect other reservoir and oil properties, than just viscosity. The expert system format is a good one to use here because we can easily change the program as the technology changes. On the other hand, gas injection techniques are the opposite extreme of steam flooding. Gas injection techniques tend to work best in deep reservoirs containing light oils. Chemical flooding is usually used with low- to medium-viscosity oils, and reservoir depth is usually not a problem. Microbial techniques are new and primarily experimental at this time, and little is known about them. Chemical flooding is divided into polymer, surfactant-polymer, and alkaline recovery techniques. Gas injection is divided into hydrocarbon, nitrogen and flue gas, and carbon dioxide. Thermal flooding is further subdivided into *in situ* combustion and steam flooding techniques. The microbial category is not subdivided. Figure 8 shows the search tree for the expert system.

Expert system users, are quizzed by the system about their particular problems. Typical questions are about the rock formation type, well depth, reservoir temperature and pressure, reservoir thickness, reservoir permeability, oil saturation, oil gravity, viscosity, and composition. Based on the answers to these questions, the system presents users with an ordered list of preferred techniques for their particular wells.

The solid lines in Fig. 8 represent the most common calculated paths to the solution method. The dotted lines show other possible paths to the same solution method. In other words, with this particular expert system a final solution can be found before the system has zeroed in on the correct EOR category. Our search tree is really a directed graph. The reason for this is probably that this system should really be goal driven or use a backward chaining search instead of the normal forward chaining approach used in CLIPS. Although the CLIPS User's Guide (1) shows how to force CLIPS to do backward chaining we did not use this approach. Instead, we forced it into the forward chaining mode by using a scoring system and following a data-driven approach. It would be an interesting exercise to try a goal-driven approach and see if it is easier to program.

Figure 9 is a portion of the search space used by this expert system. Figure 10 shows a portion of the dialog with CLIPS that is used in solving a simple problem.

CLIPS asks the user the following questions:

*What is the well depth?*

The user answers, *2000 feet*, this is a relatively shallow well.

CLIPS asks *what is the formation type*?

The user answers *Sandstone.*

CLIPS asks questions about payzone thickness, reservoir temperature, permeability, porosity, and oil saturation. The well is at a low temperature corresponding to the relatively shallow depth and has a high permeability. The last two questions shown in Fig. 10 have to do with oil gravity and viscosity. In this case the oil is heavy and quite viscous. The shallow well, high permeability, and heavy high-viscosity oil lead to the conclusion that thermal techniques will work well with this reservoir. The printed list shows the two thermal techniques, steam flooding and *in situ* combustion rate very high on the list to be considered for further study, with scores of 17 and 15, respectively. Two other possibilities are Alkaline flood and microbial drive, with scores of 9 and 7 respectively. To make this expert system work in the forward chaining mode, we decided to ask a lot of questions up front, score the answer to each question, and make decisions based on these scores. The scoring system is empirical and works like this. If the answer to a question presents a condition for which it is impossible for a given method to work, that method will receive a score of -100. If it is very difficult for that method to work at those conditions, a score of -5 is assigned to that method for that question. Other levels of difficulty are assigned -3 and -1 for difficult and just slightly difficult. The method gets a 0 if the answer to the question is not important for that particular method. Scores of 1, 3, and 5 are given for fair,

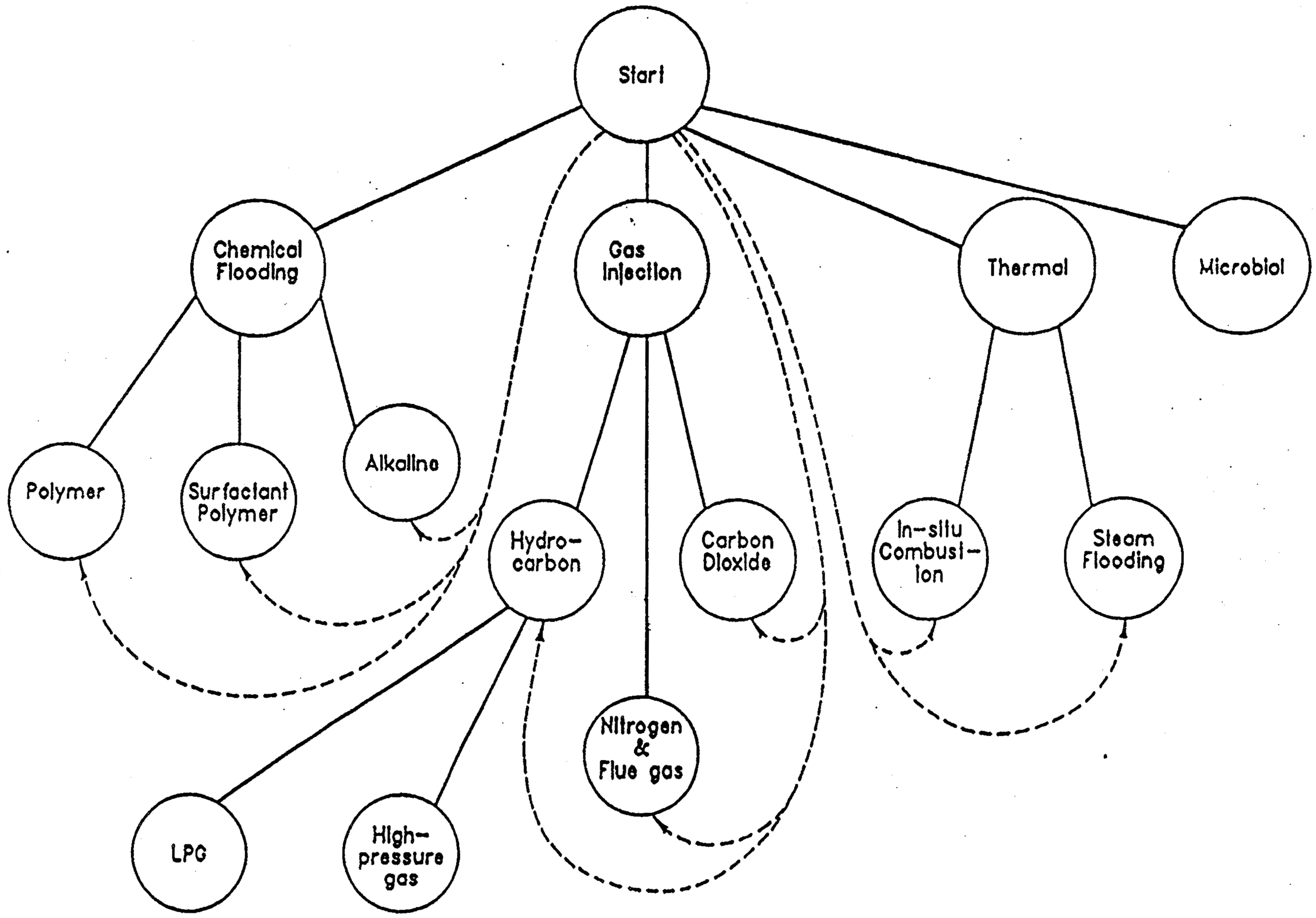


Fig. 8. Search tree for EOR expert assistant.

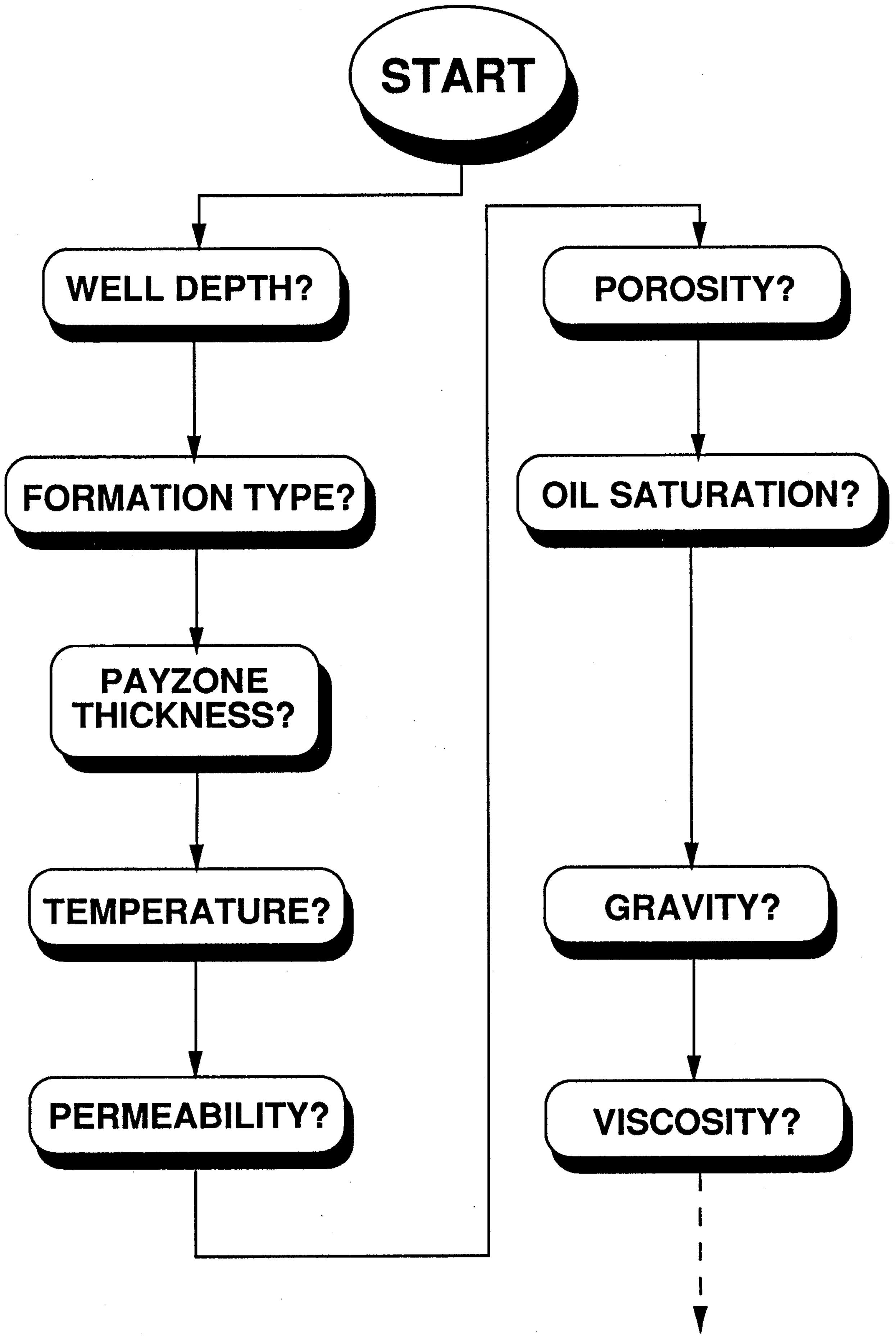


Fig. 9. A portion of the search space for the EOR expert system.

```
What is the well depth ? (feet)

2000

What is the formation type ?
(Sandstone, Carbonate, or Unconsolidated-sand)

Sandstone

What is the payzone thickness ? (feet)

30

What is the reservoir temperature ? (F)

110

What is the reservoir permeability ? (md)

1000

What is the reservoir porosity ? (%)

28

What is the reservoir oil saturation ? (%)

50

What is the oil gravity at 60 F ? (degrees API)

18

What is the oil viscosity at reservoir temperature ? (cp)

500

                         .
                         .
                         .

The following list contains the candidate EOR methods and their relative
scores

Steam flood     score = 17
In-situ combustion  score = 15
Alkaline flood   score = 9
Microbial drive   score = 7
```

Fig. 10. A portion of the dialogue with the EOR expert system.

good, and optimal operating conditions for a given method. As an example, the hydrocarbon gas injection technique and the surfactant-polymer chemical flood technique cannot be used with as high a viscosity as 400 centipoise, so they both score -100 for this question. On the other hand, this is an optimal condition for the thermal techniques steam flooding and *in situ* combustion, both of which score 5's. Microbial drive scores a 1 on this question, and all other methods score a -5.

At the end of the session, the scores are tallied. Any method with a score greater than 5 makes the list of candidates for further consideration. So far, this approach has given realistic results.

Some of the other questions that are asked that are not shown in Figs. 9 and 10 have to do with reservoir pressure, oil composition, salinity, rock wetability, etc.

In this system, because different operating conditions are required as different methods are used, one question can lead to as many as eight rules. The entire expert system currently has about 50 rules.

## EXPERT CONSULTANT AND CONTROL SYSTEM FOR THE SILICON CARBIDE WHISKER GROWTH PROCESS

Silicon carbide whiskers are a very strong material that resemble cat's whiskers. They are produced primarily as a reinforcing material for strengthening ceramic or metallic composite materials. Whiskers can be used as randomly oriented chopped fibers or they can be grown in long lengths, which can be made into yarns and woven. When woven, these fibers create an even more effective directional reinforcement. Although the primary purpose of the whiskers is for the compositing of materials for strength, other uses are also being considered.

Whisker-production is a semi-batch process that is extremely difficult to model mathematically. We are using rules accumulated from many years of trial and error experience to successfully set up and run the process. Two catalyst types, two reactor configurations, and several sets of process conditions are available. When the proper combination of these variables is chosen, the desired result can be attained.

In the past, as long as one person would set up and operate the whisker production runs each time, it was considered adequate for that person to keep the rules in his or her head. But because the whiskers process is now a candidate for technology transfer, we are faced with the problem of how to transfer the expertise to industry without transferring the expert. We therefore designed an expert system to organize and assist in the solution process.

The first phase of the expert system design was to build an expert consultant to provide the user with enough information to correctly set up the run and produce the desired results. The setup information was then incorporated into the rule base to make up the second phase, the control system. The control system corrects perturbations in the process conditions to ensure that the proper corrections can be made and to ensure that the desired product will be obtained at the end of the run.

The SiO (the ingredient used to make silicon carbide) production rate is proportional to the concentration of $SiO_2$ and CO in the brick, and those concentrations are diminishing with time. This is the transient batch portion of the process. As shown in Fig. 11, a mixture of gases containing methane, the carbon source for the silicon carbide production, is forced through the reactor. For many situations, the composition and flow rate of this gas mixture does not vary throughout the length of a production run. This is the steady-state portion of the process. The silicon carbide is formed by Eq. (2).

$$SIO + CH_4 \dashrightarrow SIC + H_2O + H_2 \qquad (2)$$

An idealized growth sequence for a silicon carbide whisker is shown in Fig. 12.

The SIO formed by Reaction 1 must mix with $CH_4$ in the process gas stream. These gases must find their way to a whisker growth surface, as shown in Fig. 11. Flow-visualization studies (14) have shown that this path is tortuous, involving several different modes of mass transfer. Figure 13 shows the modes of mass transport from the SiO generator to the whisker growth surface. Figure 14 shows the steps involved in the overall kinetic process for growth the whiskers.

We have learned a great deal about the silicon carbide whisker growth process and we can now grow them quite well. The foregoing discussion indicates how hard this process is to model with normal mathematical-algorithmic techniques. In fact, we first tried the mathematical modeling approach and found that our models would not adequately predict whisker yield and type from a particular experimental setup. Furthermore, they were inadequate for process control. We had to ask the question "*How can we grow whiskers as well as we do, when we do not understand the physics and chemistry of the process any better than we do?*"

The answer to this question is that our expert operators have learned excellent rules of thumb, through years of trial-and-error experiences, for setting up and running whisker growth experiments. Our next question was "*How do we capture this expertise so that the process can be set up and run by people who are not experienced experts?*" This question is especially important since the technology of the whiskers process has been earmarked for transfer to industry. We would like to be able to transfer the technology without transferring our experts as well. So the answer to this question was to write an expert system or systems to capture this expertise.

## THE EXPERT SYSTEMS

Our goal for this stage of the project was to develop a small PC-based expert system in two phases, as shown in Fig. 15. In Phase I, we developed a whisker growth consultant to help the operator set up a whisker run for the desired whisker type and quantity. The knowledge developed during this phase was entered into a knowledge base that could be used with the expert control system. The expert consultant developed in this phase can be used with the expert control system developed in Phase II, or it can stand alone. The expert control system developed in Phase II uses rules to control our laboratory-scale silicon carbide whisker growth process. The expert control system uses rules and facts

Because our system is a "laboratory-scale" project, we have not added elaborate sensory devices to test for perturbations or upsets. Instead, we use a human sensor: the operator. After observing the controlled variable readings, the operator asks the expert system whether the process is behaving correctly. The expert system responds and suggests which, if any, corrections should be made. The operator then makes the corrections. This system is designed so that automatic controls and sensors can easily be added at a later date or for a larger operation.

The expert system is rule-based, and it is designed to run on a PC. For easy access by the operator, the PC is kept in the laboratory associated with the whisker process equipment.

## THE SILICON CARBIDE WHISKER GROWTH PROCESS

On the laboratory scale, the silicon carbide whisker process is a semi-batch process. That is, part of the process is run in the transient batch mode and part of the process is run in the steady-state continuous flow mode. Figure 11 is a diagram of the silicon carbide whisker reactor. Silicon dioxide bricks impregnated with graphite are placed inside the reactor. After they are heated, these bricks produce silicon monoxide by reaction (1).

$$SiO_2 \quad + \quad C \quad \text{---------->} \quad SiO \quad + \quad CO \qquad (1)$$

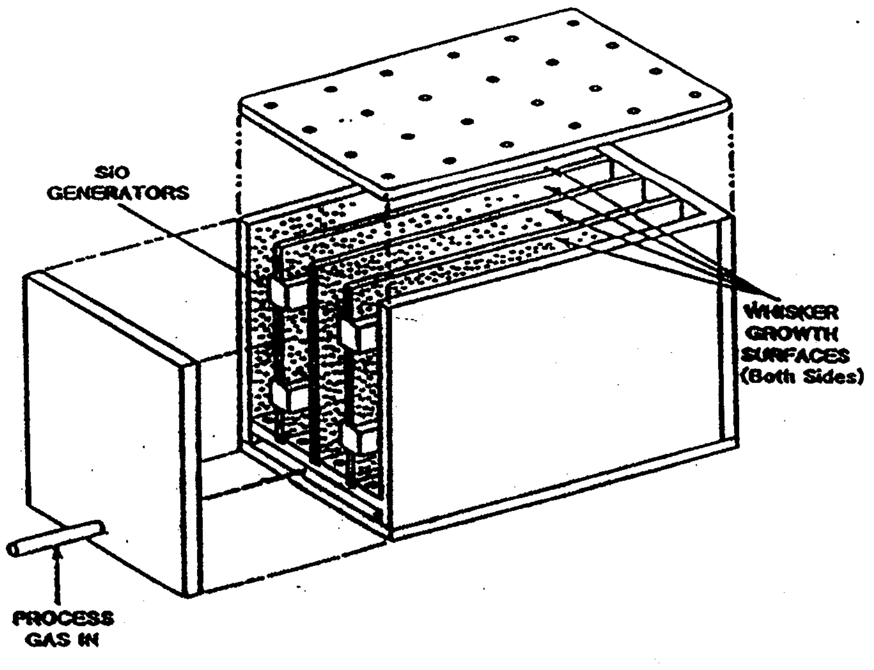


**Fig. 11. The Los Alamos silicon carbide whisker production reactor.**

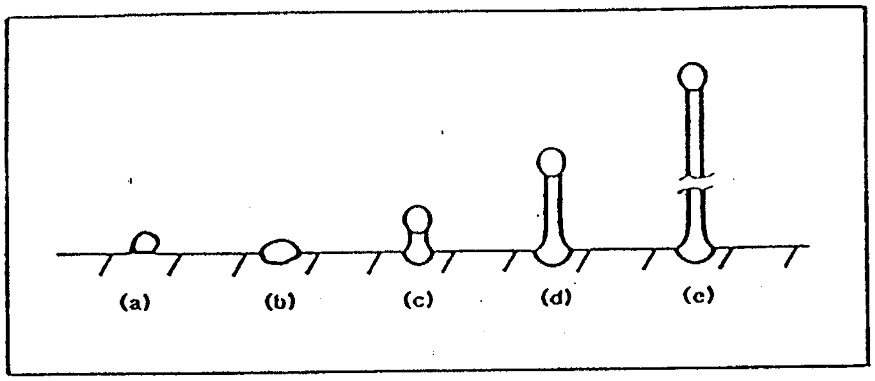


Fig. 12. Idealized growth sequence for the Los Alamos silicon carbide whisker production: (a) metallic catalyst is melted on a substrate, (b) catalyst forms a crater in the substrate, (c) silicon carbide whisker is nucleated, and (d,e) whisker grows away from the substrate with liquid catalyst ball at the tip.

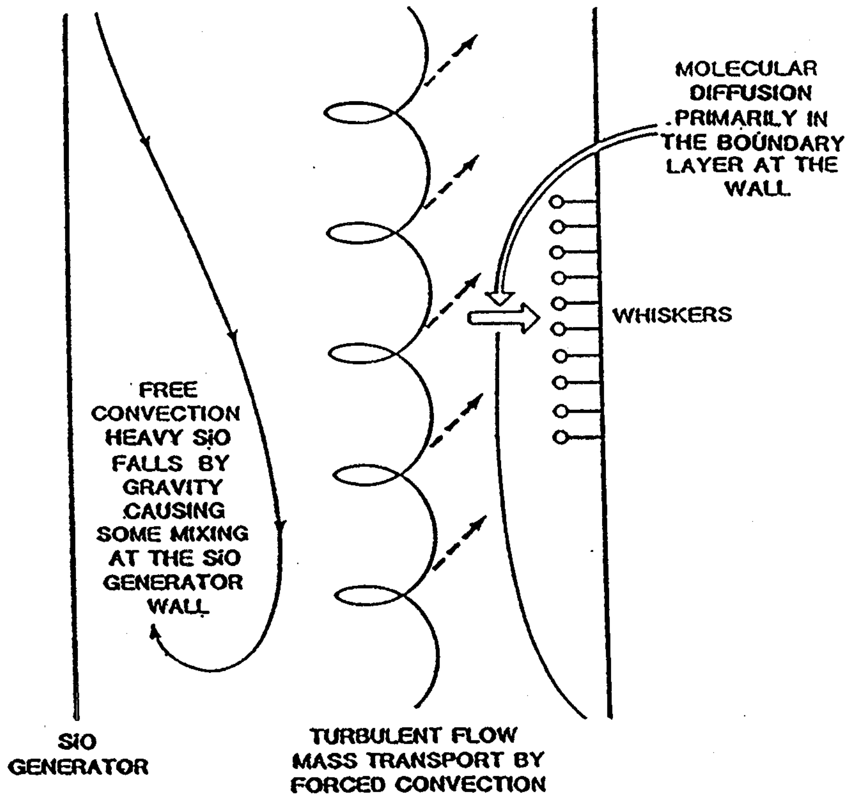

MOLECULAR DIFFUSION PRIMARILY IN THE BOUNDARY LAYER AT THE WALL
WHISKERS
FREE CONVECTION HEAVY SiO FALLS BY GRAVITY CAUSING SOME MIXING AT THE SiO GENERATOR WALL
SiO GENERATOR
TURBULENT FLOW MASS TRANSPORT BY FORCED CONVECTION

Fig. 13. Gas-phase mass transport modes.

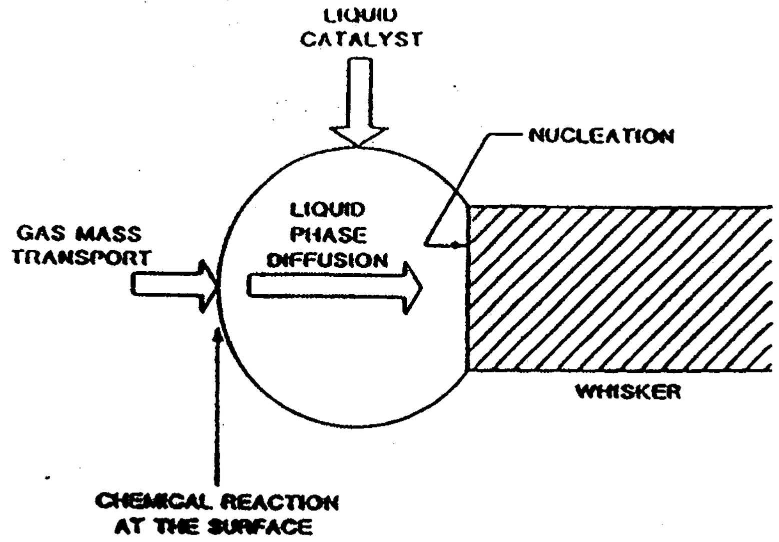


**Fig. 14. Steps in the overall kinetic process for growing silicon carbide whiskers.**

from the knowledge base shown in Fig. 15. Because our process is "laboratory-scale," our controls are manually operated and our sensor output is manually observed. This means that our control system is somewhat limited because it cannot work without an operator interface. This scenario is depicted in Fig. 16. Although this system is adequate for our current process, it would fall short for a full-scale whisker production plant. The full-scale plant scenario is depicted in Fig. 17. This type of of control system is the ultimate goal of future work.

Figure 18, taken from Ref. (15) shows an empirical phase diagram for the growth, and the types of whiskers that can be produced as a function of gas composition. The abscissa represents a change from silicon-rich to carbon-rich gas mixtures. The ordinate represents the silicon monoxide concentration in the gas phase. The properties for the whiskers from categories one through seven, shown at the top of the chart, depend primarily on the whisker diameter. To date, there is some commercial interest in all sections of the chart except areas E and F. These are the combined species and the large bent needles.

For this study we have lumped the whisker types into slightly different groups, based on lengths and diameters. The whisker lengths vary from about 1/8 in. to about 3-1/2 in., which we have divided into three categories: short, medium length, and very long. The whisker diameters vary from 0.1 to 15 in. We have divided them into three groups: small, medium, and large. With short whiskers we are interested only in small diameters. With long and medium length whiskers, we are interested only in those with medium and large diameters.

In addition to developing rules to produce a particular whisker type, we have developed some rules to help maximize the production of those whiskers under various operating constraints. These rules can be divided into three categories: (1) how to obtain the maximum yield with new growth plates, (2) how to obtain the maximum yield with used growth plates, and (3) how to obtain a

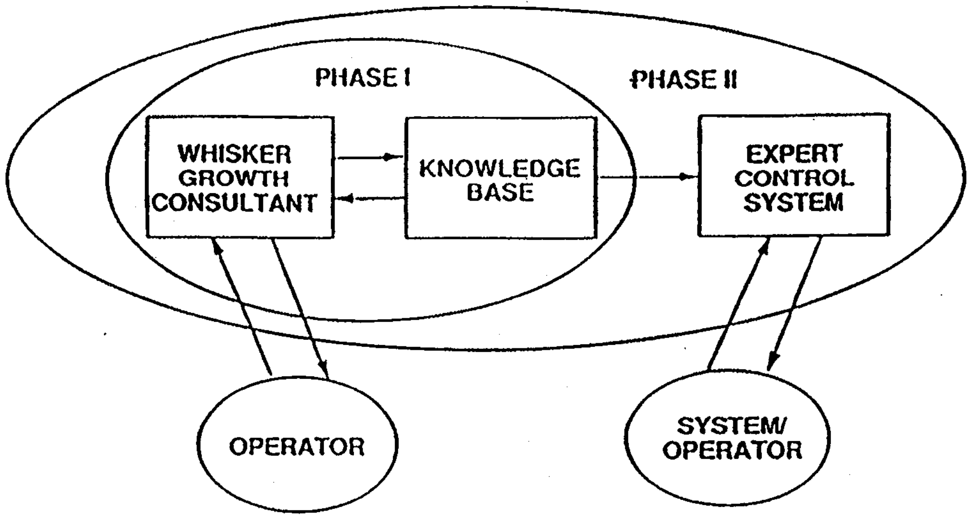


**Fig. 15. The two phases of the expert system design.**

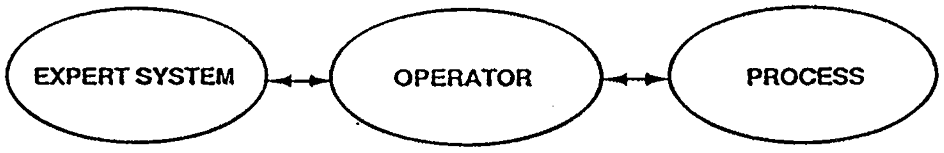


**Fig. 16. Demonstration of the operator interface between the current expert system and the process.**

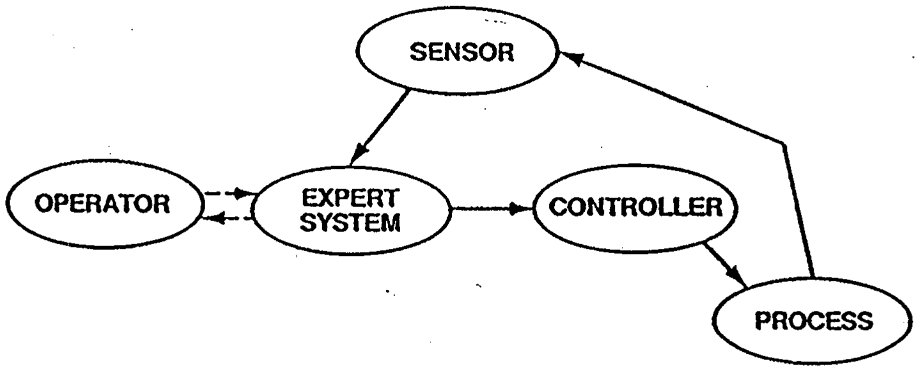


**Fig. 17. Desired expert system/process interface for full-scale plants.**

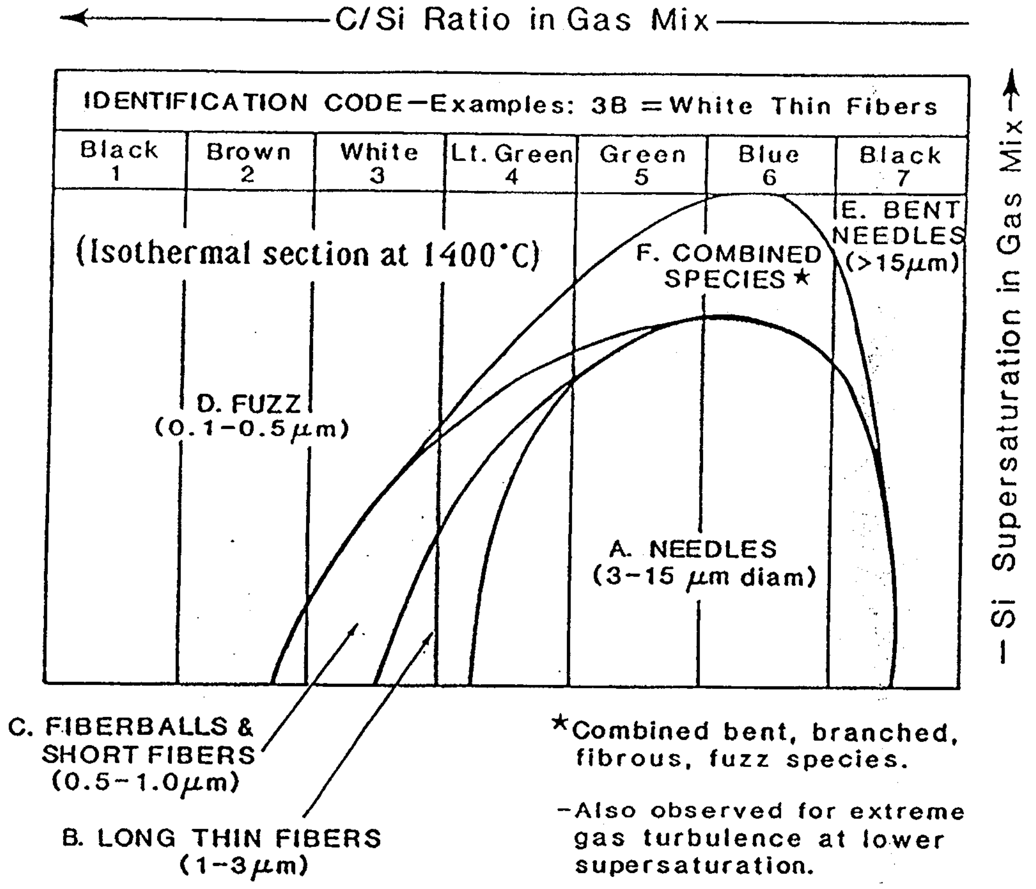

C/Si Ratio in Gas Mix
IDENTIFICATION CODE—Examples: 3B = White Thin Fibers
Black 1
Brown 2
White 3
Lt. Green 4
Green 5
Blue 6
Black 7
(Isothermal section at 1400°C)
E. BENT NEEDLES (>15μm)
F. COMBINED SPECIES★
D. FUZZ (0.1−0.5μm)
A. NEEDLES (3−15 μm diam)
C. FIBERBALLS & SHORT FIBERS (0.5−1.0μm)
B. LONG THIN FIBERS (1−3μm)
★Combined bent, branched, fibrous, fuzz species.
−Also observed for extreme gas turbulence at lower supersaturation.
—Si Supersaturation in Gas Mix

**Fig. 18. Empirical phase diagram for the growth of silicon carbide whiskers.**

maximum yield in a limited run time. We can also have combinations of these three categories. These rules depend upon the type of whisker that we are trying to produce and are included in our whisker growth consultant expert system.

For our laboratory scale operation, we have been primarily concerned with Categories (1) and (2). New growth plates require different gas compositions than used growth plates to produce the same quantity of whiskers. After one run, the new growth plates are coated with silicon carbide, and from then on the silicon carbide participates in the process chemistry. After about four runs the whisker production degrades to the degree that we must replace the plates. We have developed some cost analyses of our process and find it to be labor and material intensive. Replacing growth plates after every run is too expensive, even for a "laboratory-scale" process. Figure 19 shows the general shape of the time-versus-whisker yield curve. Because we are a laboratory operation, our approach is to run the reactor as long as possible to produce the maximum yield.

Figure 19 suggests that a point of diminishing returns is reached before the reactor is shut down. Because this process is a candidate for technology transfer to industry, we have developed some production rules for maximizing yields with shorter run times based on the curve shown in Fig. 19. We assume that an industrial process, even if a batch process similar to ours, would be optimized in a different manner. For example, if several batches were run in one day to the point of diminishing returns, more whiskers would be produced than in our previous maximum production run, yet in the same amount of time.

Figure 20 is a simplified search tree for our whisker growth consultant. The leaves of the tree represent operating conditions that will produce the type of whiskers we want to make. We can change the production from medium length to long whiskers by changing the reactor and catalyst type. To produce short whiskers, we must change the gas composition. Whisker diameter depends primarily on catalyst choice and particle size. At first glance picking the proper production rules for a given run seems straight forward. However, rules obtained from our database have shown that this procedure is more complicated to set up and run optimally than it appears at first observation. For example, gas compositions and temperatures should be different, depending upon the catalyst and the particle size used. The rules for maximizing the whisker yield in a shorter period of time depend upon the intended whisker diameter and length. Long whiskers are not normally produced in shorter run times, and so on.

The expert control system requires the information given to the knowledge base by the operator and the expert consultant. Because, in our system, sensors do not communicate directly with the expert control system, the operator must observe the sensor output, communicate with the control program, and then, if necessary, adjust the system controls manually. The current system has only eleven sensed variables and eight possible control adjustment actions. Three of the sensed variables are temperature, pressure, and total inlet flow. The other eight variables are the inlet and outlet compositions of the four process gases, hydrogen, carbon monoxide, nitrogen, and methane. The eight control adjustment actions are shutdown; adjust temperature; adjust total inlet flow; adjust the flow of the four individual inlet gases (hydrogen, carbon monoxide, nitrogen, and methane); or take no action. The amount of adjustment is highly

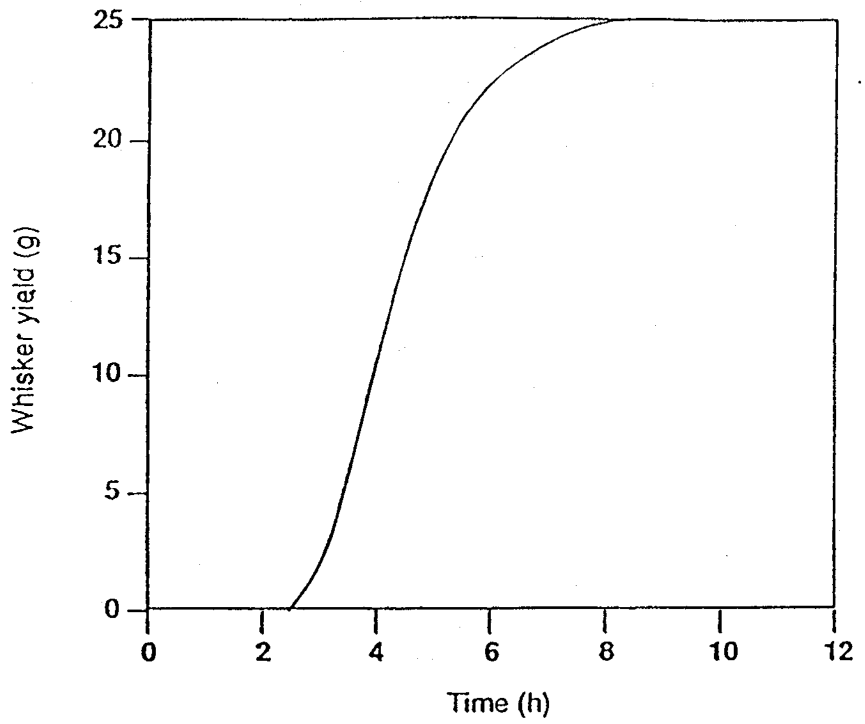


**Fig. 19. A time-vs-yield curve for the silicon carbide whisker growth process.**

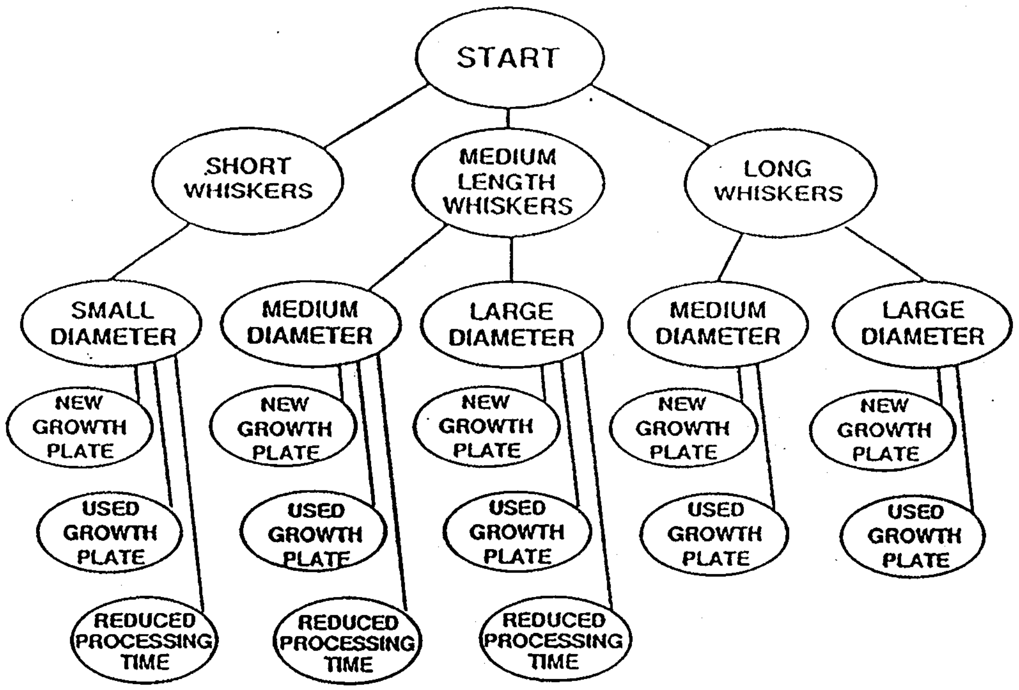


**Fig. 20. A search tree for the whisker growth consultant.**

dependent upon the current reading and the run setup conditions supplied by the expert whisker growth consultant. Figure 21 shows part of the search tree for the expert control system.

Figure 22 is a simplified search space diagram for our expert consultant. The rectangular blocks represent the major decision points in the program. Figure 23 shows the CLIPS dialogue with the whisker growth consultant. It follows the search space shown in Fig. 22. Some of the values given in Fig. 23 are fictitious, because the whisker process falls under the purview of United States Export Control Laws and some information is subject to limited access. The questions in Fig. 23 are questions asked by the expert system with user supplied answers. Note that, at each decision point, the user is given the opportunity to use values other than those recommended. The system was set up this way because, at the end of the consulting session, the correct values must be available for use by the control system.

Figure 24 is a search space diagram for the expert control system. Figure 25 shows the CLIPS dialogue with the expert control system. Note that the control system makes many of its decisions based on parameter values given during the session with the expert consultant.

It should be pointed out that our laboratory scale process is relatively simple. Our operation is not subject to some of the problems of a full-scale plant, such as real-time time-constraint problems caused by the need to search through thousands of rules before an urgent decision can be made. Our expert system works fast enough for the whisker process at this scale. Evidence for this is given in the last line of Fig. 25, "*Reduce the inlet composition... and check with me again in forty minutes.*" In the laboratory environment, gas-stream compositions are only monitored every 40 minutes. This is adequate.

## CONCLUSIONS

The expert systems discussed here have been useful. In each case, we intend to expaned the systems to be even more useful. In the case of the expert assistant used with the ASPEN code, we intend to upgrade the expert system to add more capability. With the expert control system, we plan to fully automate the system and take the operator out of the loop, as shown in Fig. 17. The EOR expert screening assistant requires updates to the methodology as the technology improves. An example of this, as pointed out in the paper, is how the use of steam flooding as an EOR technique is expanding.

The expert system shell is a very good vehicle for this kind of programming, that is, for writing programs that have to be changed and updated regularly; because it is easy to just add rules to the existing code. To date CLIPS has worked very well on these problems, and the PC has been an adequate, if not preferred, platform for the work. We intend to expand our work and explore other approaches to these problems, as discussed in Ref. 16. For example the goal-driven or backward-chaining approach may work better on some of these problems than the data-driven or forward-chaining approach used in CLIPS. As the problem space becomes larger, there may be significant performance differences between these paradigms. CLIPS can be programmed in this mode, but it may be easier to go to other shells if this is the case.

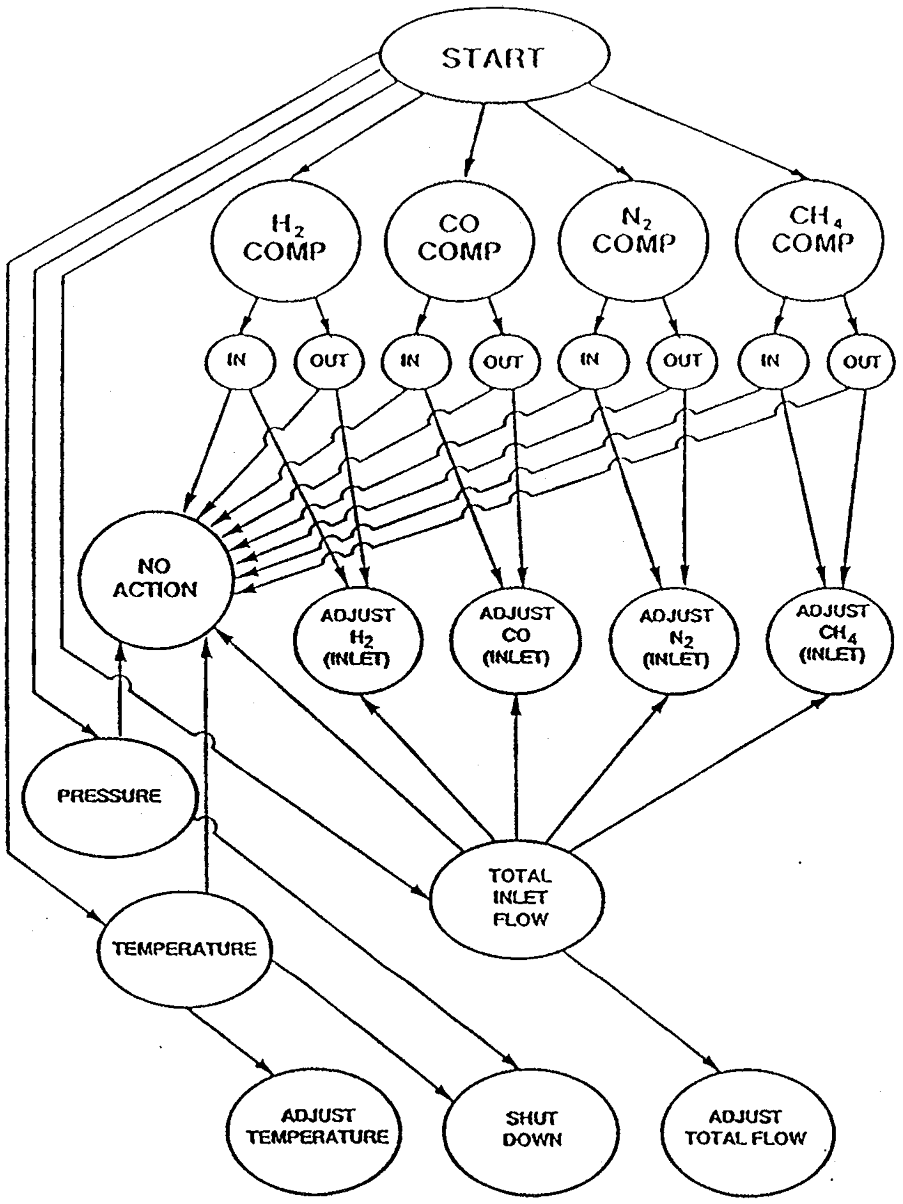


Fig. 21. A search tree for the expert control system.

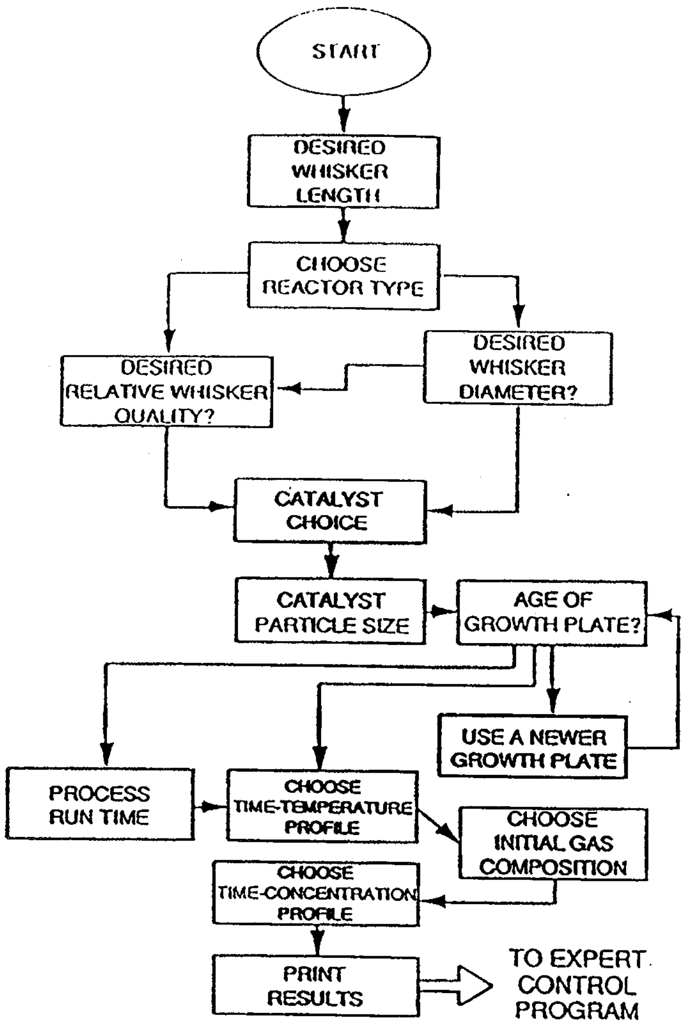

START
DESIRED WHISKER LENGTH
CHOOSE REACTOR TYPE
DESIRED RELATIVE WHISKER QUALITY?
DESIRED WHISKER DIAMETER?
CATALYST CHOICE
CATALYST PARTICLE SIZE
AGE OF GROWTH PLATE?
USE A NEWER GROWTH PLATE
PROCESS RUN TIME
CHOOSE TIME-TEMPERATURE PROFILE
CHOOSE INITIAL GAS COMPOSITION
CHOOSE TIME-CONCENTRATION PROFILE
PRINT RESULTS
TO EXPERT CONTROL PROGRAM

**Fig. 22. The search space diagram for the expert consultant.**

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

3.0

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

B

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

10

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

manganese

We recommend sieve size 20-25, what will you use ?
(25-32, 20-25, or 15-20)

20-25

How many times have you used the reactor growth plate ?
(0 or greater)

0

We recommend time-temperature profile A, which will you use ?
(A, B, C, or D)

A

We recommend that you use the following initial gas composition:
H2 = 80.0 %
CO = 5.5 %
N2 = 14 %
CH4 = 0.5 %

Will you use this ? (yes or no)

yes

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

 (A, B, C, or D)

 A
```

**Fig. 23. Dialogue with the expert consultant.**

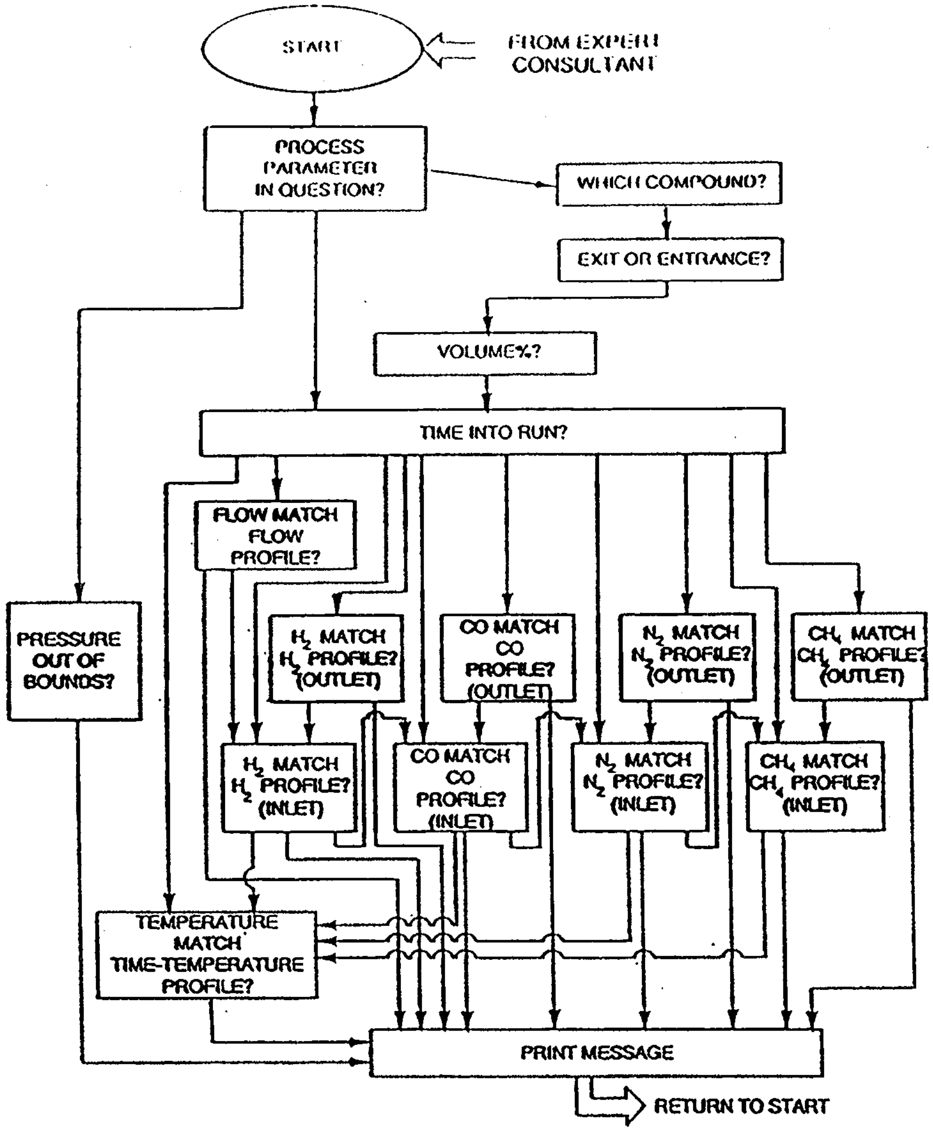

START
FROM EXPERT CONSULTANT
PROCESS PARAMETER IN QUESTION?
WHICH COMPOUND?
EXIT OR ENTRANCE?
VOLUME%?
TIME INTO RUN?
FLOW MATCH FLOW PROFILE?
PRESSURE OUT OF BOUNDS?
H2 MATCH H2 PROFILE? (OUTLET)
CO MATCH CO PROFILE? (OUTLET)
N2 MATCH N2 PROFILE? (OUTLET)
CH4 MATCH CH4 PROFILE? (OUTLET)
H2 MATCH H2 PROFILE? (INLET)
CO MATCH CO PROFILE? (INLET)
N2 MATCH N2 PROFILE? (INLET)
CH4 MATCH CH4 PROFILE? (INLET)
TEMPERATURE MATCH TIME-TEMPERATURE PROFILE?
PRINT MESSAGE
RETURN TO START

**Fig. 24. The search space diagram for the expert control system.**

```
What process parameter do you wish to question ?
  (Temperature/ Pressure/ Total-Flow/ Gas-Composition)

Gas-Composition

Which component ? (H2/ CO/ N2/ CH4)

CO

Exit or Entrance Composition ? (Exit/ Entrance)

Exit

What is the volume percent ?

5

How many minutes since the run began ?

420

The volume percent is too high.

What is the inlet CO volume percent ?

4.6

What is the reactor temperature in degrees centigrade ?

1400

The temperature is OK.

Reduce the inlet composition to 2.9 volume percent and check
with me again in forty minutes.
```

**Fig. 25. Dialogue with the expert control system.**

## REFERENCES

1. J. C. Giarratano, CLIPS User's Guide, Version 4.2 of CLIPS, Artificial Intelligence Section, Lyndon B. Johnson Space Center (June 3, 1988).

2. ASPEN User Manual, Department of Chemical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, December 1981 (ASPEN Project for the USDOE, Contract Numbers E(49-18)-2295, Task No. 9, and DE-AC21-81MC16481).

3. K. C. Chao and J. D. Seader, "A General Correlation of Vapor-Liquid Equilibria in Hydrocarbon Mixtures," *AIChE Journal*, Vol. 7, No. 4 (December 1961) pp. 598-605.

4. R. Banares-Alcantara, "Development of a Consultant for Physical Property Predictions," Master's Thesis, Department of Chemical Engineering, Carnegie Mellon University (1982).

5. R. Banares-Alcantara, A. W. Westerberg, and M. D. Rychener, "Development of an Expert System for Physical Property Predictions," *Computers and Chemical Engineering*, Vol. 9, No. 2 (1985) pp. 127-142.

6. R. F. Wilcox and S. L. White, "Selecting the Proper Model to Simulate Vapor-Liquid Equilibrium," *Chemical Engineering* (October 27, 1986) pp. 141-144.

7. J. M. Prausnitz, Molecular Thermodynamics of Fluid-Phase Equilibria, (Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1960).

8. R. C. Reid, J. M. Prausnitz, and B. E Poling, The Properties of Gases and Liquids, 4th Ed. (McGraw-Hill Book Company, New York, 1987).

9. D. R. Hart, "Liquid Activity Coefficients," Chemical Engineering (November 23, 1987) pp. 131-138.

10. H. G. Grayson and C. W. Streed, "Vapor-Liquid Equilibria for High Temperature, High Pressure Hydrogen-Hydrocarbon Systems," paper presented at the Sixth World Petroleum Conference, Frankfurt, West Germany (June 19-26, 1963).

11. J. J. g. Stosur, "The Potential of Enhanced Oil Recovery," *International Journal of Energy Research*, Vol. 10, 357-370 (1986).

12. J. J. Taber and F. D. Martin, "Technical Screening Guides for Enhanced Recovery of Oil," paper presented at the 58th Annual Society of Petroleum Engineers Technical Conference, San Francisco, California, October 5-8, 1983 (SPE 12069).

13. G. O. Goodlett, M. M. Honarpour, H. B. Carroll, and P. S. Sarathi, "Lab Evaluation Requires Appropriate Techniques--Screening for EOR-I," *Oil and Gas Journal*, June 23, 1986, pp. 47-54.

14. W. J. Parkinson and D. E. Christiansen, "An Experimental Study of the Effect of Reactant Gas Mixing on Silicon Carbide Whisker Growth," Los Alamos National Laboratory report LA-10658-MS (February 1986).

15. P. D. Shalek, D. S. Phillips, D. E. Christiansen, J. D. Katz, W. J. Parkinson, and J. J. Petrovic, "Synthesis and Characterization of VLS-Derived Silicon Carbide Whiskers," Proceedings of the International Conference on Whisker- and Fiber-toughened Ceramics, Oak Ridge, Tennessee, June 7-9, 1988.

16. G. F. Luger and W. A. Stubblefield, Artificial Intelligence and the Design of Expert System (The Benjamin/Cummings Publishing Company, Inc., Redwood City, California) 1989.

# BUILDING ENGINEERING EXPERT SYSTEMS IN CLIPS

Ken Porter
Harris Corporation
PO Box 98000, Mailstop R3/7257
Melbourne, FL 32902

## INTRODUCTION

This paper is intended for CLIPS developers with a working knowledge of expert systems and the CLIPS syntax. It discusses Rete pattern matching and rule-fact interaction, explains several development and debug techniques, and gives advice on compiling CLIPS and knowledge bases. The techniques apply to CLIPS versions 4.2 and 4.3, especially in the PC/DOS environment. Two examples developed by the author are analyzed and compared.

## TYPICAL DATA DRIVEN APPLICATIONS

Many professionals use computers to process information and make decisions, often spending much time at low-level data processing tasks. The "information density" of data is proportional to how much effort was spent refining, summarizing, and analyzing the raw data with respect to a given purpose. Environments in which CLIPS can be effective allow a rule base to augment conventional software using previously automated data. Professionals who are experts at their jobs may already use batch programs or "macros" to customize and link existing information processing tools. Increasing the degree of automation for data processing activities increases these experts' productivity.

A potential CLIPS engineering application may be identified as a data processing task requiring experience more than creative thinking, and where computer automation already assists in some part of the task. The expert performs routine steps to handle 60 to 80 percent of the data with a computer. The remaining tasks require decisions based on experience or professional skills, which is probably why an expert is doing the job in the first place.

Business applications for CLIPS expert systems may be found as both expert "assistant" programs used by the expert to process some already automated information, and also as expert "advisor" systems which are used by non-experts to increase their level of performance. The CLIPS forward chaining inference engine is suited for assistant applications which process data streams. CLIPS was used to develop an embedded application for Wall Street in which an expert securities monitoring system ran alongside conventional securities trader displays.[1] CLIPS is less suited for advisor applications, which are typically problems of classification or diagnosis.

An important factor in selecting an expert system shell for a problem is how well the shell features support the problem environment. Many simpler backwards chaining shells have better interactive user interface support than CLIPS, making them more effective for interactive advisor applications. However, construction of a basic goal structure and rules for combining evidence are not too difficult in CLIPS. By asserting goals as control facts and creating rule conditions, or left-hand side (LHS) patterns, which match on the goals, the CLIPS forward chaining inference engine can be made to emulate backwards chaining behavior.[2] Note that while forward chaining shells can be made to run backwards, the reverse is not generally true.

## TWO ENGINEERING EXPERT SYSTEMS

Two examples will help illustrate how to build information processing applications in CLIPS. Both examples are from the engineering discipline of automatic testing. The first is a small engineering tool, while the second example pushes the limits of the common PC.

**TRACER.** The first expert system automates the translation of wiring information. The objective is to create an end-to-end wire list from a somewhat random, CAD-generated file of wire path segments. Electrical circuits for a chassis include a network of connectors which is already documented in a CAD application. The number of connectors in a path and their sequence is arbitrary. Some paths branch out and others converge, forming an interconnected network with several large common nodes. The CAD data consists of wire segments referenced by their endpoint connector names and pin numbers. The task is to trace through a list of wire segments in arbitrary order, determine their interconnection structure, and print it in several formats. The programmer working on this problem in a conventional language encountered difficulties tracing through the connector network due to non-uniformly deep nesting and recursion, which suggested a data-driven, rule based solution.

The expert system which solves this problem is comprised of three separate rule bases, TRACER, LISTER, and NOD2ATL (summarized in Table 1). TRACER processes the wiring information by tracing depth-first from a known endpoint. The wire segments are read from an ASCII file and asserted as facts with fields for connector names and pin numbers. Wire path tracing employs heuristic knowledge about how to identify which connectors and pins plug together. Terminal blocks are represented like connectors but can be recognized by features in their names. When the path traces to a terminal block or other endpoint, that node is specially labeled. A second trace from the terminal blocks connects them together. The resulting node list is saved as an ASCII file. The two other rule bases read the node list and format the information. LISTER prints connector names and paths as a conventional single-ended wire list for engineers to use in troubleshooting the wiring. NOD2ATL generates a file header of the path data which is required by the ATLAS programming language. The functionality was split between three rule bases to make it easy to modify either wire listing format without affecting the basic path tracing functionality.

**HEIRS.** The second example expert system helps engineers design the wiring for an interface test adapter (ITA), used in automatic circuit card testing to connect the circuit card to the tester. Testing a circuit card requires electrical power, various kinds of stimuli, and measurements at certain connector pins. The automatic tester has power supplies, digital meters, signal generators, and other programmable instruments connected to a large patch panel. Electrical design for an ITA consists mostly of mapping the circuit card connector pins to the right tester patch panel pins. But in some cases, conflicting signal requirements result from two or more tests being performed at the same circuit card pin (at different steps in the tester's program). For instance, it may damage an ohmmeter to leave it connected when power is applied to a circuit. Most of these cases can be solved by adding a tester-controlled relay to switch between the conflicting signals.

The design process and domain heuristics of an experienced test engineer were used to build a knowledge base which could handle most of the signal requirements correctly and automatically. This expert system, the Harris Expert ITA Routing System (HEIRS), contained 77 rules and several hundred facts. HEIRS was intended to be a proof-of-concept prototype, but user evaluation showed this prototype could save an ITA design engineer about 25% of his design time. Extending the system's knowledge base to cover more signal types and resolve more problem cases proved to be easy. Table 2 contains a synopsis of the HEIRS expert system.

## USING CLIPS EFFECTIVELY

**Heuristic versus deterministic programming.** The rich rule syntax of CLIPS permits both heuristic and deterministic programming. It is important to decide which method best suits the particular purpose of each rule. Heuristics may define domain knowledge as relationships between symbols. This knowledge is often implemented as a set of simple rules by translating the symbol relationships into rule conditions. Each rule contributes its knowledge based on changes to the facts it recognizes. This can simulate the thought process of human experts, and is one of the primary motivations for developing a rule-based system. However, non-deterministic programming methods are not always the best. A deterministic task programmed as multiple rules may not execute properly if unanticipated events occur.

Tasks such as reading records and parsing data from a file can often be performed by a routine written as the conclusion, or right-hand side (RHS), of a single rule. RHS commands can be combined to create a conventional program module. The CLIPS syntax for RHS commands includes if...then structures, while loops, math expressions, fact assertion and retraction, and basic string and multi-field variable manipulations. External programs can be called and data passed. Environmental commands may be used to determine available memory, read and write files, and communicate with the operating system. When building such program modules, make sure the RHS routine is self-sufficient. Try to put the entire routine in one rule to ensure that execution will not be interrupted.

**Pattern matching.** Pattern matching and fact management functions can exhibit poor performance if you don't keep pattern matching efficiency in mind when developing the knowledge base. Just as a good screwdriver makes a poor hammer, you must use the CLIPS rule properties and symbolic functions to solve the problem, and not try to implement a conventional deterministic solution.

CLIPS uses a Rete network of compiled rule conditions to monitor the matching of rule conditions and facts. The Rete network improves the speed of rule-fact matching, but as a consequence, new rules cannot be accommodated once CLIPS is executing. CLIPS activates rules by posting them on an agenda when all of their LHS conditions are met. The CLIPS matching process cycles through the knowledge base checking all the facts against each rule condition. As new facts are asserted, each rule may be unaffected, partially instanciated, or posted and activated during each cycle, depending on how many of the rule's conditions are met and also their order in the LHS. Understanding the effects that condition order has on rule instanciation is the key to effective use of the Rete pattern matching algorithm.

To get a feeling for how the RETE network works, picture a Japanese pachinko game. In this centuries-old form of pinball, little balls are shot up onto an inclined board and sometimes catch in traps on the board as they roll down. Compiled rule conditions are like the traps, pre-arranged in a network. Facts are like the little balls, which catch in the traps if they match the pattern of the rule condition. Visualizing rules as fixed structures which catch the dynamic facts may help in understanding the synergy of rule-fact interaction in CLIPS.

Arrange each rule's LHS patterns to minimize partial matches, which consume CPU time and memory. Of course, put the pattern least likely to match first. However, be sure to consider facts as well as rules when deciding how many rule-fact combinations may match up. Multi-field variables (e.g., $?x) must handle zero or more fields and are specially handled by CLIPS in a way that uses more memory than fixed patterns. CLIPS also creates temporary memory records of rule-fact matches when it needs to transfer variable bindings from an initial match to subsequent conditions. Consequently, executing a knowledge base with many variable pattern matches can use a lot of memory. Because CLIPS implements multiple matches in an outer-join or N-by-M manner, a knowledge base containing many multi-field facts and rules with several multi-field variables in the same LHS pattern can result in extreme execution memory requirements.

## DEVELOPMENT TECHNIQUES

**Interfacing.** CLIPS has only a basic capability for I/O handling of ASCII strings, but this still permits its powerful inference capabilities to be applied to many information processing applications in engineering and business environments. Although no direct interfaces to common data processing applications such as dBase or Lotus are provided, most commercial packages interface with standard text files to which CLIPS can easily read and write. For example, the NOD2ATL rule base transforms a nodal structure into ready-to-compile source for ATLAS, a higher-order tester control language similar to FORTRAN in syntax. Files and command scripts may be used to interface with spreadsheets, databases, CAD tools, and even accounting systems.

CLIPS permits direct calls to user-written routines in C, making virtually any desired extension possible. A user can define external functions for use on both the LHS and RHS of rules. The growing popularity of C, along with the ever-increasing power of available hardware platforms, makes this CLIPS feature particularly useful in a computer-oriented job environment such as engineering or data processing.

The documentation supplied with CLIPS is very well written, making it easy to use and modify. The Advanced Programming Guide has annotated source listings for interfaces to Ada and FORTRAN.

**Phase state control.** Many processes may be divided into simpler sub-tasks, a problem domain feature which can be used to structure a large knowledge base. By segregating the rules into functional sets and assigning each a phase control fact, unwanted LHS matching will be eliminated at the first level of search. However, masking rules to be active only in a certain phase conflicts with the opportunistic nature of rule based systems. Rule base segregation should be used only when the problem is readily separable into sub-tasks with little in common.

To implement phase controls in a knowledge base, analyze the problem domain to identify sub-tasks which are mostly independent. The creation of intermediate forms of information often identifies suitable sub-task boundaries. Partition the knowledge base into sub-task subsets which transform information from one form to the next, asserting and retracting temporary facts in the knowledge base. Assign each rule subset a unique phase control fact and make it the first pattern in each rule of that subset. Finally, to control the phase state progression, make a very low salience (low priority) rule that matches any phase control fact, but just the phase control fact. When the fact base is exhausted by the current phase, this rule retracts the old control fact and asserts the next control fact. The new control fact enables more new rules. The process cannot stall because the phase change rule is always matched. You don't really have to retract the old control fact unless you want to be sure those rules cannot fire again. In fact, it is possible to build a knowledge base that not only progresses linearly but also iterates or recurses through many phase states using this phase control fact technique.

In the HEIRS expert system, rules are split into 7 phases. The phase control technique is implemented as one large rule and seven phase name strings in an initial "deffact" statement, as shown in Figures 1 and 2. One phase is further sub-divided to reduce a peak execution memory requirement, where three rules have multi-field variable patterns that match nearly every field in a large group of multi-field facts.

Asserting the phase names as strings in an initial fact makes it easy to change their execution order by re-arranging their name strings. Note that the "phase-state 0" fact controls where execution starts, at phase 1 in this example. The end of the phase control rule, shown in Figure 2, is where the phase state number is incremented. This keeps the process going. The phase control rule's LHS, with a salience of -1000, binds the control fact and tests for a error indication. The RHS is a series of "if" clauses which assert facts, open and close files, and perform other actions as needed at the start of each phase (actually, at the end of the previous phase). Several other rules are used to test for certain kinds of errors. One rule has no explicit LHS and a salience lower than any other rule. It detects if the knowledge base has stalled, indicating a problem with the phase control structure.

**Knowledge base segregation.** If it is possible to split the problem into two or more completely independent sub-tasks, build a separate knowledge base for each sub-task. The CLIPS save and load commands may be used to create intermediate fact files. Using "bload" and "bsave" is faster, and has the added feature that the binary files are not directly readable. Control for loading and executing successive rule bases can be implemented through operating system control scripts (batch files in DOS). The TRACER/LISTER wire list generator was able to process more facts in a memory-limited DOS environment because the tracing task was separable from the listing task. Also, developing several smaller rule sets is easier than one big one. Creating and debugging a set of rules to generate a new output format was simplified by not having to consider possible interactions with rules which performed the tracing function.

**Fact management.** Get rid of information as soon as it is no longer needed by the knowledge base. This minimizes the number of facts and thus the possibilities for unwanted matching. Using a high salience rule, bind the fact to be deleted in a LHS pattern and use the RHS to perform any appropriate final action (such as writing to a file) and retract the fact. Use enough LHS patterns and tests to make certain this high salience rule matches only the facts ready to be removed.

Attaching a process state vector "tag" to each fact when it enters the knowledge base can simplify fact management. You should alter the tag when significant fact modifications are performed. This can also be very helpful for tracing and analyzing execution problems in the non-deterministic CLIPS environment.

**Monitoring rule base execution.** Once a rule base for an embedded application is running properly, there is often little visibility of its internal functions. CLIPS may be processing hundreds of pieces of information per second without requiring or attracting user attention, acting as a knowledge based "invisible" assistant. While unobtrusive execution is beneficial in many situations, illustrating system activity may be important for both debug and demonstration purposes.

An animated narrative of the system's behavior can be created by using simple format statements which print to the screen when rules are fired. A text-based display of the executing system can be built using the "fprintout" and "format" commands. Four text colors are available with the DOS window interface version of CLIPS by using the I/O router's "wtrace" for white, "wdialog" for tan, "werror" for green, and "wagenda", "wdisplay" or "stdout" for cyan (light blue). Note that tan is a difficult color for many color monitors, and could result in any hue from pink or orange to brown.

For rules which make significant changes to the fact base, put descriptive comments in the fprintout commands. It helps to quote the key fact associated with the rule activation. Such RHS actions can be used initially for debugging the rule base, then dressed up later to make an execution monitoring display for the expert system user.

## DEBUG TECHNIQUES

Although CLIPS comes with only a few debugging aids, they can be effective when used in the right way. These include breakpoints, the "watch" and "dribble" functions, batch commands, and the Cross Reference, Style and Verification utility (CRSV).

**Watching execution.** CLIPS does not have a fancy graphical developers interface, so watching a knowledge base during execution requires planning. The watch functions report on changes to the fact base, rule activations, and rule firings (and compiling in version 4.3) by sending text to the screen. It is best to use all three watch functions in observing the behavior of new rules, but one event can often send too much information scrolling off the screen. To prevent this, turn on the "dribble" function at the start of execution. Dump the facts to the screen whenever unexpected events occur and at the end of execution. This thoroughly documents the system state in the dribble file for later analysis.

Breakpoints in CLIPS will halt execution immediately prior to firing the specified rule. To use a breakpoint repeatedly or several breakpoints in combination, create a text file of CLIPS commands and invoke CLIPS with the "-f" batch file option. A typical debug batch file might look like:

```
(dribble-on "debug.log")     ; save results
(load "main.clp")            ; load primary rule file
(reset)
(run 2)                      ; for initialization as required
(set-break <rule-name>)      ; area of interest
(facts)                      ; system state before
(watch all)                  ; turn on trace information
(run 20)                     ; limit firing
(facts)                      ; system state after
```

When the batch file finishes, CLIPS will remain in the interactive mode until you exit, allowing further manual stepping and other debug actions.

**Conserving memory.** CLIPS normally reserves memory to store the optional quoted comment of each rule so it can perform a "pretty print" listing if needed. The "conserve-mem" command allows you to turn off this feature, saving about 400 to 1000 bytes of memory per rule (see Table 4).

If a knowledge base will compile and begin executing but then runs out of memory, it may be making too many variable matches. CLIPS can dynamically allocate memory, and will attempt to re-use previously allocated memory when an allocation attempts fail. This will cause CLIPS to slow down considerably and write "**** DEALLOCATING MEMORY ****" messages to the screen, but it may continue to run if a few Kbytes more memory is enough. To avoid run-time memory problems, restrict the use of LHS variables to a minimum and organize the LHS so that conditions with variables come after those with none.

**Using CRSV.** The CRSV utility may be useful for checking the consistency and style of a knowledge base, even when separated into several files. CRSV is helpful mainly for large knowledge bases in their final stage of development. Because it assumes facts to be written in an attribute-value format, it is most useful for to knowledge bases which adhere to that paradigm. The version supplied with CLIPS 4.3 is more powerful than the 4.2 version, but neither are well suited to analyze knowledge represented in object-attribute-value formats.

To use the CRSV utility, first invoke it with the create (-c) option to parse the rules and generate a file containing descriptions of LHS patterns, fact structures, and external function calls. Edit these descriptions using any editor and then use them to filter your rule base for unusual, and often erroneous, rule structures and facts using the verify (-d) option. Because CRSV comparison keys on the first field in a fact, facts which begin with a variable are problematic. Be careful when using the create option -- the first file name after the create option is used to store the newly created descriptions and if a file by that name already exists it will be overwritten. Forgetting this and putting the name of the rule file immediately after the create option will cause CRSV to destroy the rule file you meant to check.

For a large knowledge base, it is best to re-direct the CRSV output to an external file when using the verbose (-v) or cross-reference (-x) options. These options can create literally dozens of pages of output for a non-trivial sized knowledge base. Conversely, the summary (-r) and style (-s) options produce much less output. Note that the style option is the only one turned on by default, so specifying this option turns off style warnings and code analysis.

A new feature of CLIPS 4.3 is the ability to generate a trace file during execution and analyze it with CRSV afterwards. This should be helpful for knowledge bases which employ string manipulations of facts and other run-time gyrations that CRSV misses when analyzing the rule and fact patterns in the static knowledge base.

## COMPILING

CLIPS is supplied with complete source code and excellent documentation, making it one of the most adaptable expert system shells available. The C source follows the ANSI standard, so compiling it is trouble-free for most compilers and hardware. You may want to compile CLIPS to customize its features. Merging CLIPS with a knowledge base into one executable can provide additional benefits.

**Customizing CLIPS.** For the most part, the CLIPS DOS version 4.2 from COSMIC is compiled just right for learning the CLIPS syntax and running small applications. In version 4.3, COSMIC took a different approach and supplied a minimal DOS configuration. However, tailoring the CLIPS executable to alter its features is simple. You don't need to know much about the C language as long as you understand basic programming concepts. You can make smaller compiled expert systems or more powerful, faster development versions simply by editing the master setup file and re-compiling the CLIPS source files with any ANSI C compiler.

The COSMIC CLIPS 4.2 executable includes an integrated EMACS editor with incremental rule compiling. This version runs well, but has a bug which makes the help system practically useless. The COSMIC CLIPS 4.3 executable fixes the help problem, but does not have the editor. The COSMIC 4.3 executable initially takes about 70K less memory, but uses twice as much memory for loading each rule as the 4.2 executable. Also, the COSMIC 4.3 executable seems to have a stack overflow problem when loading a large number of facts. Re-compiling CLIPS allows you to tune it for best performance in a particular application.

The header file SETUP.H contains 24 compiler flags (18 in version 4.2) that may be edited to easily tailor CLIPS. Tailoring instructions are included in the comments of SETUP.H, and the compiler flags are described well in the Advanced Programming Guide[3]. There are several features worth changing for a developer's version, and some just waste memory in a non-development environment. To help keep track of the more than 40 source files, CLIPS is distributed with compiler support for Turbo C 1.5 or 2.0, Microsoft C 5.1, Zortech C 1.07, Lattice C 3.0, and Lightspeed C on the Macintosh.

For a developer's version, you may want to turn on the flag for CLP_TIME and possibly BLOCK_MEMORY or BLOAD. The CLP_TIME function is very helpful for tuning CLIPS execution speed because it prints the execution time along with the number of rules fired whenever CLIPS stops. The BLOCK_MEMORY flag helps if memory is not a problem but the knowledge base needs to run a bit faster. The BLOAD flag may help to interface separate knowledge bases by saving and loading facts quickly. Whatever you do, be sure to change the opening message CLIPS displays when it starts up to provide an indication of which version is running. The familiar opening text "CLIPS (V4.x0 mm/dd/yy)" in the file MAIN.C should be replaced by a terse description of the flags changed, other alterations such as stack size, and perhaps the C compiler used. This is essential for version 4.2, which lacks the "options" command of version 4.3 so you cannot recover the configuration information later.

For a bare-bones executable version, you definitely want to turn off the CLP_EDIT, CLP_HELP, and CLP_TEXTPRO flags to save about 25 Kbytes of memory. Breakpoints are not useful after the rules have been debugged, so turn them off too. Other CLIPS features which you can eliminate if you don't use them include: extended math (anything more than add, subtract, multiply, and divide), any unused I/O functions, the "deffacts" and "deftemplates" commands, and the string and multi-field functions. Together, these can save up to 100K bytes of memory. Use the block memory flag to speed up memory functions if you have memory to spare.

Of the two popular compilers Microsoft C and Borland's Turbo C, it is probably best to use Turbo C if you have little or no C language experience. Turbo C is one of the more user-friendly C compiler on the market, with pull-down menus and on-line help. Be sure to set the large memory model flag from the setup menu. There may be some problems in using Turbo C for demanding applications -- too many facts can cause Turbo C versions of CLIPS to overwrite memory and possibly crash the system.

Microsoft C (version 5.1) is more flexible and possibly better suited if you can at least pass the "Hello, world" test[4]. The Microsoft C 5.1 optimizing compiler generates CLIPS versions which run 15-25% faster than the COSMIC executable using the "/Ox" switch for maximum optimization. However, Microsoft C 5.1 does not have an integrated development environment and takes longer to compile. Table 3 shows Microsoft C 5.1 compiler switches you need for successful compilation and others that may improve the performance of your application.

Several details are of note in using the Microsoft C 5.1 compiler. If you are not using the COSMIC-supplied make file to control compilation, you will want to use the switch "/FeCLIPS" to make the executable be named CLIPS instead of the name of the first source file, ANALYSIS. The source file RULEMNGR.C will usually get a fatal compiler error (error code C1001) due to the "/Ox" switch. If you are compiling all the source files and this happens, don't panic -- just compile RULEMNGR.C again by itself without any optimization. Use the library switch "/link /NOE" at the end of the compiler command line to prevent the symbol "_iota" from being "multiply defined" (error code L2044). To handle a large number of facts, which may result in the run-time error message "R6000 stack overflow", use the "/F 1000" switch to increase the stack size from a default of 2K bytes to 4K bytes or larger.

**Compiling a knowledge base.** Compiling both the rule base and CLIPS functions into a single executable file will save considerable memory. This also makes the knowledge base opaque to the user, which is useful for distributing applications that incorporate proprietary knowledge. A compiled application is built by compiling the knowledge base, making CLIPS run-time modules, and linking them into a single executable.

To make a compiled expert system, first make two new versions of CLIPS: a rules-to-c version (using the CLP_RULE_COMP flag in SETUP.H), and a run-time version (using the RUN_TIME flag). You must make new versions because both these flags are turned off in the COSMIC-distributed versions of CLIPS. After editing the source files, re-compile them to make new object files. Link the rules-to-c version object files to make a new CLIPS executable with which you can compile a rule base into C source code.

Before compiling again to make the run-time version object modules, edit the file MAIN.C to add the functions "init_clips()" and "init_c_rules_1()" immediately before the reset and run commands already there. You must keep all the other flags in their original state for this version, except set the IBM_MSC flag to 1. (Note that most environment commands and some embedded function calls are not available in the run-time version -- a complete list of these is at the top of page II-31 in the Advanced Programming Guide. These functions are mainly debug commands entered at the CLIPS prompt, and not usually used in rules.) Don't link the run-time object files together yet, as they can't do anything without the knowledge base.

The knowledge base is actually compiled twice: once into C source code, and again into C object modules. First, load the rules and initial facts into your new rules-to-c version of CLIPS. Next, enter the rules-to-c command with a suitable module name and reference number, e.g., "(rules-to-c expersys 1)<cr>", and wait while CLIPS automatically generates 8 text files (10 in version 4.2). The generated files contain hash tables equivalent to the compiled knowledge base. Compile them into object modules using a large or huge memory model. Finally, link the resulting knowledge base object modules with the CLIPS run-time object modules to make a complete executable. The resulting compiled application will behave the same as CLIPS did with the original knowledge base, except use significantly less memory and perhaps run a bit faster.

**Compiled versus standard rules.** Table 4 shows a comparison between standard versus compiled knowledge bases for several CLIPS applications. The MAB rule base (Monkeys and Bananas, CLIPS 4.2 version) is included in Table 4 as a point of reference. MAB and HEIRS were compiled using CLIPS 4.3 source code; TRACER and NOD2ATL used the older version 4.2 source code.

Table 4 illustrates several salient characteristics of CLIPS versions 4.2 and 4.3 in the DOS environment. Perhaps the most striking is that the COSMIC CLIPS version 4.3 uses about twice as much memory for loading rules as version 4.2. This can cause the newer CLIPS to run out of memory while loading or executing, even though it takes less memory to start with. Preliminary investigations show that the increased memory usage is not evident after re-compiling CLIPS 4.3 with Microsoft C 5.1.

As may be expected, execution times in Table 4 are dominated by the number of facts, not the number of rules. The two knowledge bases with the least rules but the most facts, TRACER and NOD2ATL, take about five times longer to execute than HEIRS, which has several times more rules but far fewer facts. Execution of all the compiled knowledge bases is somewhat faster than the same rule base loaded into COSMIC's CLIPS 4.2 version; the COSMIC 4.3 version results are inconclusive.

Figure 3 plots the number of rules versus size of the compiled expert systems. The result is a linear relationship, with a core size of about 180K bytes plus 640 bytes per compiled rule. Compiling a knowledge base saves from 85K bytes (TRACER) to 185K bytes (HEIRS) of memory. For the HEIRS knowledge base with CLIPS 4.3, this saves enough memory for it to run successfully while the un-compiled version does little more than load the rules before running out of memory.

Figure 4 plots the additional memory used per rule during execution of the different expert systems.

Two trends are apparent, one for small knowledge bases and a higher rate for larger knowledge bases. For smaller knowledge bases, the line defined by TRACER, NOD2ATL and MAB shows additional memory requirements of about 8K bytes plus 1K byte per rule. HEIRS indicates that larger knowledge bases use about 3K bytes per rule (with CLIPS 4.2).

## SUMMARY

CLIPS is suitable for many data-driven information processing tasks. A sound understanding of its characteristics and their implications is essential for effective application development. You may avoid potentially limiting mistakes by applying this paper's concepts and techniques. Compiling CLIPS can alter its features, solve marginal memory problems, and protect proprietary knowledge.

## ACRONYMS AND ABBREVIATIONS

| | |
|---|---|
| ANSI | American National Standards Institute |
| ASCII | American Standard Code for Information Interchange |
| ATLAS | Abbreviated Test Language for All Systems |
| CLIPS | C Language Integrated Production System |
| COSMIC | Computer Software Management and Information Center |
| CRSV | Cross Reference, Style and Verification utility |
| DATSA | Depot Automatic Test System for Avionics |
| HEIRS | Harris Expert ITA Routing System |
| ITA | Interface Test Adapter |
| K | multiply by 1024 |
| LHS | left-hand side, premise of a rule |
| LISTER | node list to wirelist expert system |
| NOD2ATL | node list to ATLAS expert system |
| RHS | right-hand side, conclusion of a rule |
| TRACER | a wire tracing expert system |
| UUT | unit under test |

## REFERENCES

1. Leinweber, D., "Knowledge-Based Systems for Financial Applications," IEEE Expert, Fall 1988, Vol. 3, No. 3, pp. 18-31.

2. A good example of backwards chaining is the WINE.CLP example distributed with CLIPS. See the CLIPS User's Guide, Chapter 10, for details on building goal structures and combining evidence.

3. CLIPS Advanced Programming Guide, NASA/COSMIC ref # JSC-22948 (for version 4.3), pp II-62 to II-65.

4. In many C language primers, the first C program a beginner writes is an output statement which prints the message "Hello, world" to the terminal. Anyone who understands programming concepts and is familiar with C language development enough to create this simple program should have little trouble using the Microsoft C compiler.

Table 1. A Summary of TRACER, LISTER, and NOD2ATL

| | TRACER | LISTER | NOD2ATL |
|---|---|---|---|
| Rules | 9 | 8 | 19 |
| Preconditions/rule | 1.7 | 1.9 | 2.8 |
| RAM for rules | 16K | - | 25K |
| Facts | 992 | 97 | 97 |
| RAM requirement | 521K | - | 491K |
| Hardware | VAX/PC | PC | PC |
| Run time (min:sec) | 3:18/6:09 | - | 6:53 |
| Development time | about 1.5 man-months total | | |

Table 2. The HEIRS Expert System

| | | |
|---|---|---|
| Rules | 77 | |
| Preconditions/rule | 4.3 (0-8) | |
| RAM for rules | 147K | |
| Facts | 97 UUT + 280 DATSA* | |
| RAM for facts | 91K | |
| Hardware | PC-AT | VAX 780/8800 |
| Run time (min:sec) | 2:02 | 1:40 / 0:31 |
| Development time | about 5 man-months | |

* UUT is the circuit card, DATSA is the tester.

Table 3. Microsoft C 5.1 compiler flags for CLIPS

| | |
|---|---|
| /AL | Uses large memory model **REQUIRED** |
| /F <hex number> | Sets the stack size |
| /FeCLIPS | Names the executable CLIPS |
| /FPa | Uses alternate math library |
| /G2 | Uses 80286 instruction set |
| /Ox | Makes faster executable |
| /link /NOE | Avoids _iota error **REQUIRED** |

Table 4.
A Comparison of Conventional and Compiled Knowledge Bases.

| | CLIPS 4.2 | Mod. 4.2 | CLIPS 4.3 | Compiled |
|---|---|---|---|---|
| CLIPS shell | 271 | 218 | 198 | - |
| TRACER - 8 rules | 24.6 | 16.4 | 33.8 | 198 |
| RAM/rule | 3.1 | 2.1 | 4.2 | - |
| rules (c) | 16.4 | 16.4 | 30.7 | - |
| RAM/rule (c) | 2.1 | 2.1 | 3.8 | - |
| RAM to run (c) | 250 | 252 | -(1)- | 238 |
| run time (c) | 6:09 | 3:54 | -(1)- | 4:35 |
| NOD2ATL - 19 rules | 32.7 | 32.7 | 56.3 | 200 |
| RAM/rule | 1.7 | 1.7 | 3.0 | - |
| rules (c) | 24.6 | 24.5 | 50.2 | - |
| RAM/rule (c) | 1.3 | 1.3 | 2.6 | - |
| RAM to run (c) | - | - | - | 286 |
| run time (c) | 6:53 | 5:38 | -(1)- | 5:30 |
| HEIRS - 77 rules | 184 | 184 | -(2)- | 290 |
| RAM/rule | 2.4 | 2.4 | -(2)- | - |
| rules (c) | 147 | 148 | 278 | - |
| RAM/rule (c) | 1.9 | 1.9 | 3.6 | - |
| RAM to run (c) | 238 | 238 | -(1)- | 98 |
| run time (c) | 1:22 | 1:06 | -(1)- | 1:10 |
| MAB - 32 rules | 40.9 | 40.9 | 72.7 | 213 |
| RAM/rule | 1.3 | 1.3 | 2.3 | - |
| rules (c) | 32.7 | 32.7 | 65.4 | - |
| RAM/rule (c) | 1.0 | 1.0 | 2.0 | - |
| RAM to run (c) | 40.9 | 40.9 | 231 | - |
| run-time (c) | 0:03.4 | 0:03.0 | 0:02.1 | 0:03.4 |

NOTES:
RAM values in Kbytes (x1024), times as minutes:seconds
All tests run on: ACER 900 PC-AT 286 @ 12 MHz, DOS 3.3, 640K RAM, 589K free, 28 ms disk
(c) rules loaded with (conserve-mem on)
(1) COSMIC version 4.3 ran out of memory while executing
(2) COSMIC version 4.3 ran out of memory while loading

```
(deffacts initial-facts
    (phase-state-names
        "LOADING UUT TEST REQUIREMENTS"
        "ANALYZING SIGNAL PATH REQUIREMENTS"
        "COMPARING SIGNAL PATH REQUIREMENTS"
        "RESOLVING PATH CONFLICTS"
        "ASSIGNING POWER AND GROUNDS"
        "ASSIGNING DATSA RESOURCES"
        "PRINTING WIRELIST")
   (need-to-load design.dat)
   (worst-id-con none-assigned none-assigned 0)
   (tasks expand consolidate1 consolidate2 compare)
   (phase-state 0))
```

Figure 1. Initial facts for HEIRS include phase names.

```
(defrule change-to-next-phase
   (declare (salience -1000))
   ?f1 <- (phase-state ?phase)
   ?f2 <- (phase-state-names ?next $?list)
   (not (error ?))
   =>
   (retract ?f1 ?f2)
   (assert (end-of-phase ?phase))
   (format t "%n---------------------- ")
   (format wtrace "Phase %d Complete" ?phase)
   (format t " -------------------%n%n%n================< ")
(format wtrace "%s" ?next)
   (format t " >================")
   (if (eq ?phase 0)
      then
      (assert (tag-num 10000)))
   (if (eq ?phase 3)
      then
      (assert (need-to-load datsa06.pin)
              (need-to-load datsa02b.pin)
              (need-to-load datsa01b.pin)))
   (if (eq ?phase 4)
      then
      (assert (term-block-num 1)
              (need-to-load datsa01a.pin)
              (need-to-load datsa02a.pin)))
   (if (eq ?phase 5)
      then
      (assert (need-to-load datsa09.pin)
              (need-to-load datsa13a.pin)
              (need-to-load datsa03a.pin)
              (need-to-load datsa17.pin)
              (need-to-load datsa19.pin)
              (need-to-load datsa21.pin)))
   (if (eq ?phase 6)
      then
      (open "wire.lis" list "w")
      (format t "%n> opening file \"")
      (format wtrace "wire.lis")
      (format t "\" for writing "))
   (assert (phase-state =(+ ?phase 1))
           (phase-state-names $?list)))
```

Figure 2. HEIRS phase change rule performs initial actions when changing to certain phases.

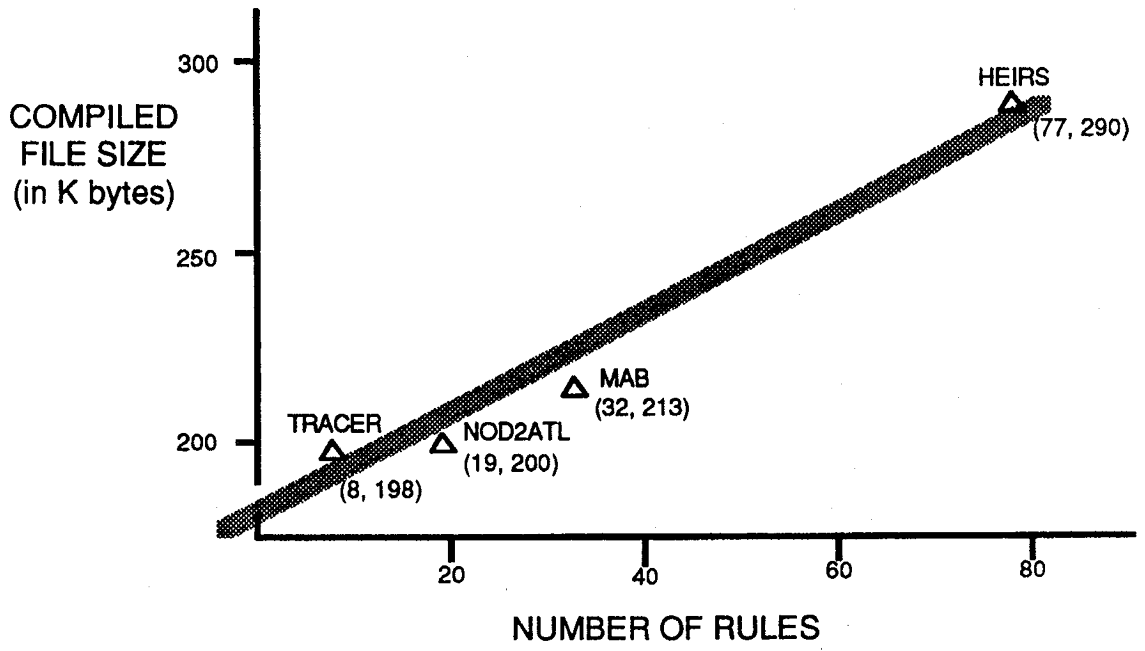


FIGURE 3. COMPILED FILE SIZE VS. NUMBER OF RULES

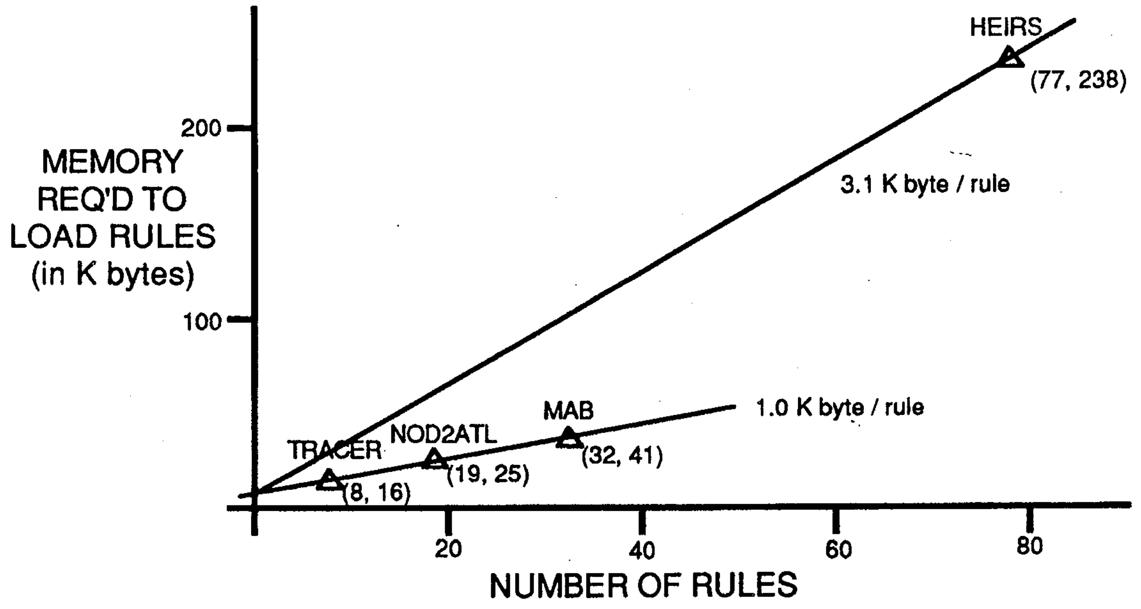


FIGURE 4. MEMORY TO LOAD VS. NUMBER OF RULES

# DYNAMIC ARRAY PROCESSING FOR COMPUTATIONALLY INTENSIVE EXPERT SYSTEMS IN CLIPS

N.N.Athavale[1], R.K.Ragade[1], T.E.Fenske[2], M.A.Cassaro[2]
University of Louisville
Louisville KY 40292

## ABSTRACT

This paper puts forth an architecture for implementing a loop for advanced data structure of arrays in CLIPS. An attempt is made to use multifield variables in such an architecture to process a set of data during the decision making cycle. Also, current limitations on the expert system shells are discussed in brief in this paper.The resulting architecture is designed to circumvent the current limitations set by the expert system shell and also by the operating environment. Such advanced data structures are needed for tightly coupling symbolic and numeric computation modules.

## INTRODUCTION

All engineering design projects are highly computationally intensive. Many low cost expert systems shells are better at symbolic computation than numeric. Few of them handle external modules and very few of them allow use of advanced data structures such as arrays. Numeric processes or symbolic processes alone will not address all the problems in the design process, but a coupled system can provide a better solution. Two types of coupling are possible: i) a tightly coupled system and a ii) a loosely coupled system. The tightly coupled system should have some knowledge of the numeric processes used within the systems [1]. It makes the system more robust in nature. In the loosely coupled system, the expert system module will not have any knowledge of the numeric processes being used.

Even so, a loosely coupled system will have advantages over the individual symbolic or numeric processing. Any coupled system can fully automate the design and analysis process. Usually, numerical processes in engineering design and analysis are computationally very complex. There are a large number of iterations. For large problems, there is a very large storage requirement for intermediate or incomplete results. Such a bottleneck limits the usage of these programs. Intelligent processing can handle the numerical processes to avoid this bottleneck.

One is naturally led to ask, how an expert system shell may help in reducing the bottleneck mentioned above.

This project is partially funded by the Federal Highway Administration and the Kentucky Department of Transportation

[1]Department of Engineering Mathematics And Computer Science

[2]Department of Civil Engineering

## ROLE OF EXPERT SYSTEMS SHELLS:

Expert system environments can have limitations. These limitations are usually are imposed by their development shells and environments. A variety of expert system development shells are available in the market. At one end of the spectrum are less expensive systems and at the other end of the spectrum are the high performance systems. Many expert system shells with less cost are limited in their capabilities such as inability to represent data structures, a small library of basic numeric functions, and poor interfacing capabilities. High performance, complex expert system shells have increased capabilities that make use of LISP based architectures in order to improve their performance. Such machines are expensive and also restrict number of users as they need intensive training in its use [2].

Many shells do not have advanced data structures in their environment. Thus, the programmers have to simulate advanced data structure processing in such shells. Such overhead makes the package less efficient, as many sessions of loading the interfaces and read/write on disk files consume precious time. In many cases, advanced data structures will dominate the time complexity of the complete package. Also, a few auxiliary control rules are required to simulate such data structures. Many microcomputer based shells limit the available memory for interfaces. This limitation demands hardware changes and thus is more expensive to use.

A number of shells interact with a user, on a screen basis. Every new input is asked using a new screen. A few shells provide graphics functions to custom design the screens. Such shells caution programmers when dealing with screen design issues in the interfaces. A programmer may not use any sensitive parameters such as control characters for the screen designs. This limits a programmer from changing the monitor parameters set by the shell.

In engineering design problems, one is typically faced with a large number of inputs. A user would be tired with too many screens, each scrolling leisurely to ask for just one input. Interfacing schemas are also as limited as in the case of screen handling. Such shells can call any external program but no external program can call such a shell. Thus, the control always remains within the shell. Many programmers prefer the locus of control to be in external modules rather than in an expert system. Every time a programmer decides to execute an external module a decision has to be made by the expert system and execute the module as an external program. This adds to the already high time complexity of the integrated package.

Many shells allow external data communication. Often this communication is allowed only through data files. This interaction using files is very slow and an inefficient way of communication between interfaces and expert system. If one or more external modules exchange data with the expert system then the system runs at slower speed, resulting in a longer response time. To some extent, the performance of an expert system can be improved by regrouping the rules.

A few utility programs are accessories to such shells. However, these programs can take a long time to be understood by the programmers for better programming. Typical utilities are provided to improve the performance by rearranging the rules in the expert system, to provide a cross reference of the rules and the variables, to transfer the rules from one operating systems environment to another, merging many expert systems into a single expert system. Computer-Aided Engineering applications demand intensive numeric processing abilities from the expert systems. The above mentioned limitations may not fulfill the demands of the programmer developing a heterogeneous system for an engineering design problem.

In summary, the degree of coupling can be examined along the dimensions of data communication, array handling and module invocation. These issues are discussed with reference to the interaction of CLIPS in a heterogenous package called KYBAS1.2 [3].

## KYBAS 1.2 ARCHITECTURE :

A prototype expert system for the Kentucky Bridge Analysis (KYBAS) project is developed in CLIPS. The prototype consists of three main modules: 1. expert system, 2. numeric code 3. interfaces.

This package processes arrays of data during the decision making and analysis phases in design. This is particularly crucial in our implementation of a prototype framework of interacting expert systems for structural analysis and design of highway bridges [4]. The Master Program, written in FORTRAN, controls the heterogeneous system all the time. Initially, it calls the expert system to obtain the geometry of the bridge and to design a finite element mesh.

The expert system generates the mesh and executes interfaces to get the structural and material information needed to create the data file for the finite element analysis code. The finite element analysis code analyzes the mesh generated by the expert system and outputs the results into a data file. This architecture is possible as CLIPS can be called by any external programs [5].

The KYBAS project has a major goal of an intelligent computer assisted bridge design. There are different functions to be performed during the design process of a highway bridge. First, the location of a bridge is determined. After this decision, the type of the bridge is decided. The bridge can be a single span bridge, a multi-span bridge or a curved bridge. The type is based on the
surrounding conditions of the bridge. These conditions include the under-passing structure, skewness and anticipated traffic load. The type of bridge also depends upon construction and fabrication cost.

## KNOWLEDGE REPRESENTATION IN KYBAS:

A rule-based expert system is used to represent the knowledge of an expert in the finite element modeling of structures. The rules are written to apply this knowledge to generate an optimum size grid for finite element model and pass it on to the numeric modules during the analysis process.

There are four major groups of the as shown in Figure 1. The input group of rules gathers all data related to the geometry of a bridge. It is further subdivided into functionally dependent subgroups.

The first subgroup, SPANS, accepts the numerical model of the spans in a bridge. The second subgroup is BEAMS. This group takes in geometry of the beams used in the bridge. The third group is DIAPHRAGMS. It reads in the geometry of the diaphragms in the bridge. The next group SUPPORTS takes in information about the boundary conditions of the bridge supports. The last subgroup CURBS reads data about the skewness of the bridge, distances between the curbs and immediate beams. It is a part of BEAMS subgroup.

The next major group is PROCESS-AND-DECIDE group of rules. The PROCESS-AND-DECIDE group is subdivided into subgroups similar to input groups. The first subgroup LATERAL uses the available knowledge and processes the data to generate the number of nodes in a column in lateral direction. The second subgroup LONGITUDINAL decides the number of nodes in the longitudinal direction. This set of rules decides any further refinement of the mesh at specific positions and its effect on the rest of the mesh. The architecture used for this set is a Loop-Architecture inside another Loop-Architecture. This architecture is explained later in the paper.

The other subgroups at the top level are not developed in the current phase of the system, but they are on the agenda for the next phase of development. These subgroups include a SKEWNESS group, a COST group and a DATABASE query group. The last main group is the OUTPUT group. This group outputs the decisions and the key information generated, in the file EXPERT.DAT. The interfaces are invoked to generate and analyze the finite element mesh of the bridge. The analysis of the mesh is not passed to the expert system at this time.

## ARRAY HANDLING:

Arrays can be represented in different ways in a programming language. Arrays can be defined as strings of numbers stored one after the other, they can be a linked list of numbers or arrays can be stored in a record with each field is a location in the array.

The simple way to represent an array is to define it as a new data structure and use the widely available array handling mechanisms to process the arrays. However, this type of representation will have its overheads in an application program such as an expert system development tool written in a higher level language. Its complexity will be determined by the host higher level language and such representation will demand more memory from the system. Also, such definitions will orient the expert system shells around the numeric processing and the control mechanisms in the shell will be influenced by advanced data structures. This effect is not desirable for expert system shells as expert system shells are written to improve symbolic processing than the traditional programming languages without any special hardware.

Arrays of numbers or strings can be represented in CLIPS. These are called "multifield" variables.

CLIPS provides a mechanism to represent multifield variables. These variables can be strings of characters or numbers or combination of both. This data structure can be used to simulate an array in CLIPS. A multifield variable consisting only of numbers can be treated as a single dimensional array or a vector of values. The next section explains an architecture for such an array simulation.

## LOOP ARCHITECTURE :

The same set of rules works on different sets of data iteratively. This architecture is unique as it handles the arrays without any explicit loop constructs and avoids any side-effects on the rest of the system. These side effects can cause execution of unwanted rules, and also cause memory overflow due to the growing list of facts. This is the architecture referred to as the Loop-Architecture in an earlier section.

The multifield variable can be treated as a fact during a particular array processing application. Every time a single value is processed from the array, the index, another fact, is incremented and the fact with previous array index is deleted. The new contents are substituted for the old contents of a location indexed by an index variable and the array with the previous values is replaced by the array with new values.

The first rule will check the index, and if the index is within the range, the value will be accessed and processed in the action part of the rule. When a new value is established for the current value, following actions are taken.

1. The fact representing the index variable is modified,
2. The new value is placed at the current location in the array, and,
3. The old image of array is deleted.

The constructs required for such a loop are shown in figure 2. The TEST construct checks the index for its values and the reassertion of the facts causes CLIPS inference engine to fire the same rule again. This process continues until all the values in an array are processed. This end of processing can be detected by the TEST of index variables. The example in the figure 2 works as follows:
Initially the user resets the system and runs the rule. The first rule takes the values and asserts facts so that the second rule is fired. At this time the rules enter in a loop.

The fact **(loop ?i ?max)** represents the index variable for the array
**(list $?array-elements).**
The variable **?i** is the index variable and the number of elements in the array is represented by the variable **?max**. This implementation starts with the **?i = 1** and terminates
when **(?i > ?max)**. The second rule uses this **?i** to access the **i**th variable in the array and compares it with 5.0. If the **i**th element is greater than 5.0 then the rule prints out "above threshold value". The next action taken by this rule increments the value of **?i** and asserts a new fact. This causes the same rule to fire and it continues until the limiting conditions are satisfied. The factlist at the end of execution of this loop is as shown in figure 3. Note the extra unwanted fact **(loop 5 4)** is present.

However, this implementation leaves few extra facts in the environment. The fact simulating the array index is not retracted after the test fails and such extra facts may cause side effects in the system. Thus, a new implementation is developed to retract such extra leftovers from the array processing in CLIPS.

The figure 4 shows that the action part of the rule contains a conditional branching in the actions. It uses

**"if......then.....else"**

construct to examine the index variable and to select the group actions to be taken. The "then" part contains the actions related to the array processing and the "else" part contains the retraction of the extra facts. This implementation performs comparably with the previous one but, it avoids any potential side effects making it a more desired implementation. The example shown in figure 4 executes as follows:

The execution starts after the user resets the system and issues a **(run)** command. The first rule asserts the necessary facts and these facts cause second rule to fire. The second rule starts with the

facts **(count ?i), (no_of_elements ?max)**

and

**(list $?array_elements).**

The variable **?i** is initialized to 1 by the first rule. There are 4 elements in the array list. It is represented by the fact (**max 4**). The action part of this rule contains conditional actions. If the count is greater than **?max** , then the rule retracts unwanted fact (**count 5**) and also (**max 4**) if necessary. If the count is not greater than ?max, then it carries out its actions in the similar way as that of the previous example. The factlist, after completion of the execution, will not contain any unwanted facts (figure 5). This increases the space utilization of the program and also avoids any potential failures.

The multidimensional arrays can be simulated by two multifield variables. The first multifield variable contains the values of array rows in sequence. i.e. the first rows of values is followed by the second rows of values and so on. The second multifield variable contains the dimensions. The first dimension is followed by the second dimension and so on.

This multidimensional array can be processed in an implementation
similar to the vector (single dimension) processing implementation. If the multidimensional array has n dimensions then (n-1) more conditions are added to control the index of each dimension.

This implementation requires more number of rules, but, it provides a viable scheme to simulate multidimensional arrays. This is an important requirement of engineering applications involving expert systems. This implementation causes the system to be more inefficient. However, the system can be used more effectively and can cover a broader range of activities.

## CONCLUSIONS:

This paper provides a feasible framework for the implementation of multidimensional array processing in CLIPS. The system performs better if the TEST construct is not used.

The **"if.......then.....else"** construct makes same number of comparisons but provides a mechanism to delete undesired facts left over during the array processing. A traditional way of array handling will provide the applications a greater flexibility, with a penalty of reduced efficiency. This bottleneck can possibly be overcome by the issues in the parallel processing. The degree of coupling will influence the performance of such systems. This architecture is reusable in many expert system architectures including distributed expert systems.

## REFERENCES :

[1] Kitzmiller, C. T.; Kowalik, J. S. "Symbolic and Numerical Computing in Knowledge based Systems," Coupling Symbolic and Numerical Computing in Expert Systems, ed. by: Kowalik J. S., Elsevier Science Publ., New York NY 1986.

[2] Hojjat Adeli, "Expert System Shells" in Hojjat Adeli (ed.) Expert Systems in Construction and Structural Engineering, Chapman and Hall, London UK, 1988.

[3] Athavale N.N., R.K.Ragade, Cassaro M.A. and Fenske T.E., " A Prototype for KYBAS: the Kentucky Bridge Analysis System" Proceedings of the Third International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Charleston SC, July 15-18 1990.

[4] Athavale, N.N., "Development of an expert systems framework for Kentucky Bridge Analysis System (KYBAS)", Masters Thesis, Dept. of Engineering Mathematics and Computer Science, University of Louisville, Louisville Kentucky, 1989.

[5] Giarratano, J.C., "CLIPS User Guide" Artificial Intelligence Section, Lyndon B. Johnson Space Center, June 1988.

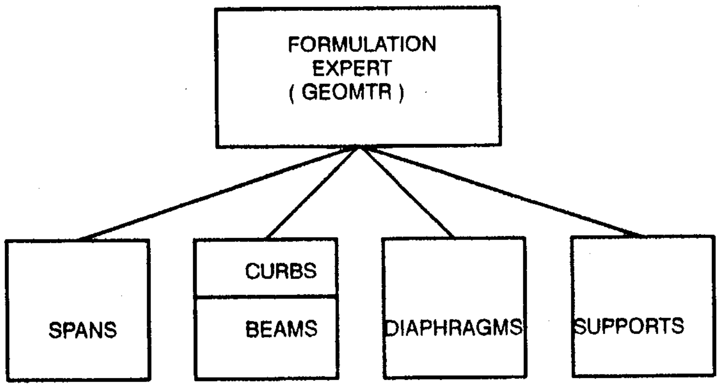


Figure 1. Four major groups in the expert system.

```
(defrule rule1 "initiate the processing"
?rem1 <- (initial-fact)

=>
(retract ?rem1)
(assert (list 1 2 3 4))
(assert (loop 1 4)))   ;initialize index variable to 1.

(defrule rule2  "go into the loop"
(list $?array_elements)
?rem1 <- (loop ?i ?max)
(test (<= ?i ?max))

=>
(retract ?rem1)
(assert (loop =(+ ?i 1) ?max))
(bind ?tmp (nth ?i $?array_elements))
(if  (> ?tmp 5) then
(fprintout t "above threshold value" crlf))
```

Figure 2. Array simulation in CLIPS

```
(loop 5 4)
(list 1 2 3 4)
```

Figure 3. Facts present after the execution of rules.

```
(defrule rule1 "initiate the  processing"
?rem1 <- (initial-fact)

=>
(retract ?rem1)
(assert (list 1 2 3 4))
(assert (no_of_elements 4))
(assert (count 1)))   ;initialize index variable to 1.

(defrule rule2  "go into the loop"
(list $?array_elements)
(no_of_elements ?max)
?rem1 <- (count ?i)

=>
(retract ?rem1)
(if (<= ?i ?max) then
              (assert (count =(+ ?i 1)))
              (bind ?tmp (nth ?i $?array_elements))
              (if  (> ?tmp 5) then
              (fprintout t "above threshold value" crlf))
```

Figure 4. Array simulation in CLIPS

```
(list 1 2 3 4)
```

Figure 5. Facts present after the execution of rules.

## B1 Session:
## Intelligent Tutors and Training

# Intelligent Tutoring Using HyperCLIPS

Randall W. Hill, Jr.
Brad Pickering

Jet Propulsion Laboratory
4800 Oak Grove Drive M/S 125-123
Pasadena, CA 91109

## 1. Introduction

HyperCard® 1 is a popular hypertext-like system used for building user interfaces to databases and other applications, and CLIPS is a highly portable government-owned expert system shell. We developed HyperCLIPS in order to fill a gap in the U.S. Army's computer-based instruction tool set; it was conceived as a development environment for building adaptive practical exercises for subject-matter problem-solving, though it is not limited to this approach to tutoring. Once HyperCLIPS was developed, we set out to implement a practical exercise prototype using HyperCLIPS in order to demonstrate the following concepts: learning can be facilitated by doing; student performance evaluation can be done in real-time; and the problems in a practical exercise can be adapted to the individual student's knowledge.

## 2. Intelligent Tutoring: Theme and Variations

### 2.1 Approaches to intelligent tutoring

The way in which computer-assisted instruction systems are typically implemented is to use the *tutorial and inquiry* mode of instruction wherein the computer presents a sequence of frames of graphic and textual information to the student, and at predetermined points in the lesson a multiple choice quiz is given to test the student's retention of the subject contents. This approach has become fairly standard both in the Army [1] and in the computer industry (and is supportable by HyperCLIPS), but it is by no means the only way of training someone through the use of a computer. A number of other approaches have been demonstrated: WEST [2] exercises arithmetic skills through the use of a game. A *coach* intervenes when the student has recognizable

1. HyperCard® is a registered trademark of Apple Computer, Inc.

trouble with an issue on how to do some arithmetic operation. WHY [3] uses *Socratic dialogue*, teaching through the use of questions and examples, as a means of instruction about meteorology. STEAMER [4] is an interactive simulation of a steam-plant on a naval vessel where the student learns by manipulating the controls over the environment, and IDEBUGGY [5] is an interactive practical exercise system for doing problems in subtraction. One of the common threads running through all of these systems is that they attempt to do more than just present information along a set course of instruction (i.e. they attempt to be more than electronic page-turners). What distinguishes systems such as the ones mentioned above is that they provide a more dynamic environment for the student than the traditional computer-assisted instruction (CAI) approach. This dynamism is created by making the instruction more reactive to the student through the use of simulations of physical systems, coaching, interactive dialogues, and adaptive practical exercises. Student modeling plays a key role in deciding what to do next; the model provides a way of identifying misconceptions as well as what the student does not know. Our claim in this paper is that HyperCLIPS can support a wide range of these instructional techniques, as we will illustrate with a detailed example in a later section.

**2.2 Student Modeling**

One of the ways to make an instructional system adaptive is through the use of student modeling. A student model is a representation of the current state of the student's knowledge [6]. It is used for the purpose of identifying missing knowledge as well as for diagnosing misconceptions. There are two well known techniques used for doing student modeling:

2.2.1 Overlay method - One of the most common and simplest ways of doing student modeling is the overlay method. The basic idea behind this method is that whatever the student knows or learns will be a subset of what the expert knows. So, the first step is to build an expert model of the domain knowledge. As the student interacts with the tutoring system through tests and practical exercises, the system can begin to *overlay* the student's demonstrated knowledge on the expert model. By doing differential analysis, the tutoring system can detect knowledge gaps that require further tutoring [7]. In the example described in section 4 we use the overlay method to model the student's knowledge of map symbols. The limitation with this approach to modeling is that it only indicates where the knowledge gaps exist, but it does not identify misconceptions [6].

2.2.2 Bug libraries - Another approach used in student modeling is to develop a library of "bugs", or misconceptions , that represent the procedures or errorful inference rules that might be used by the student to perform some operation. The bug library is an enumeration of the possible misconceptions a student might have about a subject, and it can be used to generate "bad" solutions to problems the student is solving in order to match with an incorrect student response. This approach overcomes limitations of the overlay method, but it has the disadvantage that it takes a great deal of effort to enumerate the misconceptions for a particular domain -- and even in the most well-defined domains, such as subtraction, there may be difficulties in pinpointing the misconception due to multiple interacting "bugs" [5,6].

## 3. HyperCLIPS

### 3.1 HyperCLIPS = HyperCard + CLIPS

Based on our review of the work that has been done in the field of intelligent tutoring, we knew that we needed two essential capabilities: (1) a graphical user interface for representing non-textual data, and (2) a reasonable knowledge-based representation language and inference engine. For this reason it seemed natural to integrate the capabilities of HyperCard and CLIPS. HyperCard takes advantage of many of the user interface features of the Apple Macintosh®[2] in a hypertext-like environment, thereby providing a great deal of flexibility for building the front-end to a tutoring system. CLIPS, on the other hand, provides the knowledge-based reasoning needed for doing some forms of student modeling. HyperCLIPS has many potential uses other than intelligent tutoring, and it provides many of the features we desired for this particular domain.

### 3.2 Using HyperCLIPS

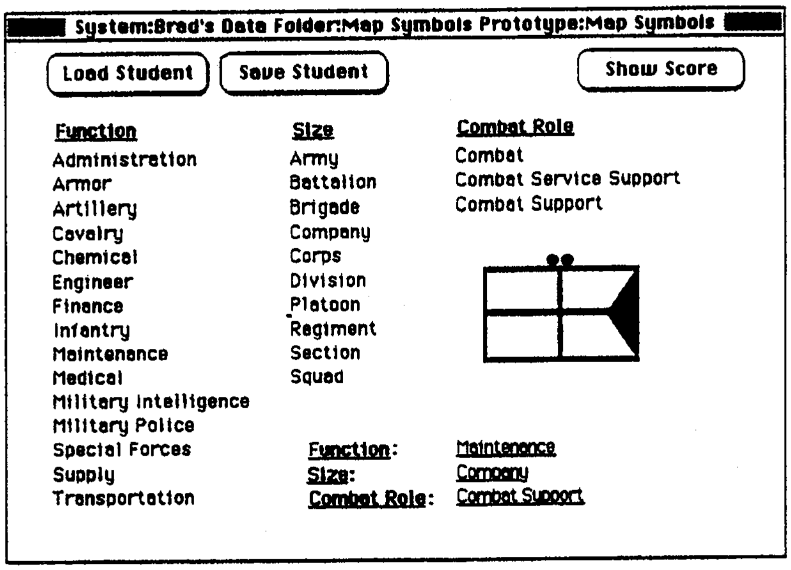
HyperCLIPS is fairly easy to use if one is already familiar with HyperCard and with CLIPS. The main issue that confronts the HyperCLIPS user is how to separate the user interface from the inference procedures. The feature of HyperCLIPS that must be learned is how to pass control back and forth between the user interface and the CLIPS production system. HyperCLIPS provides several commands for passing control, which are described in greater detail in [8].

## 4. An Example: A Practical Exercise in Map Symbol Recognition

### 4.1 The world of map symbols

We chose map symbol recognition as the domain for demonstrating the benefits of using HyperCLIPS for creating practical exercises. Map reading is a basic skill for the military intelligence analyst, for whom the military unit map symbol conveys

[2] Macintosh® is a registered trademark of Apple Computer, Inc.


System:Brad's Data Folder:Map Symbols Prototype:Map Symbols

| Function | Size | Combat Role |
|---|---|---|
| Administration | Army | Combat |
| Armor | Battalion | Combat Service Support |
| Artillery | Brigade | Combat Support |
| Cavalry | Company | |
| Chemical | Corps | |
| Engineer | Division | |
| Finance | Platoon | |
| Infantry | Regiment | |
| Maintenance | Section | |
| Medical | Squad | |
| Military Intelligence | | |
| Military Police | | |
| Special Forces | | |
| Supply | | |
| Transportation | | |

Function: Maintenance
Size: Company
Combat Role: Combat Support

**Figure 1: Map Symbol Exercise. Choose a function, size and combat role by clicking on a list item. The chosen items appear in the lower right area.**

important information about the size, function and combat role of a battlefield unit. These symbols are small icons that are placed on a map to denote the geographic locations of opposing and friendly forces. In addition to being geographically contextualized, the military unit map symbol is juxtaposed among other such symbols, thereby providing the analyst with the basis for making inferences about the disposition, strength and strategy of an opposing force. Without a thorough knowledge of how to interpret military unit map symbols, the intelligence analyst is not able to make inferences about an opposing force's composition and capability.

## 4.2 Instructional objectives

The goal of the practical exercise is to help the student master the skill of recognizing military unit map symbols through the use of an interactive test environment (we assume that the student has already received some instruction on the basic rules for recognizing a map symbol and is ready to put this knowledge into practice). Implicit to this goal is the objective of helping the student to learn the rules for analyzing a map symbol by decomposing it into its atomic components rather than trying to look at the symbol as a whole. There are two kinds of learning involved with this process: rote memorization of the sub-components and analysis by decomposition.

## 4.3 Interaction strategy

From the user's perspective, interaction with the practical exercise is simple: the student is presented a military unit map symbol, the student attempts to identify the symbol, the system corrects errors in identification and provides feedback, and a new map symbol is presented. (See Figure 1 for an example of the interface.) One of the salient features of the practical exercise is that the symbols that are presented to the student are generated randomly from the student model, which provides information about where the student's weaknesses exist. Hence, the interaction with each student is unique; it is adapted to the needs of the individual as predicated by the student model, which is built up from interaction with the student.

**4.4 Representing knowledge about map symbols**

The domain knowledge about military unit map symbols is fairly simple and can be broken into three categories: the unit size (squad, platoon, ...) , unit function (artillery, armor, ...), and unit combat role (combat, combat support, ....). This information is stored in templates in the CLIPS production system. Each sub-component of the three categories also has a standard graphical portrayal -- the procedure for drawing the sub-component is represented in HyperTalk®[3], the language used in HyperCard to manipulate the user interface. The link between the production system representation and the user interface representation is made by giving each sub-component a "picture name" that can be used for communicating between the two representations. In addition to the template information about the map symbol sub-components, there are rules for assembling map symbols from sub-components. The rules encode the structural information about map symbols and are used to execute the generation of a symbol for the practical exercise.

**4.5 Representing a model of the student**

We chose to use the *overlay method* for representing the student. In this case, the expert model is comprised of the set of all military unit map symbol sub-components. The template for symbol has a slot called "mastery-level" that keeps track of how many times the student has correctly identified the particular sub-component. When the production system is generating a new problem, it evaluates the relative mastery-level of the various sub-components and derives a "weakness set" from which to randomly choose sub-components for the next symbol. This is a relatively simple way of representing what the student knows with respect a domain and it provides a way of making the problem generator adapt itself to the weaknesses of the student.

**4.6 Controlling the session**

The control over the practical exercise is mostly directed from within CLIPS, though this is not strictly true since CLIPS does not have direct control over HyperCard, rather, control is passed back and forth between the two systems. But the control initiative is directed from the CLIPS environment, where we have designed five control

[3]HyperTalk® is a registered trademark of Apple Computer, Inc.

phases for executing the practical exercise:

4.6.1 Generate a problem - Using the student model, determine the set of map symbol concepts where the student's knowledge is weakest. From this "weakness set", randomly choose a legal combination of map symbol sub-components that can be assembled for the next problem in the practical exercise.

4.6.2 Problem presentation - Generate a message from the production system to the user interface specifying the combination of map symbol sub-components to draw for the presentation. By sending a message to the user interface, control is passed to the HyperCard environment. The HyperCLIPS message is parsed by the user interface software and the appropriate map symbol sub-components are drawn on the screen. Note that the knowledge about how to draw a symbol is stored in the user interface environment rather than in the production system.

4.6.3 Interact with the student - The user interface software acts to both present the problem and elicit a response from the student. In our practical exercise we eliminate the need for the student to type in the answer by providing lists of choices from each category of map symbol sub-component. Each of the choices is actually a button that can be selected by clicking the mouse, which results in the selected phrase appearing under the map symbol (see Figure 1). The student completes the solution by clicking on the map symbol itself, which is also a button, and the next phase begins wherein the answer is analyzed.

4.6.4 Evaluate the student solution - The user interface collects the student's response to the problem and composes a message that is sent back to the production system for evaluation. Since the map symbol domain is a simple one and there is only one solution to each problem, the analysis is straightforward . The student model is updated on the basis of the knowledge displayed about each map symbol sub-component. Each correct answer boosts the mastery rating of that item in the model, while each incorrect answer lowers it. Next, a brief explanation is composed so that the student is made aware of both the correct answer and what the actual symbol is for each incorrect answer. The explanation information is composed into a message that is sent to the user interface and displayed on a "solution screen" that shows the correct identities of the symbol in the original problem as well as for the incorrect answer given by the student.

4.6.5 Cleanup - The cleanup phase is the transition point between problems that prepares the system to begin again from the generate problem phase.

## 5. Conclusions

We have argued that HyperCLIPS supports concepts in intelligent tutoring by taking advantage of the graphical user interface building capability provided by HyperCard and the knowledge-based representation and reasoning power of CLIPS. Our practical exercise system demonstrates that it is feasible to build an adaptive practical exercise for simple cognitive skills -- the map symbol recognition practical exercise is adapted on the basis of a student model that is constructed step-by-step as the student attempts to solve the problems presented by the system. It remains to be seen how effective this approach will be in more complex problem domains, but this is where much of our cur rent and future is focused. Other issues that must be addressed are: how to standardize the process of acquiring a student model, how to build a planning system to work from the student model, and how effective HyperCLIPS is for building real-time end-user systems.

## 6. References

[1] Interactive Courseware (ICW) Management and Development. TRADOC CIRCULAR 351-88-1, 30 December 1988.

[2] Burton, Richard R. and J.S. Brown. "An investigation of computer coaching for informal learning activities" . In D. Sleeman and J.S. Brown (Eds.). Intelligent Tutoring Systems (1982): 79-98. New York: Academic Press.

[3] Stevens, Albert, Allan Collins, and Sarah E. Goldin. "Misconceptions in students' understanding". In D. Sleeman and J.S. Brown (Eds.). Intelligent Tutoring Systems (1982): 13-24. New York: Academic Press.

[4] Hollan, J.D., E.L. Hutchins and L. Weitzman. "Steamer: An interactive inspectable simulation-based training system" . The AI Magazine, 5(2), 15-27.

[5] Burton, Richard R. "Diagnosing bugs in a simple procedural skill". In In D. Sleeman and J.S. Brown (Eds.). Intelligent Tutoring Systems (1982): 157-183. New York: Academic Press.

[6] VanLehn, Kurt. "Student Modeling". In Martha C. Polson and J. Jeffrey Richardson (Eds.), Foundations of Intelligent Tutoring Systems (1988): 55-78. Hillsdale, New Jersey: Lawrence Erlbaum Associates Publishers.

[7] Wilkins, David C., William J. Clancey and Bruce G. Buchanan. "Using and Evaluating Differential Modeling in Intelligent Tutoring and Apprentice Learning Systems". In Joseph Psotka, L. Dan Massey and Sharon A. Mutter, Intelligent Tutoring Systems: Lessons Learned (1988):257-275. Hillsdale, New Jersey: Lawrence Erlbaum Associates Publishers.

[8] Pickering, William B. and Randall W. Hill, Jr.. "HyperCLIPS: A HyperCard Interface to CLIPS". CLIPS User Conference, 1990, Houston, Texas.

# Developing an Intelligent Computer-Aided Trainer

Grace Hua
Computer Sciences Corporation
16511 Space Center Boulevard
Houston, Texas 77058

## Abstract

The Payload-assist module Deploys/Intelligent Computer-Aided Training (PD/ICAT) system was developed as a prototype for intelligent tutoring systems with the intention of seeing PD/ICAT evolve and produce a general ICAT architecture and development environment that can be adapted by a wide variety of training tasks. The proposed architecture is composed of a user interface, a domain expert, a training session manager, a trainee model and a training scenario generator. The PD/ICAT prototype was developed in the LISP environment. Although it has been well received by its peers and users, it could not be delivered to its end users for practical use because of specific hardware and software constraints. To facilitate delivery of PD/ICAT to its users and to prepare for a more widely accepted development and delivery environment for future ICAT applications, we have ported this training system to a UNIX workstation and adopted use of a conventional language, C, and a C-based rule-based language, CLIPS. A rapid conversion of the PD/ICAT expert system to CLIPS was possible because the knowledge was basically represented as a forward chaining rule base. The resulting CLIPS rule base has been tested successfully in other ICATs as well. Therefore, the porting effort has proven to be a positive step toward our ultimate goal of building a general purpose ICAT development environment.

## I. Introduction

A large number of academic and industrial researchers have explored the application of artificial intelligence concepts to the task of tutoring or training, but few completed systems have received adequate acceptance or been adopted for routine use because of various reasons, such as lack of a trainee model, lack of agreement on an architecture for intelligent tutoring systems and lack of an optimum delivery environment. The Software Technology Branch (STB), formerly Artificial Intelligence Section, at NASA/Johnson Space Center has developed an autonomous Intelligent Computer-Aided Trainer (ICAT) for training Mission Control Center Flight Dynamics Officers to perform payload-assist module deploys from the Space Shuttle. The purpose of this ICAT, named PD/ICAT, is to provide the transition between the fundamental knowledge acquired from textbooks and the skills required for performing successfully in large-scale integrated simulations. It is a supplement to the existing modes of training for highly skilled personnel in accomplishing complex, mission-critical tasks. Full discussions of PD/ICAT have been addressed in previous papers [1], [2].

Concurrent with the development of PD/ICAT, we have proposed an architecture together with a software environment for constructing future training systems [3]. The efficacy of this architecture and development environment is currently being tested by means of building two additional ICATs which address distinctly different training tasks at NASA. Upon refinement and completion of these systems, they will evolve into a generic architecture and a general purpose development environment which will facilitate the rapid creation and adaptation of intelligent tutoring systems in a wide variety of complex, procedural tasks.

## II. General ICAT Architecture

The ICAT architecture developed for PD/ICAT consists of five basic components, namely, user interface, domain expert, training session manager, trainee model and training scenario generator, connected by a blackboard which serves as a common factbase used for communication between the five components (Figure 1). With the exception of the trainee model, each component makes assertions to the blackboard. The user interface allows the trainee to assert actions and communicate with the ICAT. It is implemented with a windowing system. The rest of the components are rule-based and each of them examines the blackboard for information to be processed at various stages of the training.

The domain expert contains data representing knowledge in the specific training domain. It recognizes both correct knowledge and common trainee misconceptions or mistakes. The training session manager compares the assertion of the domain expert with that of the trainee to detect errors and provides the text to coach the trainee through the lesson. PD/ICAT's trainee model stores assertions made by the training session manager as a result of trainee actions. It contains a record of all trainee interactions with the training session manager, including successful accomplishments, errors and requests for help. Finally, the training scenario generator integrates data in the trainee model with a database of training scenarios associated with the domain expert to structure unique lessons for the trainee.

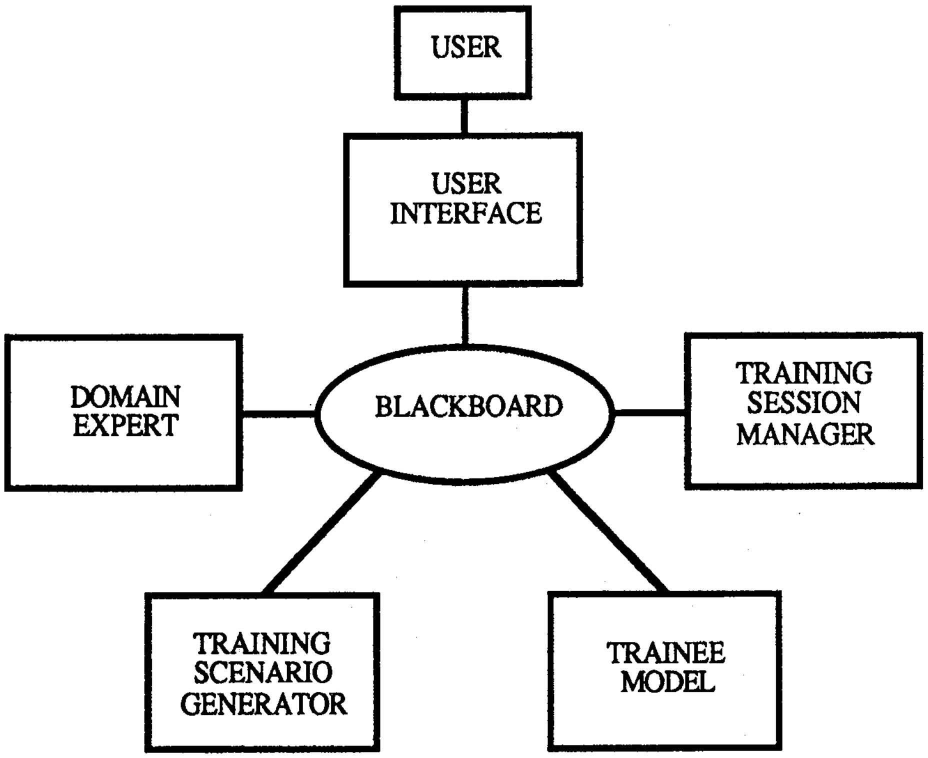


Figure 1. General ICAT Architecture

## III. Delivery of ICATs

PD/ICAT was initially developed on a LISP machine. The user interface and trainee model were written mostly in LISP while the domain expert, training session manager and part of the training scenario generator were implemented as a rule base in ART (Automated Reasoning Tool licensed by Inference Corporation). Three and a half man-years were invested by STB in this project from its inception to its maturity and acceptance by the users. Despite their acceptance, PD/ICAT users were unable to adopt this training system for regular use because they did not have the equivalent hardware and software specifications required for executing the system. Users were limited to using the same machines which STB was using for development work and were forced to be away from their normal training environment. This situation reduced the effectiveness of the training system tremendously.

At the same time, there were also the prospect of building more ICATs for various other training tasks and the potential of generating a general ICAT architecture and development environment to be considered. If each ICAT needs to be developed in its own unique hardware and software environment, the costs of the training systems will certainly be high and ICATs would be deemed infeasible for common applications. Meanwhile, it will also be impossible to generalize across several diversified ICAT environments to produce a single general purpose environment.

PD/ICAT became a typical example of the expert system delivery problem that has been inhibiting NASA's technology advancement in the artificial intelligence world [4]. It seemed inevitable that this training system would have to be ported to a conventional computer if it was to be the forerunner of many more usable ICATs for other disciplines. PD/ICAT's end users' operational environment consists of a wide variety of conventional computers. A Unix workstation was selected as the ultimate delivery environment. The decision was made to convert LISP code and ART rules to C and CLIPS, respectively, while the user interface would be rewritten in another more portable windowing package--X Window System. C and CLIPS were chosen as opposed to any commercial LISP tool delivered on workstations because of the former's availability, portability and general popularity. By choosing a highly portable environment, we improved the possibility of reusing portions of the PD/ICAT software in subsequent ICAT projects.

The scope of rehosting PD/ICAT from a LISP machine to a UNIX/C environment can be divided into three areas: user interface, C functions and the rule base, in the order of decreasing amount of effort actually required to complete the conversion. The large amount of time used in building the user interface with the X Window System was due to a learning curve involved in mastering this rather new technology. Again, here we have selected to trade start up time for portability and adaptability for future ICATs. LISP to C conversion required a major rewrite, even though the effort was facilitated, to a certain extent, by utilizing a commercial library which was able to duplicate some LISP functionality in the C programming language. LISP lists and conses were translated into strings whereas LISP structures were recreated as C structures. Over 4,000 lines of code were converted successfully in a five month period.

Interestingly, the rule base of PD/ICAT comprises the largest amount of code, yet required the least time to convert. Approximately 300 ART rules were converted to CLIPS rather smoothly in less than a month. This was made possible because rehosting PD/ICAT to a conventional computing environment was a foresight of the development team even before inception of the project. Therefore, the ART rules were already written to facilitate translation into CLIPS, utilizing forward chaining and other capabilities which can be duplicated easily in CLIPS.

## IV. Conversion of PD/ICAT Rule Base to CLIPS

The extent of modifications which need to be implemented in the PD/ICAT rules for them to function correctly in CLIPS is described here.

## 1. Language Syntax

Only a few areas of syntax differences were identified in PD/ICAT.

- **Predicate function names**
- **Math function names**
- **Multi-field functions:** Multi-field functions were, by far, the most drastic of all the syntax differences being addressed, but CLIPS string functions and multi-field functions have provided adequate means to replace all ART multi-field functions.
- **Global variables:** Global variables were used mostly, but not exclusively, for salience declaration. They were replaced by constant values.

## 2. LISP Data Types

There were over 40 external functions interfacing with CLIPS in PD/ICAT. Aside from rewriting these functions in C, there were additional considerations in passing data from CLIPS to these external functions when LISP data types were involved. The two major data types we have encountered in PD/ICAT were lists and keywords.

- **Passing list constructs to C**

  In PD/ICAT, since list constructs in the external functions were being retained by utilizing the commercial LISP-C library, this governed the method which we chose to convert the list data type used in the CLIPS rules. A list on the right hand side of a rule can be constructed by using the *str-cat* function to concatenate the elements. For example,

  *(list 'm50 ?dvx ?dvy ?dvz)* in ART

  will translate to

  *(str-cat m50 " " ?dvx " " ?dvy " " ?dvz)* in CLIPS.

  By using the LISP-C *list* function in the external function, *list(rstring(1)),* a list look-alike construct

  *"(m50 2.3308 -0.1968 0.8822)"*

  will be returned. However, unless the external function needs to perform additional LISP-like functions on the list, string representations of floating point numbers will have to be converted back to float type.

  If a list construct is not required in the external function, list elements in the rule can be appended into a multi-field value by using the *mv-append* function. For example,

  *(list 'm50 ?dvx ?dvy ?dvz)* in ART

  will translate to

  *(mv-append m50 ?dvx ?dvy ?dvz)* in CLIPS.

Inversely, if the external function expects a list construct from CLIPS, as in most cases in PD/ICAT, and the rule contains a multi-field variable, then the multi-field variable will need to be converted to a string. For example,

*(list$ ?dv)* in ART

will translate to

*(str-implode $?dv)* in CLIPS.

Then, again, the LISP-C *list* function can be used to return the list construct.

- **Passing keyword arguments to C**

  Two options were considered in converting keyword arguments. The first one retains the original ART syntax, for example,

  *(reset-error-counts :total t :errors nil).*

  In this case, it is the external function's responsibility to first identify the keyword and then to access the next field as its argument.

  The second alternative will require a change in the syntax of the function call:

  *(reset-error-counts ":total t" ":errors nil").*

  In this case, since the argument is concatenated to the keyword, it ensures that the argument is always in the proper place and simplifies the checking process in the external function. The previous method allows keyword arguments of all valid CLIPS values whereas the second one requires all arguments to be already in string format or converted to string type first. For PD/ICAT, because all keyword arguments are either words or strings, we opted for the second method.

### 3. Programming Style

With the exception of syntax modification and LISP data type conversion, none of the PD/ICAT rules were changed when we first attempted to test them in CLIPS. But we soon noticed that in two instances, differences between the CLIPS inference engine and ART inference engine were apparently significant enough to cause rules to be activated in a different sequence, and, consequently, result in unexpected and undesirable firing sequences which led to malfunctions in the training system. A closer look at the rules actually pointed to deficiencies in two of them where patterns on the left hand side were not sufficiently constrained for the desired activation to occur, or not to occur. Once the problems were identified, they were quickly resolved by adding constraints in the rules. Programming discrepancies such as these had not manifested themselves before; their occurrences in the CLIPS environment have actually helped us restructure the PD/ICAT rule base.

## V. Results of Conversion

PD/ICAT's rule-based expert system has been running on the UNIX workstation in the C and CLIPS environment for some time already and we have made several positive observations. First of all, the rule base is being stored in compiled format. Even though it has to be loaded into the training environment every time a lesson begins, the comparatively small size of the binary files (compiled rules together with the ICAT executable) and short program load time are much more desirable than having to save an entire ICAT environment in a LISP world or the longer program load time experienced on the LISP machine. Secondly, because the user interface and expert system can be detached from each other easily, debugging the rule base by using the CLIPS debugging tools is facilitated. Thirdly, interruptions from garbage collection have been eliminated. Although ICATs are not real-time processes, frequent occurrences of garbage collection in the previous ICAT environment did have an adverse effect on the trainee, whose interest is to concentrate on the training domain rather than battling the system.

Finally, we are one step closer to our goal of adapting the ICAT expert system for other training tasks. We have already successfully integrated PD/ICAT's training session manager and trainee model with a second ICAT which STB is developing for Kennedy Space Center in training System Engineers to perform tests on various subsystems of the Space Shuttle's main propulsion system. The delivery vehicle selected for this project is also a UNIX workstation. By using the general architecture, little work is needed to adapt PD/ICAT's training session manager and trainee model to this ICAT. So the development effort can be focused on the knowledge domain and user interface. More recently, a third ICAT is in the process of adopting the same architecture and ICAT expert system. While part of PD/ICAT's rule base is being reused by these ICATs, these later development efforts also serve to refine the proposed architecture and expert system into a generic training architecture for a broader spectrum of training domains.

## VI. Conclusion

Porting the ICAT expert system from LISP to C was a wise move. The cost was minimal compared to the long term benefits. However, establishing a generic training session manager and trainee model does not constitute all the solutions to creating a general purpose ICAT development environment. Most of us in the expert system development world recognize that knowledge acquisition and knowledge representation can be major stumbling blocks to quick turnarounds of usable applications. This is true for ICATs as well. To date, we are still in the primitive stages of testing and building tools for capturing expert knowledge and translating it into a representation which can be integrated with ICATs. When these tools approach maturity, the production of a general purpose ICAT development environment will be another step closer to reality.

## VII. References

1. Loftin, R. B., Wang, L., Baffes, P., Hua, G., "An Intelligent Training System for Space Shuttle Flight Controllers", Telematics and Informatics, 5(3):151-161.

2. Loftin, R. B., Wang, L., Baffes, P., Hua, G., "An Intelligent Training System for Space Shuttle Flight Controllers", Innovative Applications of Artificial Intelligence, H. Schorr and A. Rappaport, eds., AAAI Press, Menlo Park, CA, 1989.

3. Loftin, R. B., "A General Architecture for Intelligent Training Systems", National Aeronautics and Space Administration (NASA)/American Society for Engineering Education (ASEE) Summer Faculty Fellowship Program--1987, Final Reports, Volume 2, W. B. Jones, Jr. and S. H. Goldstein, eds., Johnson Space Center, Houston, TX, November 1987, 19:1-15.

4. Culbert, C., Riley, G., Savely, R. T., Giarratano, J., "A Solution to the Expert System Delivery Problem", Proceedings of the ISA 88 International Conference and Exhibit, Houston, TX, October 16-21, 1988, 1663-1670.

# INCORPORATING CLIPS INTO A PERSONAL-COMPUTER-BASED INTELLIGENT TUTORING SYSTEM

Stephen J. Mueller
Computer Sciences Corporation
16511 Space Center Blvd.
Houston, Tx 77058

## ABSTRACT

A large number of Intelligent Tutoring Systems (ITSs) have been built since they were first proposed in the early 1970's. Research conducted on the use of the best of these systems has demonstrated their effectiveness in tutoring in selected domains. Computer Sciences Corporation, Applied Technology Division, Houston Operations has been tasked by the Spacecraft Software Division at NASA/Johnson Space Center (NASA/JSC) to develop a number of ITSs in a variety of domains and on many different platforms. This paper will address issues facing the development of an ITS on a personal computer using the CLIPS (C Language Integrated Production System) language.

For an ITS to be widely accepted, not only must it be effective, flexible, and very responsive, it must also be capable of functioning on readily available computers. There are many issues to consider when using CLIPS to develop an ITS on a personal computer. Some of these issues are: when to use CLIPS and when to use a procedural language such as C, how to maximize speed and minimize memory usage, and how to decrease the time required to load your rule base once you are ready to deliver the system. Based on experiences in developing the CLIPS Intelligent Tutoring System (CLIPSITS) on an IBM PC clone and an intelligent Physics Tutor on a Macintosh II, this paper reports results on how to address some of these issues. It also suggests approaches for maintaining a powerful learning environment while delivering robust performance within the speed and memory constraints of the personal computer.

## ITS ARCHITECTURE CONSIDERATIONS

Assuming that we are only considering using the languages C and CLIPS to build an ITS, there are two basic approaches which can be taken when designing an ITS architecture. The main driving program can be written in CLIPS with calls to external C functions to handle interface and miscellaneous issues, or the main driving program can be written in C with a call to the CLIPS expert system to handle the tutoring. Let's discuss some of the advantages and disadvantages of these two approaches.

Intelligent Tutoring Systems which are built entirely in CLIPS, with the exception of calls to external C functions, are very nice for applications involving a single procedure to be tutored. The rules are very specific and check for particular student actions and particular errors. Rule-based code also tends to have more generic modules than C-based systems and is therefore more readily reused by other ITS developments. The programmer of CLIPS-based systems must be cautioned, however, against an overabundance of C function calls. External C function calls, especially those used in left hand side patterns, tend to be very slow. Since speed is very critical in an ITS, careful consideration should be given to the overall design.

Systems which have the main program written in C tend to have more advantages than CLIPS initiated ones. Initialization, rule loading, resetting, and other procedural operations can be performed in the C code before the CLIPS code is called. Once CLIPS returns control to the main program, the CLIPS fact list can easily be reset for the next problem. This technique allows the procedural operations to be handled by a procedural language and the non-procedural operations to be handled by CLIPS. Systems designed using this approach are very effective for handling multi-problem/procedure domains. A multi-problem ITS, such as an algebra tutor which would ask a student to solve a number of distinct problems, would consist mainly of generic rules. The generic rules would provide the basic knowledge about how to solve algebra and specific problem description facts would exist to define the current problem being worked. The expert system would pass its results to the main program by calling external functions.

## CHOOSING THE APPROPRIATE LANGUAGE

It can be argued that there is no single architecture or development language that would be ideal for every type of ITS. The architecture and the languages used to code each module are dependent upon the type of application, the hardware and operating system used, and the speed and memory requirements of the system.

An issue to consider before designing an ITS is whether or not there is a good reason to use a rule-based language. If the expert knowledge can be just as easily represented in C code as it can in CLIPS, then it is probably best not to involve the overhead associated with CLIPS. Rule-based systems tend to be easier to write but slower to execute than procedural systems. Rule-based code is also very powerful for checking multiple solutions paths at a time. An ITS whose main goal is to tutor the student in some procedure usually allows several optional steps or paths at any point in time. Rules can easily determine which path the student is following with minimal coding. Determining which type of language, rule-based or procedural, to use when building an ITS can be very difficult. Most ITS development is a hybrid of procedural and rule-based languages. The procedural code typically provides the graphics and student input capabilities, and the rule-based code provides the domain knowledge and student feedback and modeling capabilities. Often times the language used to write one module will dictate the language used to write another. For example, if you choose to keep the student model facts in the fact-list, then the error handling code will most likely also be written in CLIPS. However, if you choose to dump this student data into a C data file, then your error handler would probably be written in C.

## PERFORMANCE

Another issue to consider when building an ITS on a personal computer is finding a balance between speed and memory requirements. There are several factors which affect the speed of a system. The number of rules, facts, and the arrangement and type of rule patterns all affect speed. Probably the single most important factor is the number of facts in the fact list. In one recent experiment, 42 specific rules were reduced down to 6 generic rules and instead of the system running faster it actually ran slower. This was because the number of facts necessary to support generic rules was greater and the number of partial rule matches increased.

Speed can also be improved by minimizing the use of the test function and multi-field variables. There is a large overhead associated with these as well as external C function calls on the left hand side of the rule.

Finally, the ordering of the patterns on the left hand side of a rule can also be arranged to maximize speed or minimize memory usage. Consider the following two sample rules:

```
#1  (defrule Speed            #2  (defrule Memory
         (a)                               (c)
         (b)                               (a)
         (c)                               (b)
   =>                                =>
         ... )                             ... )
```

Assume that the fact list initially contains the facts (a) and (b). As soon as the fact (c) is asserted, sample #1 would be activated faster than sample #2. This is because in sample #1, the instantiations for patterns (a) and (b) were already completed before pattern (c) was matched. Therefore, there was only one pattern in the rule remaining to be instantiated. In sample #2, however, there were initially no patterns instantiated since CLIPS stops as soon as it finds a pattern it can't instantiate, namely pattern (c). As soon as fact (c) appears on the fact list, then all three patterns are instantiated and the rule activates. The key, then, is if you want to maximize for speed, place all your static facts as high on the left hand side as possible. Complete as many partial rule matches as possible at a time which won't be as noticeable to the student (in the case of an ITS) so that the rule is ready to fire as soon as it finds the one or so remaining facts. The best time for this is in between problems or while the student is reading text on the screen.

The method for minimizing memory usage is just the opposite of the method for maximizing speed performance. To minimize memory usage, avoid as many partial instantiations as possible. Place your static patterns near the bottom on the left hand side and arrange your rules so that your most critical pattern is at the top of your patterns to be matched. In this way, once your critical pattern is matched, the other patterns in the rule should also be matched and very few partial memory eating matches will occur.

Typically the rules of an ITS will represent a compromise between speed and memory efficiency. The system should be tuned so that the rules associated with student actions which require immediate response are speed efficient and the others are memory efficient. Obviously, the more memory you have available to your application, the more attention you can pay to optimizing your rules for speed.

## THE DELIVERY VERSION

The final delivered version of the ITS should be as simple as possible for the student to execute. Forcing a non programmer to enter CLIPS commands to load rules, reset and run can be very frustrating. The time spent waiting for the rules to load can also be rather annoying. There are several options available. The first is to start CLIPS running and use the "bload" command instead of the "load_rules" command to load your rules into the system. If your rules have already been saved in binary form using "bsave", loading will be speeded up tremendously. The second approach is to create a CLIPS run-time program. To do this you would need to create a run-time version of the basic CLIPS language, compile the CLIPS rules and any necessary C code and link these together. This produces a single executable program name for the student to enter and performs the reset and run commands for him. The third approach is to compile the CLIPS source together with your C source code and create a single executable. From within your C code you would issue the "bload" command to load your previously generated binary rule base or the "load_rules" command if your rules have not been converted to binary form using the "bsave" command.

Probably the overall best approach would be the second. The student would have to issue only one command to activate the tutor and it will get the tutor up and running faster than any of the others. Close behind, though, is the third approach using the "bload" capability. This approach activates the tutor a little more slowly but it relieves the programmer of having to build a special run-time version of CLIPS and of having to compile the rules into many separate files. Typically this approach would be used during prototyping stages and building a run-time executable would be used for the final delivery. Regardless of the approach taken for loading rules and initiating the program execution, the performance of the ITS will be the same.

## SUMMARY

The design of an ITS should take into consideration the application, hardware, and performance requirements of the system. If the application tutors only a single procedure, then the easiest approach may be to write the entire system in CLIPS, with the exception of the necessary external calls to C for user interface and miscellaneous capabilities. Care should be taken, though, that a response to a student action occurs quickly enough to avoid frustration and confusion by the student. If the ITS runs too slowly, given the memory and speed constraints of the personal computer, then the rules must be optimized for speed and/or portions of the ITS should be rewritten in C. If the application tutors the student in performing a number of related problems, then the best approach is to embed CLIPS within a C main program and allow C to handle all the housekeeping duties before and after each problem.

Rules can be optimized either for speed or memory. To optimize for speed, place your least changing and static patterns at the top on the left hand side of the rules. Try to arrange your rule patterns so that when a student input is received, all the other patterns in the rule being activated have already been matched and the only one remaining is the pattern associated with the student input. To optimize for memory, do just the opposite. Inhibit as many partial rule matches as possible until all the facts you need are in the fact list. Creating partial instantiation networks consumes memory.

Finally, when delivering the ITS for use, its best to provide the student with a single command to initiate the tutor. This can be accomplished by building a CLIPS run-time executable or by compiling CLIPS with your C code and using the "load_rules" or "bload" functions. Either approach is an effective way of decoupling the student from the implementation details.

## A2 Session:
## Intelligent Software Engineering

# Hardware Independence Checkout Software

*Barry W. Cameron*
*H. R. Helbig*

Advanced Computing Solutions, Inc.
17049 El Camino Real, Ste. 202
Houston, TX 77058

*ABSTRACT*

ACSI has developed a program utilizing CLIPS to assess compliance with various programming standards. Essentially the program parses C code to extract the name of all function calls. These are asserted as CLIPS facts which also include information about line numbers, source file names, and called functions. Rules have been devised to establish functions called that have not been defined in any of the source parsed. These are compared against lists of standards (represented as facts) using rules that check intersections and/or unions of these. By piping the output into other processes the source is appropriately commented by generating and executing sed scripts.

# Hardware Independence Checkout Software

***Barry W. Cameron***
***H. R. Helbig***

Advanced Computing Solutions, Inc.
17049 El Camino Real, Ste. 202
Houston, TX 77058

Apart from addressing functionality, all software engineering projects, given the continually increasing performance of manufacturer's available offerings, must also address design constraints derived from consideration of the developed software's future portability. In particular, the rise of UNIX$^{TM}$ as an operating system of choice running in a distributed, heterogeneous, networked environment has served to underscore the industry wide role of formal standards. Furthermore, the recent increase in software development productivity, coupled with the complexity of design required within current and proposed software engineering projects, has helped lead to an industry recognition that the issues involving portability need to be addressed during the original project design phase. If portability is not addressed as a design issue, which must include consideration of existing or emerging standards, later adaptation of resulting products or projects is likely to give rise to cost overruns. In short, emerging standards coupled with decreased development times and increased complexity have elevated portability to an organization-wide issue rather than its traditional consideration as a minor development constraint within systems and programming divisions. This new focus has, at all organizational levels, intensified the awareness of problems associated with formally verifying that developed software is actually in compliance with those emerging standards.

Formerly, above the programmer organizational level, such verification and compliance with design standards has been primarily based on assurances that established software engineering practices would somehow minimize portability issues. While such assurances, given certain hardware and limited communications topology, have worked to some degree, with the advent of open systems, new communications protocols and emerging standards that address multiple topologies and architectures, and, since there have been no tools available to verify that code developed is in fact compliant to existing standards or to determine to what degree it is in variance from those standards, UNIX$^{TM}$ software developers have been forced to opt for a "best guess" heuristic to make any portability estimation and, consequently, any estimation of future port costs.

Today, requirements that derive from portability considerations are of constant concern to all programmers. However, considering the frequency of definitions and redefinitions of standards that have occurred in the industry recently, as evidenced by the release of UI's Open Look, OSF's Motif, AT&T's SVID, Release 4, the progress of the various POSIX committees, etc., the goal has become one of writing code that ever more closely approaches portability and is one that necessarily has to involve continually evolving methods. Advanced Computing Solutions, Inc. (ACSI) was tasked by IBM on behalf of NASA to look at promoting and supporting portability for large UNIX$^{TM}$ programming projects in support of the NASA Mission Control Center Upgrade project (MCCU). The code for this upgrade was to be written in C intended to run on UNIX$^{TM}$ based computers. Part of the ACSI assignment was to develop tools to aid in insuring source level portability, which would address the issues of hardware independence through verification against some standard of same. This effort would, of course, primarily focus on the set of existing and emerging standards regarding UNIX$^{TM}$.

The status of standards efforts to define UNIX$^{TM}$ which were relevant to this project in the summer of 1989 was that IEEE had published the POSIX 1003.1 standard and had released a number of draft standards which included definitions for real time extensions, System Administration, Networking, Security, etc.; AT&T had defined SVID for Release 3; the Berkeley versions of UNIX$^{TM}$ provided the only practical network interfaces that were commercially available, and ANSI C was, practically speaking, defined, but only a few compilers were available and existed on a limited number of platforms, and, even as of this writing, not yet officially released. Real time extensions to UNIX$^{TM}$ were available from Masscomp (Concurrent) but were neither compliant nor compatible with the POSIX real time draft model. /usr/group and X-Open had defined general directions for the formal standards efforts, but there was no guarantee that those would be the directions taken. X-windows was available from MIT, but no window tool kits or window management systems had yet reached any kind of "standard" status[2, 3, 4, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16].

Since standards definitions were not available to address the entire functionality implied from the MCCU requirements[5], but each standard that did exist or was proposed addressed a particular aspect or layer of functionality, we decided that the most practical tool that could be made available was one that could easily be updated as these standards were further refined and others defined and one that could support and promote a modularity and layering of software to be supported reflective of these individual standards. This would enforce a software design that would modularize those areas that were not clearly defined leaving the majority of code relatively portable and the rest able to be upgraded modularly with modules corresponding to functional areas to be addressed by standards definition efforts. Another important and driving requirement of the ACSI task was the need to implement the software within a very constrained time and dollar budget.

One approach to standards verification would be to ensure that specific implementations of defined standards are correct and test suites to verify compliance with ANSI C, POSIX, and SVID were either available or in work at that time[1]. Another approach toward verification would be to ensure syntactic compliance with those same standards and ANSI C compilers, available from gnu, could check for syntactical compliance with the language standard, but could not verify that any system supplied functions were even defined by standards. We did not want to duplicate any of these efforts and certainly were not about to delve into the realm of code semantics. What we felt was possible was the development of a system that would determine what functions were called in a set of source files, and check to see if those were valid functions defined by standards or allowed by the specific program design, *i.e.*, allowed interfaces to lower level objects. We believed that we could implement, through a DBMS or an expert system, some method to coordinate known problem functions with some action, either the display of a message to indicate a correction, or an update to the code, or simply setting a flag indicating the function was invalid. ACSI was quite familiar with CLIPS and had previously developed a parser, for reverse engineering purposes, that could follow the calling structure of a C program. It was felt that CLIPS could provide a means of making fast comparisons to standards and a means of quickly updating standards definitions. Furthermore, we believed that the parser, after modification to output CLIPS rules and facts, could be used to identify functions used within a particular program. By devising an appropriate set of rules, different actions could be taken depending on what function was encountered and where in the source it was encountered. It was felt that this combination would provide a practical approach to supporting both software modularity and the need for frequent updates to reflect the constantly changing standards arena to support portability.

For the actual implementation (see Figure 1), the preexisting parser was modified to output, along with the calling sequence, the line number and source file name where the function was encountered. The parser output was modified to be a program description in the form of CLIPS factswhich defined what additional functions each encountered function called. This allowed unresolved functions to be identified by a single simple rule. By defining standards as sets of deffacts, and writing rules to make comparisons to different sets of standards, either as intersections, or unions of standards the required functionality was built. Further modification to the parser allowed it to identify all internally defined functions which could then be manually modified to reflect the designed visibility rules for a particular project and, therefore, support its designed modularity. The resultant deliverable, called HICS (Hardware Independence Check-out Software) has been baselined as the portability and verification tool for the MCCU project. All C code developed for the MCCU is required to first pass the HICS verifier.

For the commercial product, ZEUS, further refinements were made to provide a working environment for UNIX$^{TM}$ code development based on the standard UNIX$^{TM}$ concepts of user and group, with the addition of project and module. This allowed specific configurations of rules to be implemented for each level of a project while providing standard UNIX$^{TM}$ security and support for multiple projects. By writing a few more CLIPS code generators which automatically generate deffacts (to define standards) and rules (to take predefined actions such as commenting source code, printing messages to the screen, and taking default actions if a function is not identified), a menu driven interface was written that allowed a developer to set up the entire environment (the actions, associated messages, priority of the actions, as well as standards definitions and updates) with no knowledge at all of CLIPS.

In the current interface to CLIPS the rules and facts are all written to files and a script to load and run them is written at run-time. CLIPS is then called as a system call using the -f option with its output funneled through various filters which are used to mediate CLIPS messages, output sed (stream editor) scripts that can be used to directly update source code at the place the function was found, or print messages informing a programmer what action he should take to achieve a higher degree of portability. Figure 2 shows a code fragment that has been analyzed for compliance with the intersection of POSIX and SVID definitions.

ZEUS is under continuous further development. Using CLIPS as the communication link between the standards definitions and the parser output has distinct advantages toward furthering the product capabilities. Any other parsers, which might be designed for any specific portability checks (*e.g.* arguments to specific functions) could be readily integrated into the product. Rule sets can easily be devised and added to check for compliance with anything that can be defined in terms of CLIPS facts.

# ZEUS
## Portability Software

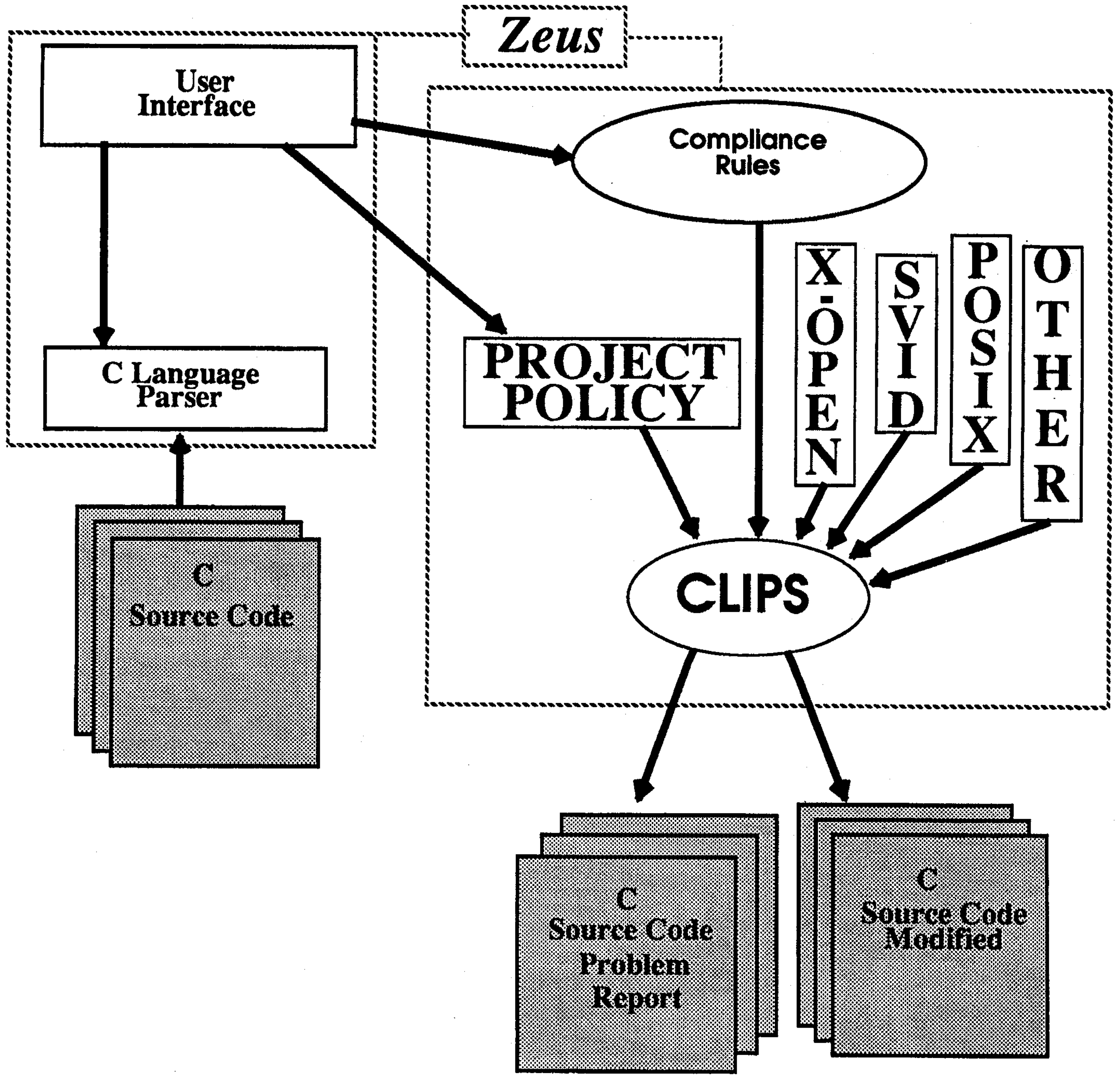


Figure 1

## ZEUS
## Portability Software

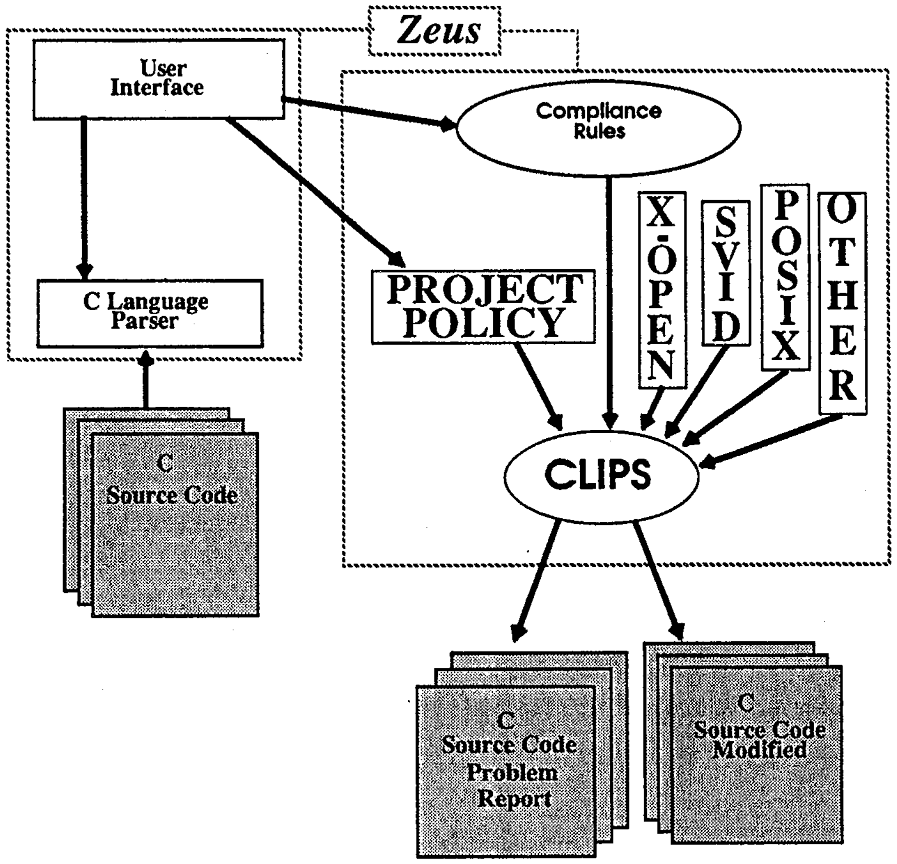


Figure 1

# Analyzed Code Fragment

```
 file_name = tmpnam(NULL);
 file = fopen ( file_name, "w" );
 if ( file == NULL )
 {
err_log (
"PERMISSION: UNIX error while computing a mode change. errno = %d",
errno, 0 );
return(0);
 }
 fclose ( file );

 /*********************************************************************
Change the mode of the file to that required by the input
arguments. Then read the modes to see what the result is.
 *********************************************************************/

 sprintf ( command, "chmod %o %s; chmod %s %s; ls -l %s; rm -f %s",
mode, file_name, changes, file_name, file_name, file_name );
/* @@@ ZEUS @@@ popen is allowed but not POSIX */
 pipe = popen ( command, "r" );
 if ( pipe == NULL )
 {
err_log (
"PERMISSION: UNIX error while computing a mode change. errno = %d",
errno, 0 );
return(0);
 }
 fgets ( dir_line, sizeof(dir_line), pipe );
/* @@@ ZEUS @@@ pclose is allowed but not POSIX */
 pclose ( pipe );

 /*********************************************************************
Convert the result back to an integer.
 *********************************************************************/

 mode = rwx_to_perm ( dir_line );
```

Figure 2

## Analyzed Code Fragment

```
 file_name = tmpnam(NULL);
 file = fopen ( file_name, "w" );
 if ( file == NULL )
 {
err_log (
"PERMISSION: UNIX error while computing a mode change. errno = %d",
errno, 0 );
return(0);
 }
 fclose ( file );

 /**********************************************************************
Change the mode of the file to that required by the input
arguments. Then read the modes to see what the result is.
 *********************************************************************/

 sprintf ( command, "chmod %o %s; chmod %s %s; ls -l %s; rm -f %s",
mode, file_name, changes, file_name, file_name, file_name );
/* @@@ ZEUS @@@ popen is allowed but not POSIX */
 pipe = popen ( command, "r" );
 if ( pipe == NULL )
 {
err_log (
"PERMISSION: UNIX error while computing a mode change. errno = %d",
errno, 0 );
return(0);
 }
 fgets ( dir_line, sizeof(dir_line), pipe );
/* @@@ ZEUS @@@ pclose is allowed but not POSIX */
 pclose ( pipe );

 /**********************************************************************
Convert the result back to an integer.
 *********************************************************************/

 mode = rwx_to_perm ( dir_line );
```

Figure 2

## REFERENCES

1. American National Standards Committee, **Draft Proposed ANSI Programmming Language C**, Obtained from Global Engineering Documents (800) 854- 7179.

2. American Telephone & Telegraph, **System V Interface Definition, Issue 2**, Vol. 1, 1986.

3. American Telephone & Telegraph, **System V Interface Definition, Issue 2**, Vol. 2, 1986.

4. American Telephone & Telegraph, **System V Interface Definition, Issue 2**, Vol. 3, 1986.

5. Dellenback, Steven W., Collier, Mark D., Knipp, Andrew R., Martin, Nancy L., Southwest Research Institute, **Hardware Independent Software Development Environment for MCC**, December 15, 1988.

6. IEEE Technical Committee on Operating Systems, **POSIX 1003.3, Draft 7, Standard for Test Methods for Measuring Conformance to POSIX 1003.1**, Institute of Electrical and Electron- ics Engineers, Inc., New York, NY, Aug. 2, 1988.

7. IEEE Technical Committee on Operating Systems, **POSIX 1003.2, Draft 8, Shell and Application Utility Interface**, Institute of Electrical and Electronics Engineers, Inc., New York, NY, Dec. 5, 1988.

8. IEEE Technical Committee on Operating Systems, **POSIX 1003.1, Portable Operating Systems Interface**, Institute of Electrical and Elec- tronics Engineers, Inc., New York, NY, Sep. 30, 1988.

9. IEEE Technical Committee on Operating Systems, **POSIX 1003.4, Draft 6, Realtime Extension for Portable Operating Systems**, Institute of Electrical and Electronics Engineers, Inc., New York, NY, Feb. 10, 1989.

10. X/Open Company, Ltd., **X/Open Portability Guide, Vol. 1, XSI Commands and Utilites**, Prentice Hall, Englewood Cliffs,, Dec. 1988.

11. X/Open Company, Ltd., **X/Open Portability Guide, Vol. 2, XSI System Interface and Headers**, Prentice Hall, Englewood Cliffs,, Dec. 1988.

12. X/Open Company, Ltd., **X/Open Portability Guide, Vol. 3, XSI Supplementary Definitions**, Prentice Hall, Englewood Cliffs,, Dec. 1988.

13. X/Open Company, Ltd., **X/Open Portability Guide, Vol. 4, Programming Languages**, Prentice Hall, Englewood Cliffs,, Dec. 1988.

14. X/Open Company, Ltd., **X/Open Portability Guide, Vol. 5, Data Management**, Prentice Hall, Englewood Cliffs,, Dec. 1988.

15. X/Open Company, Ltd., **X/Open Portability Guide, Vol. 6, X-Window System**, Prentice Hall, Englewood Cliffs,, Dec. 1988.

16. X/Open Company, Ltd., **X/Open Portability Guide, Vol. 7, Networking Services**, Prentice Hall, Englewood Cliffs,, Dec. 1988.

# AUTOMATING SYMBOLIC ANALYSIS WITH CLIPS

By

Keith E. Morris
Rockwell International Corporation
Space Transportation Systems Division
12214 Lakewood Boulevard
Downey, California 90241

## ABSTRACT

Symbolic Analysis is a methodology first applied as an aid in selecting and generating test cases for "white box" type testing of computer software programs [Reference 1]. The feasibility of automating this analysis process has recently been demonstrated through the development of a CLIPS-based prototype tool. Symbolic analysis is based on separating the logic flow diagram of a computer program into its basic elements, and then systematically examining those elements and their relationships to provide a detailed static analysis of the process that those diagrams represent. The basic logic flow diagram elements are flow structure (connections), predicates (decisions), and computations (actions).

The symbolic analysis approach supplies a disciplined step-by-step process to identify all executable program paths and produce a truth table that defines the input and output domains for each path identified. The resulting truth table is the tool that allows software test cases to be generated in a comprehensive manner to achieve total program path, input domain, and output domain coverage.

Since the manual application of symbolic analysis is extremely labor intensive and is itself error prone, automation of the process is highly desirable. Earlier attempts at automation, utilizing conventional software approaches, had only limited success.

This paper briefly describes the automation problems, the symbolic analysis expert's problem solving heuristics, and the implementation of those heuristics as a CLIPS based prototype, and the manual augmentation required. A simple application example is also provided for illustration purposes. The paper concludes with a discussion of implementation experiences, automation limitations, usage experiences, and future development suggestions.

## INTRODUCTION

A technique, referred to herein as symbolic analysis, has been successfully applied to NASA's Space Shuttle flight software for assistance in the selection and generation of test cases for software white box verification testing [Reference 1]. The feasibility of automating this analysis process has been demonstrated through the development of a prototype tool using CLIPS [Reference 2]. CLIPS, the C Language Integrated Production System, may be obtained for non-government work from COSMIC for a nominal fee.

The commonly accepted minimal test coverage criteria for white box testing require that each program statement and control flow branch be exercised at least once during the testing process. The symbolic analysis methodology identifies test cases that satisfy this condition.

The symbolic analysis approach supplies a disciplined step-by-step process to identify all executable program paths and produce a truth table that defines the input and output domains for each path identified. This truth table is the tool that allows software test cases to be generated in a comprehensive manner and for many programs to achieve total program path, input domain, and output domain coverage. A more complete description of the symbolic analysis methodology is contained in Reference 1. A summary of the basic steps is as follows:

1. Identify all possible program paths
2. Identify all path input domains
3. Identify all executable paths
4. Identify all path output domains
5. Construct a program truth table

Although the symbolic analysis methodology is comprehensive, its manual application can be tedious, labor-intensive, and error-prone. It also requires total rework whenever the tested program is modified. To overcome these problems, early, unsuccessful attempts were made to automate the analysis process by using conventional programming techniques. It was only with the availability of expert systems techniques and tools (list processing, symbolic computation, heuristic procedures, and forward chaining) that many of the problems associated with conventional programming approaches could be solved and more complete automation achieved.

For each symbolic analysis process step, the associated problem, the expert practitioner's heuristics, the CLIPS automation, and the required manual augmentation will be discussed and demonstrated with a simple example. Though simple, the example is representative of a large class of real-world programs. Additional complications arise with the use of loops, local variables set by previous entries, time dependency, computed and multiple variable predicates, etc. Although these complications are manually resolvable, their automation has not yet been attempted by the author. The CLIPS rules shown have been simplified from those actually contained in the prototype, by eliminating path counting, fact cleanup, terminal display, printing, and input/output file management.

## STEP 1. IDENTIFY ALL POSSIBLE PROGRAM PATHS

### The Problem

One of the easiest ways to approach the symbolic analysis of a program is to transform it into a directed graph form. This can often be done with tools that automatically generate flow charts from a program source code. The program flow chart shown in Fig. 1, with edge labeling a,b,...,p, is an acceptable flow chart and represents the example program that will be used throughout this paper.

The flow chart describes the program structure, the predicates (branch conditions), and the computations performed; the labels allow the individual program elements and paths to be uniquely identified.

The problem is to generate a list of all possible paths through the program from beginning to end, identifying each path, P {i}, by the set of edges, {i}, that it contains; e.g., path P (abcdijkp) contains the edges abcdijkp. Since the relationship between the number of possible paths and the number of branches varies from linear to exponential, identifying a complete list of all possible paths is generally not a small task. Although the path identification concept is simple, the more skilled symbolic analysis practitioners greatly outperform the average. It is also not uncommon for even an expert to discover at a later step that a possible path was overlooked.

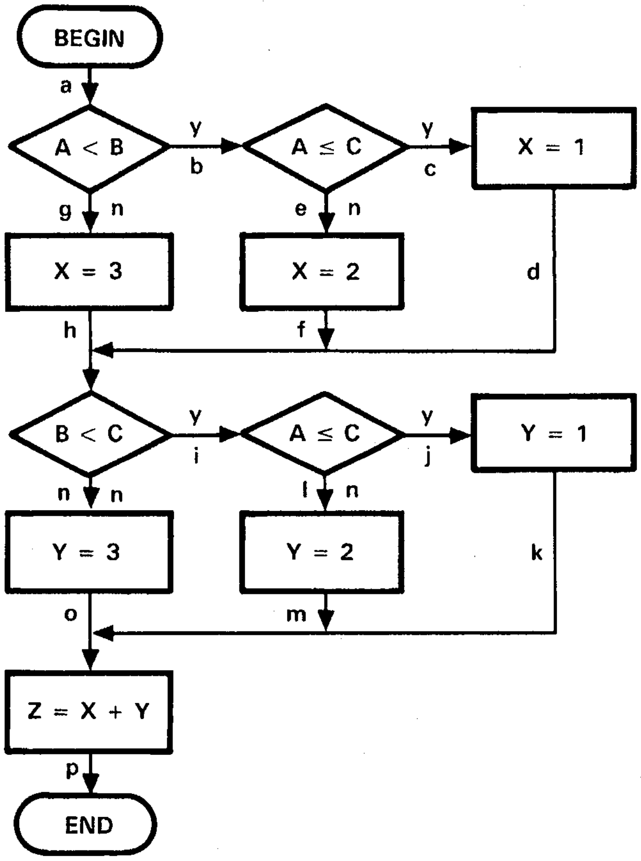


*Fig. 1. Flow Chart With Edge Labeling. The flow chart describes the program; labels allow individual element and path identification.*

## The Expert's Heuristics

The only elements that an expert analyst needs to accomplish this step are the labeled logic flow diagram and a medium to record the results. The expert's heuristics can be summarized as follows: (1) traversing the logic flow chart (Fig. 1) from top to bottom, left to right, in a systematic manner, convert the flow chart into a directed binary tree (Fig. 2); (2) using the tree, list all paths with terminating leaves.

## The CLIPS Automation

The approach is to use facts about the program structure to generate a set of program path-edge facts which have a one-to-one correspondence to the set of all possible program paths. Each path-edge fact will contain a string of elements denoting its edges, e.g., (path '1st-edge' '2nd-edge', ... 'last-edge'). These path-edge facts will serve as the basis for the subsequent symbolic analysis steps.

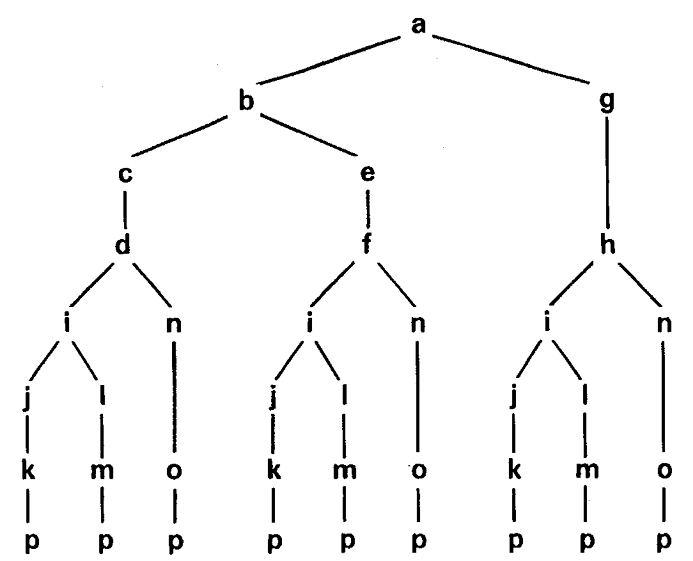


*Fig. 2. Directed Binary Tree of All Possible Paths. All possible program paths can be shown as a directed binary tree.*

To enable the set of all possible program path-edge facts to be generated, the logic flow diagram structure must be converted (manually or by another program) into CLIPS edge-connection facts of the form

```
(deffacts edge-connections
    (connects begin '1st-edge')
    (connects '1st-edge' '2nd-edge')
    . . .
    (connects 'nth-edge' end))
```

Rules working on these edge-connection facts can generate path-edge facts and test these facts for solutions and solution coverage. The process is similar to the expert's heuristics: Create path-segment facts; start each segment with the beginning edge and add successive edges to it, one at a time. Test each path-segment after each edge is added to see if it contains the ending edge; if it does, change the path-segment fact into a path-edge fact and record it for later use.

The CLIPS rules generate path-segment-edge facts of the form

```
(segment '1st-edge' '2nd-edge' ....'mth-edge')
```

and when the ending edge is encountered, the path-segment-edge facts are converted to path-edge facts of the form

```
(path '1st-edge' '2nd-edge' ....'ending-edge')
```

Note: the path-segment facts and path facts do not include "begin" or "end" as they are not edges.

The CLIPS path generation rules are as follows:

```
(defrule initialize
     ?start <- (initial-fact)
     (connects begin ?to $?more)
 =>
     (retract ?start)
     (assert (segment ?to $?more)))

(defrule create-partial-path
     (segment $?path ?from)
     (connects ?from ?to $?more)

 =>
     (assert (segment $?path ?from ?to $?more)))

(defrule create-path
      ?partial-path <- (segment $?path exit)
 =>
     (retract ?partial-path)
     (assert (path $?path)))
```

**The Manual Augmentation Process**

Since the path identification process automation is complete, there is no need for manual augmentation to complete this step.

**Example**

For the sample program, the flow graph structure information is used to construct the following edge-connection facts:

```
(deffacts edge-connections (connects begin a) (connects a b) (connects a g) (connects b c)
     (connects b e) (connects c d) (connects d i) (connects d n) (connects i j) (connects i l)
     (connects j k) (connects k p) (connects l m) (connects m p) (connects n o) (connects o p)
     (connects e f) (connects f i) (connects f n) (connects g h) (connects h i) (connects h n))
```

Executing the CLIPS rules generates the following path-edge facts:

```
(path a b c d i j k p)
(path a b c d i l m p)
(path a b c d n o p)
(path a b e f i j k p)
(path a b e f i l m p)
(path a b e f n o p)
(path a g h i j k p)
(path a g h i l m p)
(path a g h n o p)
```

These facts correspond to the set of all possible program paths:

P (abcdijkp), P (abcdilmp), P (abcdnop), P (abefijkp), P (abefilmp), P (abefnop), P (aghijkp), P (aghilmp), and P (aghnop)

## STEP 2. IDENTIFY ALL PATH INPUT DOMAINS

### The Problem

The input domain for any particular path is the set of input values that satisfy all of the predicate (branch) conditions for that path. Each path input domain, I[P ({i})], can be uniquely identified by a two-step process: (1) identify all predicates contained in the path and (2) formulate and evaluate the logical intersection (&) of the associated predicate values in terms of constants and symbolic global input values.

Each program predicate, p (j), can be uniquely identified by the label of its exiting edge and its symbolic value; e.g., the predicate p (b) = (A < B) (for edge b to be reached, the condition A < B must be true).

For each path, an equation can be formulated by using the symbolic values of the predicates that lie along the path. That equation can then be evaluated in symbolic terms to yield the path input domain; e.g., I [P (abcdijkp)] = p (b) & p (c) & p (i) & p (j) = (A < B) & (A < = C) & (A < C) & (A < C) = <(A < B < C) (for path P (abcdijkp) to be executed, the inputs must satisfy the condition A < B < C).

### The Expert's Heuristics

Using the predicate information contained in the flow diagram, the expert first tabulates all of the predicates and identifies each by its exit edge. Next, equations are formulated and evaluated for each path using the symbolic values of the predicates contained in that path.

### The CLIPS Automation

The approach is to generate a set of predicate-value facts corresponding to the set of all program path-edge facts. Each predicate-value fact will contain a string of elements representing the appropriate predicate values for the predicates contained in its corresponding path, e.g., (predicate-string '1st-predicate-value' '2nd-predicate-value' ... 'last predicate value'). This string of predicate-value elements can be evaluated symbolically to yield the corresponding path input domain. For this process, both predicate and path-edge facts are necessary.

The flow diagram predicate information from the program flow chart (Fig. 1) is converted (manually or by another program) into predicate facts (both edge and value) of the form

```
(deffacts predicates
    (predicate 'edge' '"value"')
    (predicate 'edge' '"value"')
    ...
    (predicate 'edge' '"value"'))
```

The rules for the predicate-value element string generation process are more complex than for the path-edge element string generation process. Initially for each path, an empty partial-predicate-value fact and a path-segment-edge fact (whose initial value is the path-edge fact string of edges) are created. The leading edge element is removed from the path-segment fact and tested against all predicate fact edge elements for a match. If a match is not found, the edge is not a predicate, therefore, it is discarded and the next edge is removed. If it is a predicate edge, the corresponding predicate value is appended to the end of the partial-predicate-value fact string. When the path-segment fact is reduced to empty by the leading edge removal process, the partial-predicate-value fact is converted into a predicate-value fact.

The CLIPS rules generate predicate-value facts of the form

(predicates-string '1st-predicate-value' '2nd-predicate-value' ... 'last-predicate-value')

The predicate-value string generation rules are as follows:

```
(defrule initialize
    ?start <- (initial-fact)
=>
    (retract ?start)
    (assert (phase pick)))

(defrule pick-a-path
    ?phase <- (phase pick)
    ?path <- (path $?edges)
=>
    (retract ?path ?phase)
    (assert (phase strip))
    (assert (partial-string))
    (assert (segment $?edges)))

(defrule strip-edge
    ?phase <- (phase strip)
    ?segment <- (segment ?edge $?rest)
=>
    (retract ?segment ?phase)
    (assert (phase process-edge))
    (assert (segment $?rest))
    (assert (edge ?edge)))

(defrule check-edge
    ?phase <- (phase process-edge)
    ?segment <- (edge $?edge)
    (not (predicate ?edge $?predicate))
=>
    (retract ?segment-edge ?phase)
    (assert (phase strip)))

(defrule predicate-edge
    ?phase <- (phase process-edge)
    ?segment-edge <- (edge ?edge)
    (predicate ?edge $?predicate)
    ?partial-string <- (partial-string $?predicate-string)
=>
    (retract ?partial-string ?segment-edge ?phase)
    (assert (phase strip))
    (assert (partial-string $?predicate-string $?predicate)))

(defrule form-strings
    ?phase <- (phase strip)
    ?segment <- (segment $?rest)
    (test (= (length $?rest) 0))
    ?partial-string <- (partial-string $?predicates-string)
=>
    (retract ?segment ?partial-path ?partial-string ?phase)
    (assert (predicate-string $?predicates-string))
    (assert (phase pick)))
```

**The Manual Augmentation Process**

By replacing the " " with ( ) and inserting an & between adjacent predicate value elements in the string, each predicate-value fact can be manually solved to yield the symbolic valued input domain of its corresponding path; e.g.,

(path a $\underline{b}$ $\underline{c}$ d $\underline{i}$ $\underline{j}$ k p)

(predicates-string "A < B" "A => C" "B < C" "A <= C")

corresponds to

$I[P\,(abcdijkp)] = (A < B)\ \&\ (A <= C)\ \&\ (B < C)\ \&\ (A <= C)$

Solving the right-hand side yields

$I[P\,(abcdijkp)] = (A < B < C)$

**Example**

For the sample program, the flow chart predicate information

$p(b) = (A < B)$, $p(c) = (A <= C)$, $p(e) = (C < A)$, $p(g) = (B <= A)$, $p(i) = (B < C)$,
$p(j) = (A <= C)$, $p(l) = (C < A)$, $p(n) = (C <= B)$

is used to construct the following predicate facts:

```
(deffacts predicates (predicate b "A < B")
    (predicate c "A <= C") (predicate e "C < A")
    (predicate g "B <= A") (predicate i "B < C")
    (predicate j "A <= C") (predicate l "C < A")
    (predicate n "C <= B"))
```

These predicate facts are added to the path-edge facts from Step 1 and supplied as inputs.

The predicate-value facts generated by executing the CLIPS rules for this step are combined with the path-edge facts from Step 1 to yield

```
(path a b c d i j k p)
(predicates-string “A < B” “A < = C” “B < C” “A < = C”)
(path a b c d i l m p)
(predicates-string “A < B” “A < = C” “B < C” “C < = A”)
(path a b c d n o p)
(predicates-string “A < B” “A < = C” “C < = B”)
(path a b e f i j k p)
(predicates-string “A < B” “C < A” “B < C” “A < = C”)
(path a b e f i l m p)
(predicates-string “A < B” “C < A” “B < C” “C < A”)
(path a b e f n o p)
(predicates-string “A < B” “C < A” “C < = B”)
(path a g h i j k p)
(predicates-string “B < = A” “B < C” “A < = C”)
(path a g h i l m p)
(predicates-string “B < A” “B < C” “C < A”)
(path a g h n o p)
(predicates-string “B < = A” “C< = B”)
```

Replacing the “ ” with ( ) and inserting an & between predicate elements, each predicate-value string fact is manually solved to identify the following path input domains:

I[P(abcdijkp)] = $(A < B < C)$
I[P(abcdilmp)] = null (empty)
I[P(abcdnop)] = $(A < B = C)$ or $(A < C < B)$ or $(A = C < B)$
I[P(abefijkp)] = null (empty)
I[P(abefilmp)] = null (empty)
I[P(abefnop)] = $(C < A < B)$
I[P(aghijkp)] = $(A = B < C)$ or $(B < A < C)$ or $(B < A = C)$
I[P(aghilmp)] = $(B < C < A)$
I[P(aghnop)] = $(A = B = C)$ or $(C < A = B)$ or $(C < B < A)$

## STEP 3. IDENTIFY ALL EXECUTABLE PATHS

### The Problem

Whether a path can be executed or not depends on the satisfaction of its input domain conditions. If its input domain is null or empty, there are no input values that can satisfy all path predicates and thus path execution is impossible. By using the results of Step 2, the set of all possible paths can be reduced to the set of all executable paths by discarding the paths with null (empty) input domains (Fig. 3).

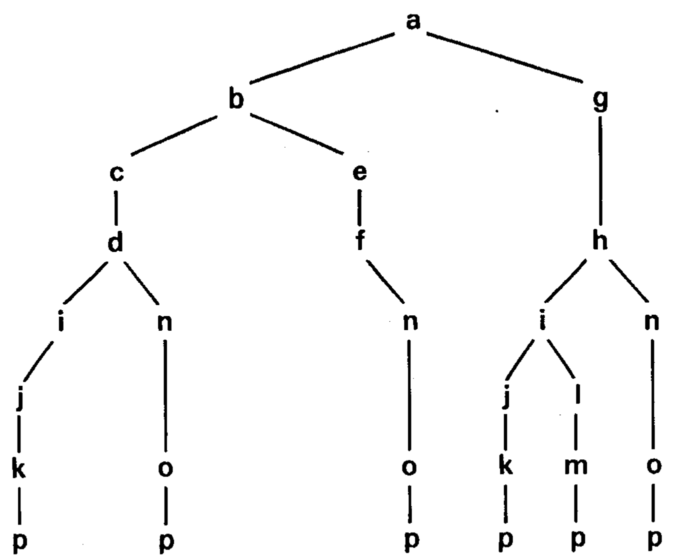


*Fig. 3. Directed Binary Tree of All Executable Paths. Eliminating Branches with empty input domains produces a directed binary tree of all executable paths.*

**The Expert's Heuristics**

The expert's heuristics can be summarized as follows: Delete all paths with null input domains, e.g., since its input domain I[P(abcdilmp)] = null, discard path P(abcdilmp).

**The CLIPS Automation**

This step is satisfied by deleting path facts with corresponding null input domains, but it has not been automated since it is easily accomplished manually.

**The Manual Augmentation Process**

All paths with corresponding null input domains are manually deleted.

**Example**

Removing the path-edge facts

(path a b c d i l m p)
(path a b e f i j k p)
(path a b e f i l m p)

corresponding to paths

P (abcdilmp), P (abefijkp), and P (abefilmp)

yields the reduced path-edge fact set

(path a b c d i j k p)
(path a b c d n o p)

```
(path a b e f n o p)
(path a g h i j k p)
(path a g h i l m p)
(path a g h n o p)
```

corresponding to the paths

P (abcdijkp), P (abcdnop), P (abefnop), P (aghijkp), P (aghilmp), and P (aghnop)

## STEP 4. IDENTIFY ALL PATH OUTPUT DOMAINS

### The Problem

The output domain of any particular path is the result of the sequential evaluation of all computations occurring along that path. Each path output domain O [P({i})] can be uniquely identified by a two-step process: (1) identify all computational elements contained in the path and (2) evaluate these computational elements sequentially in terms of constants and global input values.

Each program computational element, c (k), can be uniquely identified by its exit edge and its symbolic value; e.g, the computation c (d) = (X := 1) (for edge d to be reached, the computation replace X with 1 must be completed).

For each path, an equation can be formulated using those computations values that lie along the path and are evaluated in symbolic terms to yield its output domain; e.g., 0 [P (abcdijkp)] = c (d) & c (k) & c (p) = (X := 1) & (Y := 2) & (Z := X + Y) = (Z := 3) (if path P (abcdijkp) is executed, the output variable Z will have the value 3).

### The Expert's Heuristics

Using the computation information contained in the flow diagram, the expert first tabulates all of the computations and identifies each by its exit edge. Next, equations are formulated and evaluated for each path using the computations contained in that path.

### The CLIPS Automation

The approach is similar to that used for the path input domain generation process. A set of computation-value facts is generated that corresponds to the set of program path-edge facts. Each computation-value fact will contain a string of appropriate computation-values for the corresponding path, e.g., (computation-string '1st-computation-value' '2nd-computation-value' ... 'last-computation-value'). The computation-value elements can be processed for symbolic evaluation to yield the corresponding path output domain.

The flow diagram computation information from the program flow chart (Fig. 1) is converted into computation facts of the form

```
(deffacts computations
    (computation 'edge' '"value"')
    (computation 'edge' '"value"')
    ...
    (computation 'edge' '"value"'))
```

The rules for the computation-value element string generation process are essentially the same as those for the predicate-value string generation process previously described, with the word "computation" substituted for the word "predicate."

**The Manual Augmentation**

For each computation fact, starting with the first computation element, solve each element symbolically and substitute the result into the succeeding elements as necessary until all elements have been evaluated. The final evaluation yields the symbolic valued output domain for the corresponding path; e.g.,

(path a b $\underline{c}$ d i j $\underline{k}$ $\underline{p}$)
(computation-string "X := 1" "Y := 1" "Z := X + Y")

corresponds to

O [P (abcdijkp)] = [(X := 1) & (Y := 1) & (Z := X + Y)]

Solving the right-hand side yields

O [P (abcdijkp)] = (Z := 2)

**Example**

For the sample program, the flow diagram computation information

c (d) = (X := 1), c (f) = (X := 2), c (h) = (X := 3),
c (k) = (Y := 1), c (m) = (Y := 2), c (o) = (Y := 3), and
c (p) = (Z := X + Y)

is used to construct the following computation facts

```
(deffacts computations
    (computation d "X := 1")
    (computation f "X := 2")
    (computation h "X := 3")
    (computation k "Y := 1")
    (computation m "Y := 2")
    (computation o "Y := 3")
    (computation p "Z := X + Y"))
```

These facts are added to the path-edge facts from Step 3 and supplied as inputs.

The computation-value facts generated by the CLIPS rules for this step are combined with the path-edge facts from Step 3 to yield

(path a b c d i j k p)
(computation-string "X := 1" "Y := 1" "Z := X + Y")
(path a b c d n o p)
(computation-string "X := 1" "Y := 3" "Z := X + Y")
(path a b e f n o p)
(computation-string "X := 2" "Y := 3" "Z := X + Y")
(path a g h i j k p)
(computation-string "X := 3" "Y := 1" "Z := X + Y")
(path a g h i l m p)
(computation-string "X := 3" "Y := 2" "Z := X + Y")
(path a g h n o p)
(computation-string "X := 3" "Y := 3" "Z := X + Y")

Solving each element and substituting in subsequent elements yield the following path output domains:

O [P abcdijkp)] = (Z := 2)
O [P abcdnop)] = (Z := 5)
O [P abefnop)] = (Z := 5)
O [P aghijkp)] = (Z := 4)
O [P aghilmp)] = (Z := 5)
O [P aghnop)] = (Z := 6)

## STEP 5. CONSTRUCT A PROGRAM TRUTH TABLE

### The Problem

A program truth table specifies the output response by the program to a given input. To produce a truth table for the program, the input domains and the output domains of each executable path must be correlated. The information necessary to perform this task is contained in the results of the previous steps; e.g., the input ($A < B < C$) results in the execution of path P (abcdijkp) and produces the output ($z = 2$).

### The Expert's Heuristics

Using the results from the previous steps; construct a table of paths and their corresponding input and output values.

### The CLIPS Automation

Automation of this process has not been undertaken because the manual task is simple and straightforward, given the results available from automating the previous steps.

The facts needed are the path, predicate-value, and computation-value facts; augmenting each fact with a path number would facilitate ease of correlation.

## The Manual Augmentation Process

For each executable path resulting from Step 3, select the corresponding input and output domains from Steps 2 and 4, and place the information in a tabular format.

## Example

By using the information generated in Steps 1, 2, and 4, the following truth table is constructed:

| Path<br>P ({i}]) | Input Domain<br>I [P ({i})] | Output Domain<br>O [P ({i})] |
|---|---|---|
| P (abcdijkp) | (A < B < C) | Z = 2 |
| P (abcdnop) | (A < B = C)<br>(A < C < B)<br>(A = C < B) | Z = 4 |
| P (abefnop) | (C < A < B) | Z = 5 |
| P (aghijkp) | (A = B < C)<br>(B < A < C)<br>(B < A = C) | Z = 4 |
| P (aghnop) | (A = B = C)<br>(B = C < A)<br>C < A = B)<br>(C < B < A) | Z = 6 |
| P(aghilmp) | (B < C < A) | Z = 5 |

## SUMMARY

Formulating the structure and content of the facts was the key to the CLIPS automation. Once the fact formats were defined, the path (path-edge) generation rules were developed and tested with relative ease. The path input domain (predicate-value) generation rules seemed to pose a simpler problem; however, their development and testing actually took twice as long as the path generation rules. Since the path output domain (computation-value) generation rules were almost identical to the input domain generation rules, the time required for their development and testing was minimal. As stated earlier, the facts shown were actually augmented to produce printed reports and fact database input/output file management.

Automation of the symbolic evaluation of the predicate-value element strings and computation-value element strings to produce the input and output domains was impeded by CLIPS limitations; e.g., the symbols <, =, >, -, and + are reserved as CLIPS operators. It was for this reason that the the predicate and computation values are placed in quotes in the initial predicate and computation facts. This problem can be alleviated in most cases by substituting non-reserved symbols; e.g., EQ for =, LT for <, PLUS for +, etc. Unfortunately, these substitutions must be made manually or with a preprocessor written in another language.

The automation achieved was limited to the demonstration of a proof of concept for feasibility. In the current version, loops must be treated as separate programs. Much additional work needs to be done in automating loop handling and symbolic evaluation for determining the input and output domains. Some challenging problems have been encountered while evaluating complex inequality equations. Breaking down compound inequalities (e.g, a < = b) to their simplest elements (e.g., a < b or a = b) and using only the < and = relations in that order, as shown in this paper, facilitate the manual evaluation process. Additional work also needs to be done on correlating the facts from the individual steps and producing the final truth table.

The prototype has been used successfully on a program containing eighteen decisions with 251 possible paths. The input facts were easily constructed, the CLIPS path-edge facts were produced quickly, and the predicate-value string facts were manually evaluated in a short time. An experiment has been conducted by using the prototype on a program with 72 decisions and over 80,000 possible paths. Due to the large number of paths, the program had to be subdivided into six segments, each of which was processed separately. Manual evaluation of all predicate-value and computation-value strings to identify the input domains, executable paths, and output domains was not feasible; it was quickly determined, however, that two thirds of the possible paths could not actually be executed. The inability to analyze such a large number of paths should not be taken as a criticism of the prototype, but rather a criticism of the program design.

Future development should involve adding the ability to handle loops directly. A preprocessor should be developed to produce the program edge-connection, predicate; and computation facts directly from program source code. The preprocessor would make predetermined substitutions for CLIPS-reserved symbols in the process, removing the need for enclosing the predicate and computation values in quotes, and facilitating the automation of the evaluation process. A postprocessor should also be provided to reverse the substitution process and produce the truth table in the symbols of the original program, as is now done manually. When source code statement numbers are used in place of flow diagram edges, the symbolic analysis process requires that both the entry and exit statement numbers be supplied for identification purposes. This causes the connection, predicate, and computation facts, and the rules that operate on them to be modified accordingly.

## REFERENCES

1. Keith E. Morris, "A Symbolic Analysis Approach to White Box Verification Testing," *Proceedings* of the 1990 ASME Computers in Engineering Conference and Exhibition (to be published).

2. Joseph C. Giarratano, Ph.D, "CLIPS User's Guide, Version 4.2 of CLIPS," Artificial Intelligence Section, Lyndon B. Johnson Space Center (June 3, 1988).

## NOTES

NOTES

## NOTES

## NOTES

## NOTES

## NOTES

## NOTES

## NOTES

## <u>NOTES</u>

## B2 Session:
## Automated Knowledge Acquisition I

# EDNA: Expert Fault Digraph Analysis Using Clips

Dr. Vishweshwar V. Dixit
Computer Aided Engineering, Engineering Applications
Rockwell International, Mail Stop AB-36
12214 Lakewood Boulevard, Downey, CA 90241
(213)922-0498

## 1 Introduction

### 1.1 Fault Analysis

Traditionally fault models are usually represented by trees. Recently, digraph models have been proposed [Sack]. Digraph models closely imitate the real system dependencies and hence are easy to develop, validate and maintain. However, they can also contain directed cycles and analysis algorithms are hard to find. Available algorithms tend to be complicated and slow.

On the other hand, the tree analysis [VGRH,Tayl] is well understood and rooted in vast research effort and analytical techniques. The tree analysis algorithms are sophisticated and orders of magnitude faster. Transformation of a digraph (cyclic) into trees [CLP,LP] is a viable approach to blend the advantages of both the representations.

Neither the digraphs nor the trees provide the ability to handle heuristic knowledge. An expert system, to capture the engineering knowledge, is essential.

We propose an approach here, namely, expert network analysis. We combine the digraph representation and tree algorithms. The models are augmented by probabilistic and heuristic knowledge. Clips, an expert system shell from NASA-JSC will be used to develop a tool. The technique provides the ability to handle probabilities and heuristic knowledge. Mixed analysis, some nodes with probalilities, is possible. The tool provides graphics interface for input, query, and update. With the combined approach it is expected to be a valuable tool in the design process as well in the capture of final design knowledge.

## 2 Objectives

The tool has the following specific objectives:

- **Digraphs:** Provide the ability to handle digraphs as well as trees.
- **Knowledge Capture:** Incorporate heuristic knowledge gained through experience and reason to guide the design process for fault tolerent systems, and encode the final design knowledge for future analysis.
- **Graphics:** Provide an optional ability to input, query, and update the knowledge graphically.

## 3 Expert System Approach

Network traversal techniques will be used to obtain reachability solution. Intermediate solutions generated by previous queries could be saved to provide an *incremental* build up of solutions and avoid re-computation.

### 3.1 Digraphs

A digraph is constructed from the system schematic in a straight forward manner. For example, consider the thrust vector control function. The schematic and the digraph are shown in Figure 1. The *crew* operates the redundant TVC module to control the *servo* with a feedback loop. Digraph is described by individual component models and their interconnections. Knowledge, analytical and heuristic, is encoded into component models.

As an example of digraph consider figure 2. The digraph contains a cycle $BGF$. When node $A$ is chosen as the *top event* following prime implicants are generated: $BC$, $CE$, $CF$, $D$, $F$, $GH$, $BH$, and $EH$. An acyclic digraph for top event $A$ constructed from these terms is shown Figure 3. Algorithms developed for tree analysis are also capable of handling such graphs.

### 3.2 Expert Model

Each component has a set of input ports and a set of output ports. The behaviour of each output port is described in terms of the component state and the input ports. This need not be an exact state transition description. Probabilities and heuristic knowledge written as *rules* are appropriate. Failure modes are constructed from combinations of the state description, probabilistic, and heuristic knowledge during the procesing. The system model, a digraph with component descriptions as rules, is constructed from the schematic with the help of the persons knowledgeable about the components and their interactions.

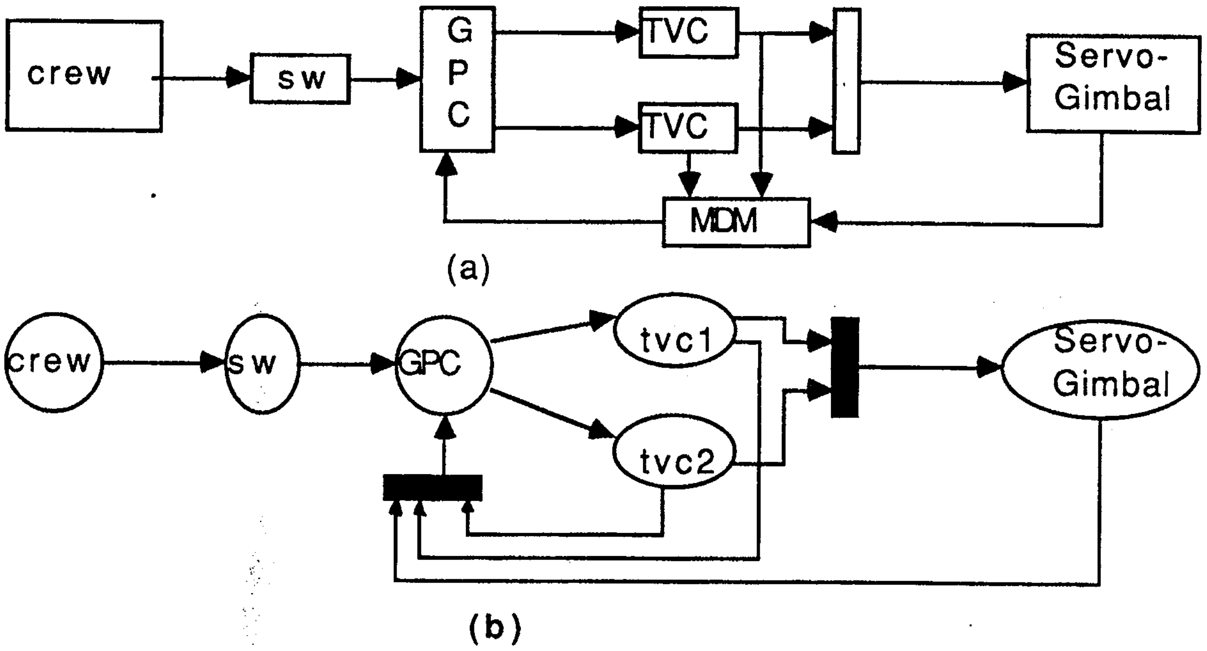


Figure 1: Thrust vector control (a) schematic and (b) digraph

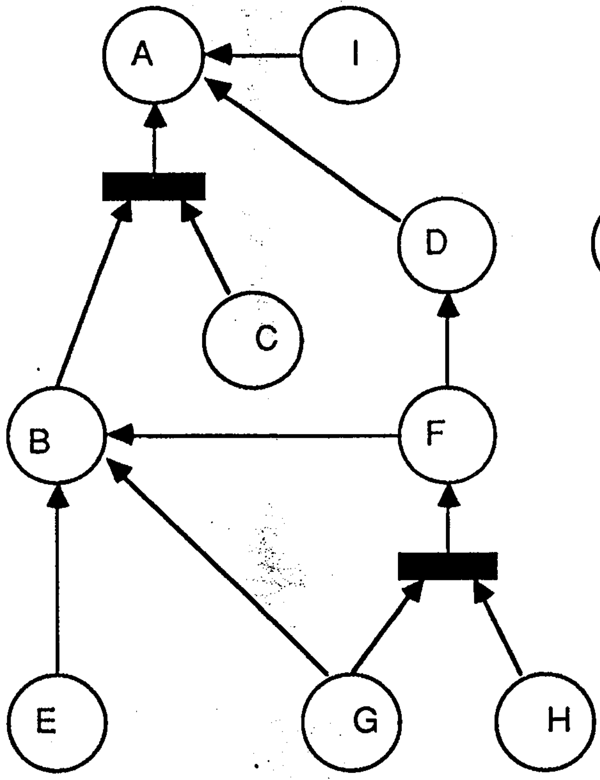


Figure 2: An Example of cyclic digraph

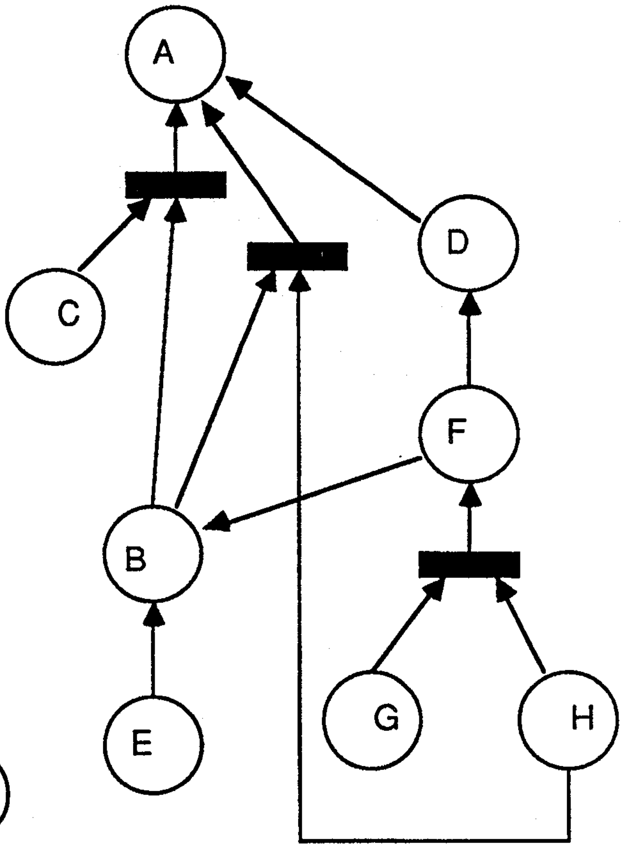


Figure 3: A cycle free digraph for top event $A$

### 3.3 Expert Network Traversal

The system model once developed can be queried for reliability, faults, and, conditions under which certain states can be reached. This is done by traversing the digraph and activating each component along the way. Analogous to petri nets, each activated component fires its rules which in turn enables other components. Both forward and backward reasoning can be accomplished.

### 3.4 Choice of Expert System

Clips is an expert system development tool - a rule based general purpose shell. Clips was developed by NASA Johnson Space Center in Houston. The development of Clips was in response to low availability of Lisp on wide variety of computers, high cost of Lisp tools, and poor integration of Lisp based tools with other languages. Clips is written in C. It is embodies primarily a forward chaining infernce engine based on Rete algorithm. The basic elements of Clips are a fact list, a set of rules, initial conditions, an infernece engine that controls the firing and decides which rule to be fired next, and any user supplied functions, written in C or other language. The C language functions can be directly linked into Clips. Syntax of Clips is similar to Inference Corp.' expert system tool ART. Clips is distributed free and not copyrighted. One may freely use modify, and distribute compiled and source code. Clips runs on a variety of machines including VAX under VMS, Sun workstations, Apple Macintosh, and IBM PC under MSDOS. Clips has been chosen for its ability to integrate with other programs, portability, and extensibility.

## 4 Expert Tool

The system model is developed by the analyst. However, the rules to manipulate the model remain common to all applications. These have to be developed only once. The major steps in the development of the expert tool are:

- integration of Clips with fault tree analysis tools.
- integration with databases.
- rules for backward queries.
- rules for forward queries.
- option of maximum *n-ton* option.
- print options: selective viewing nodes and cutsets.
- handling cycles.

```
(defrule backward-or-rule
  (fail ?target $?pre ?y $?post)
  (F-OR ?x ?y)
=>
  (if (or (member ?x $?pre) (member ?x $?post)) then
      (assert (fail ?target $?pre $?post))
   else
      (assert (fail ?target $?pre ?x $?post)))
)
(defrule backward-and-rule
  (fail ?target $?pre ?b $?post)
  (F-AND ?a ?b ?c)
=>
  (if (or (member ?a $?pre) (member ?a $?post)) then
        (if (or (member ?c $?pre) (member ?c $?post)) then
              (assert (fail ?target $?pre $?post))
         else (assert (fail ?target $?pre ?c $?post))
        )
   else
        (if (or (member ?c $?pre) (member ?c $?post)) then
              (assert (fail ?target $?pre ?a $?post))
         else (assert (fail ?target $?pre ?a ?c $?post)) ))
)
(defrule target-to-fail-or-rule
  (target ?t)
  (F-OR ?x ?t)
=>
  (assert (fail ?t ?x))
)
(defrule target-to-fail-and-rule
  (target ?t)
  (F-AND ?a ?t ?c)
=>
  (assert (fail ?t ?a ?c))
)
(defrule subsume-rule
; given two sets of failure nodes, if one is proper subset
; of the other the proper subset subsumes the bigger set
  ?f1 <- (fail ?t $?x)
  ?f2 <- (fail ?t $?y)
  (test (and (neq ?f1 ?f2) (subset $?x $?y)))
=>
  (retract ?f2)
)
(defrule print-failure
  (fail $?x)
=>
  (printout t "(fail " $?x ")" crlf)
)
```

Figure 4. Simplified Rules for Backward Query

- rules for heuristic knowledge capture and reasoning.
- rules for condensation of digraphs.

## 4.1 Integration

Sophisticated fault tree analysis algorithms and tools are readily available. Clips expert system shell could simply set up the problem and invoke such programs to solve it. Certain knowelede about components and libraries of components, are best kept in a database and retrieved when needed. Interfacing Clips with external procedures becomes essential as the system models grow in size.

As the system model gets large the response time may become unreasonable. The nodes in a cascade are *condensed* together to form a single super node. In practice, chains of nodes occur frequently and condensation offers an effective method to cope with the increasing response time.

## 4.2 System Rules

*Backward* reasoning tries to answers questions like "what (combination of) nodes could fail a given node?". The reasoning might also provide probabilities and remedies. A *singleton* is single node that affect the target node. A doubleton is combination of two nodes that together affect the target node. Probabilities get smaller quickly as the size of the such combinations increase. Hence, one might want to restrict such queries to *n-tons* with a small $n$.

## 4.3 Example

An $OR$ node is represented by a set of facts of the form $(F-OR\ \ a\ \ b)$ meaning node $b$ fails if node $a$ fails. An $AND$ node is represented by the facts of the form $(F-AND\ \ a\ \ b\ \ c)$ meaning $b$ fails if both $a$ and $c$ fail.

Figure 4 lists a simplified set of rules for backward query. Cycles in the digraph pose no problem.

Consider the example in Figure 1. Following facts representing the digraph and the fact to query the failure of the node $servo-gimbal$ are entered.

```
(F-OR crew sw)                     (F-OR sw gpc)
(F-OR gpc tvc1)                    (F-OR gpc tvc2)
(F-OR servo-gimbal mdm)            (F-AND tvc1 mdm tvc2)
(F-AND tvc1 servo-gimbal tvc2)     (F-OR mdm gpc)

                    (fail servo-gimbal)
```

The result of the backward query would be tuples of nodes that together would cause the failure of the *servo* − *gimbal*. The singletons are *crew*, *sw*, *gpc*, *mdm*. The doubleton is (*tvc*1, *tvc*2) as expected.

## 5 Conclusions

EDNA provides an *extensible and portable* AI tool for fault tolerent system design and analysis. The major benefits are

- heuristic design knowledge can be captured
- ability to handle cyclic digraphs
- use of well established tree algorithms for analysis
- previous work in tree analysis and graphics need not be discarded or repeated but incorporated
- ability to embed other applications - user interface, databases

Not only the analyst, but the system designer of fault tolerent system can use this tool.

## References

[CLP] D.L. Cummings, S.A. Lapp, and G.J. Powers. Fault Tree Synthesis From a Directed Graph Model for a Power Distribution Network. *IEEE Trans. on Reliability*, Vol. R-31, No. 2, June 1983.

[LP] S.A. Lapp and G.J. Powers. Computer-aided Synthesis of Fault-trees. *IEEE Trans. on Reliability*, April 1982.

[Sack] I.J. Sacks. Diagraph Matrix Analysis. *IEEE Trans. on Reliability*, Vol. R-34, No. 5, December 1985.

[Tayl] J.R. Taylor. An Algorithm for Fault-Tree Construction. *IEEE Trans. on Reliability*, Vol. R-31, No. 2, June 1982.

[VGRH] W.E. Vesley, F.F. Goldberg, N.H. Roberts, and D.F. Haasi. *Fault Tree Handbook.* Systems and Reliability Research Office of Nuclear Regulatory Research. US Nuclear Regulatory Commission Washington D.C. 20555.

## NOTES

## NOTES

## NOTES

## NOTES

## NOTES

## NOTES

# A MIDDLE MAN APPROACH TO KNOWLEDGE ACQUISITION IN EXPERT SYSTEMS

*aurthors*

Janice A. Jordan
Min-Jin Lin
Richard J. Mayer
Mark E. Sterle

Knowledge Based Systems Laboratory
Department of Industrial Engineering
Texas A&M University
College Station, TX 77843
(409) 845-8500

April 24, 1990

# A MIDDLE MAN APPROACH TO KNOWLEDGE ACQUISITION IN EXPERT SYSTEMS

## ABSTRACT

The Weed Control Advisor (WCA) is a robust expert system that has been successfully implemented on an IBM AT class microcomputer in CLIPS. The goal of the WCA was to demonstrate the feasibility of providing an economical, efficient, user friendly system through which Texas rice producers could obtain expert level knowledge regarding herbicide application for weed control. During the development phase of the WCA, an improved knowledge acquisition method which we call the Middle Man Approach (MMA) was applied to facilitate the communication process between the domain experts and the knowledge engineer. The MMA served to circumvent the problems associated with the more traditional forms of knowledge acquisition by placing the Middle Man, a semi-expert in the problem domain with some computer expertise, at the site of system development. The middle man was able to contribute to system development in two major ways. First, the Middle Man had experience working in rice production and could assume many of the responsibilities normally performed by the domain experts such as (1) explaining the background of the problem domain and (2) determining the important relations. Second, the Middle Man was familiar with computers and worked closely with the system developers to (1) update the rules after the domain experts reviewed the prototype, (2) contribute to the help menus and explanation portions of the expert system, (3) conduct the testing that is required to insure that the expert system gives the expected results (4) answer questions in a timely way, (5) help the knowledge engineer structure the domain knowledge into a useable form, and (6) provide insight into the end user's profile which helped in the development of the simple user friendly interface. The final results were not only that both time expended and costs were greatly reduced by using the MMA, but the quality of the system was improved. This paper will introduce the WCA system and then discuss traditional knowledge acquisition along with some of the problems often associated with it, the MMA methodology, and it's application to the WCA development.

## THE WCA EXPERT SYSTEM

Delivering the WCA on a micro-computer demonstrated the feasibility of producing a robust expert system for rice production. By robust, we mean an expert system that is user friendly, provides reasonable solutions for a wide variety of domain specific problems, explains why some solutions were suggested and others were not, and provides technical information relating to the problem solution.

The WCA is a rule-based expert system written in CLIPS and integrated with user defined C routines. CLIPS is the tool that was used to implement the rules of the expert system because of the speed, efficiency and flexibility it provides on a IBM AT class machine. The user interface was implemented with a simple menu system to obtain the user's input and display the WCA's suggestions and explanations. During program execution, the problem description is entered in the form of facts which interact with rules to produce a candidate set of preliminary solutions. Then, rules which comprise constraint sets are applied to the candidate set to produce evaluations. Finally, the evaluations are processed and a solution (or set of solutions) is displayed for the user along with explanations and general information. The user may try a new scenario by backing up to any of the input screens and revising segments of the original input.

### Need for the WCA

Weed control in rice is essential to productive rice management. Ignoring weed problems can lead to six categories of loss: yield loss, grain quality loss, land value decline, increased harvesting costs, additional herbicide costs, and the cost of extra land preparation and cultivation. The weed control problem is complex, and rice farmers usually look for solutions from two types of information sources. One is in written form, such as those published by the Texas Agricultural Experiment Station or the Texas Agricultural Extension Service, industrial communications (fact sheets, bulletins) and larger research and academic texts. The other is verbal communication in the form of conversation with county and state extension personnel, university faculty and scientists, private consultants, other producers and industry representatives. The written form is easier to access, but it is considerably more difficult for the producer to extract the necessary information from the potentially cumbersome amount of material available. The latter provides more dynamic idea exchange and problem solving but is limited by the availability of informed individuals with which a rice producer can communicate.

The WCA was developed to bring this information to the rice producer in a more convenient form that could be easily accessed. Figure 1 illustrates the WCA Concept. With a user friendly environment the producer can obtain critical information within minutes along with explanations for why a particular herbicide was chosen and what alternative methods of control are possible. The farmer can also execute alternative scenarios based on different criteria to produce different results.

Many important factors are taken into account when making decisions concerning herbicide application. The following are included in the design of the WCA. Weed species, growth stage, and size influenced herbicide effectiveness. Rice growth stage, size, production stage, and water management methods established a time period which determined the preliminary list of herbicides. Herbicide history had an impact on the legal application rates of some herbicides. Other environmental factors that affected weed management decisions were rainfall possibilities, soil moisture, soil fertility, temperature, rice stress, presence of organic matter, days to harvest, rice variety, insecticide application, and the probability of herbicide injury to rice. Weed control decisions are even more complicated due to the variety of herbicides available to choose from. Also, future implications on land preparation resulting from a weed control management policy had to be considered.

## TRADITIONAL KNOWLEDGE ACQUISITION METHOD

A typical expert system development team is comprised of a knowledge engineer and domain experts. The domain experts provide the knowledge and evaluate the expert system's performance. The knowledge engineer determines the methodology with which to implement the domain expert's knowledge.

### Phases in the Traditional Knowledge Acquisition Cycle

There are several steps to accomplishing the process of knowledge acquisition. In the beginning of expert system development, the knowledge engineer needs to acquire a basic understanding of the domain knowledge, usually accomplished by reviewing the literature and becoming familiar with the terminology. Next, a series of interviews with the domain experts are held to identify threshold values and extract the expert's thought processes. Third, the knowledge engineer models the domain experts' reasoning processes used to arrive at a decision, and builds a procedural or rule based prototype as a tool to communicate with the domain experts. Finally, the domain experts review the prototype and make recommendations that are recorded and the whole cycle begins again.

Figure 2 illustrates the knowledge acquisition process. As the diagram shows, implementation of the domain experts' recommendations may require more interviews with the domain experts in order to access the necessary information required to begin the next stage of prototype development. Validation of the prototype by the domain experts is a critical factor in checking the credibility of the expert system. The domain experts test the prototype and evaluate its performance. During the prototype demonstration, the domain experts identify necessary changes or improvements needed. This process can continue until the domain experts are satisfied that the expert system mimics the decisions made for a given situation. When the performance of the expert system satisfies the domain experts, it is delivered to the end user as a beta test version. During field testing, the end user evaluates and identifies necessary changes or improvements and the knowledge engineer continues to maintain and enhance the system.

### Problems With Traditional Knowledge Acquisition

One problem occurs when the domain experts do not have the time to explain all the terminology and reasoning connected with decision making in the problem domain. Frustration can build between the domain experts and the knowledge engineer as a result of these communication problems. Incorrect modeling approaches may emerge when communication with the domain experts is infrequent or difficult to schedule. A second problem is that the knowledge engineer may not fully understand the end user profile and may create a user interface that does not display results in a usable form or an appropriate sequence. The user interface may be too difficult to master quickly or menu explanations may be overly simplified or too complex to meet the user's needs.

### Problems With Knowledge Acquisition in the WCA

Two experts, Dr. A.D. Klosterboer and Dr. E.F. Eastin of the Texas A&M University (TAMU) Agricultural Research and Extension Center at Beaumont, Texas, provided the domain information for the WCA. Aside from their research, they are consultants to rice producers, attend conferences, document research results, and provide other tasks and services. Consequently, their schedules provided little time for the knowledge acquisition task. Since the domain experts did not live or work near the site of expert system development, the distance made communication even more difficult. During the development phase of the WCA the MMA was applied to facilitate the communication process between the domain experts and the knowledge engineer.

## THE MIDDLE MAN APPROACH

The MMA was implemented in the WCA project by C. R. "Rusty" Smith, a masters student in the Department of Agricultural Economics at TAMU. Mr. Smith had several years of experience in rice production and he worked closely with the knowledge engineer. In this case, the Middle Man had substantial knowledge about the problem domain and some computer programming skills, so he could serve as a bridge between the domain experts and the knowledge engineer.

## Tasks and Responsibilities of the Middle Man

Initially the Middle Man plays the part of the domain expert and provides background information to the knowledge engineer on the basic structure of the reasoning used by the domain experts. Occasionally, the Middle Man plays the part of the knowledge engineer to acquire detailed information from the domain experts. Since the Middle Man has sufficient knowledge about the domain, it is much easier for him to understand the logic behind the expert's reasoning.

The Middle Man takes the information available on the problem domain and compiles it into a usable form so that the knowledge engineer can build a prototype. Figure 3 illustrates the MMA as it applies to the knowledge acquisition process. The presence of the Middle Man reduces the number of cycles needed for system development because it bypasses the numerous communications required between the domain experts and the knowledge engineer.

A Middle Man who has had some exposure to computer programming, as well as the problem domain, can help the knowledge engineer build the explanation portions of the help menus. The Middle Man can update the expert system rules after the expert's have evaluated the prototype and can incorporate the expert's suggestions for enhancing the system. The Middle Man can do the extensive testing that is required to verify the system. The Middle Man also provides valuable information about the end user profile to help the knowledge engineer build a useful user interface.

## Middle Man Approach to Knowledge Acquisition in the WCA

The domain experts provided a list of the weed varieties and herbicide compounds, successful strategies, and herbicide effectiveness vs. weed size and species to the knowledge engineer. The Middle Man then created tables with the information and filled in the rest of the input requirements such as weed growth stage, weed size, rice growth stage, rice size and water management methods. The Middle Man also provided explanations for the relationship between the rice growth stage, water management method and the rice production stage. Based on these factors, the Middle Man came up with a list of preliminary herbicides with about 20 possible combinations. The Middle Man determined the environmental constraints used to select the appropriate herbicides and knew the difference between residual herbicides and contact herbicides and how they work. Finally, the initial prototype was demonstrated to the domain experts, changes were recommended by the domain experts to the Middle Man, and the knowledge engineer updated the program to work as the domain experts had expected.

## SUMMARY

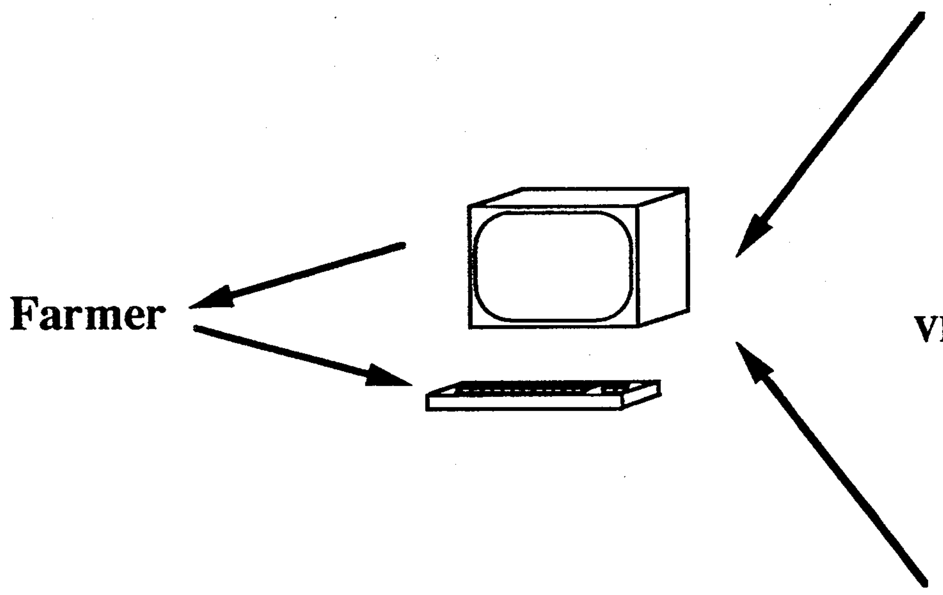
The WCA is a robust farm-level expert system that has been successfully implemented on an IBM AT class microcomputer and has proven to be an efficient and user friendly tool that the rice farmer can consult for his weed problems. The WCA was developed to give Texas rice farmers access to expert knowledge regarding weed control for rice production management through a low cost and convenient vehicle.

During the development phase of the WCA an improved knowledge acquisition method called the MMA was applied to facilitate the communication process between the domain experts and the knowledge engineer. Problems that developed while building early prototypes of the WCA system were due in part to the time constraints experienced by the domain experts. The interview process was difficult to schedule and progress on system development was slow. The MMA served to circumvent the problems associated with the more traditional forms of knowledge acquisition by putting the Middle Man, a semi-expert with some computer expertise, at the site of system development.

With the MMA, the Middle Man performed two main functions that facilitated the development of the expert system. First, the Middle Man was able to take the expert's place in many instances by giving the knowledge engineer the necessary background on the problem domain. The Middle Man knew what was relevant information to the problem domain and where to find that information. The Middle Man was available to answer questions by the knowledge engineer in a timely way when the domain experts had other commitments, and therefor, the project schedule was maintained.

Second, the Middle Man played the part of the knowledge engineer by determining the important relations in the problem domain and helping the knowledge engineer structure the domain knowledge in a useable form. The Middle Man updated the rules after the domain experts reviewed the prototype. The Middle Man contributed to the help menus and explanation portions of the expert system and was able to do the testing that is required to insure that the expert system gives the results expected by the domain experts. Finally, the Middle Man provided insight into the end user's profile which helped in the development of the simple user friendly interface. The final results were not only that both time expended and costs were greatly reduced, but the quality of the system was improved using the MMA.

## Information Sources for Weed Control Solutions



**WRITTEN FORM**

- **Publications**
  - **Texas Agricultural Experiment Station**
  - **Texas Agricultural Extension Service**
- **Industrial Communications**
  - **Fact Sheets & Bulletins**
- **Larger Texts**
  - **Research & Academic**

**VERBAL COMMUNICATION**

- **Extension Personnel**
  - **County & State**
- **University Personnel**
  - **Faculty & Scientists**
- **Private Consultants**
- **Industry Representatives**
- **Other Farmers**

figure 1

TRADITIONAL KNOWLEDGE ACQUISITION

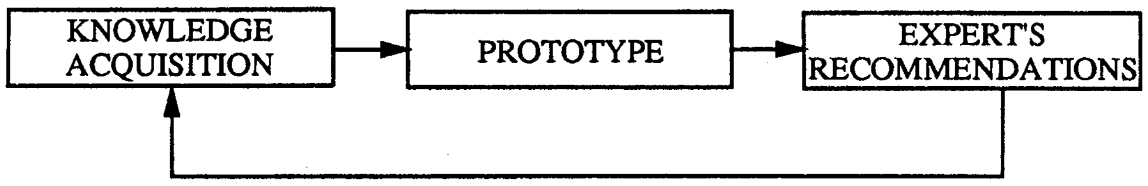


**figure 2**

THE MIDDLE MAN APPROACH

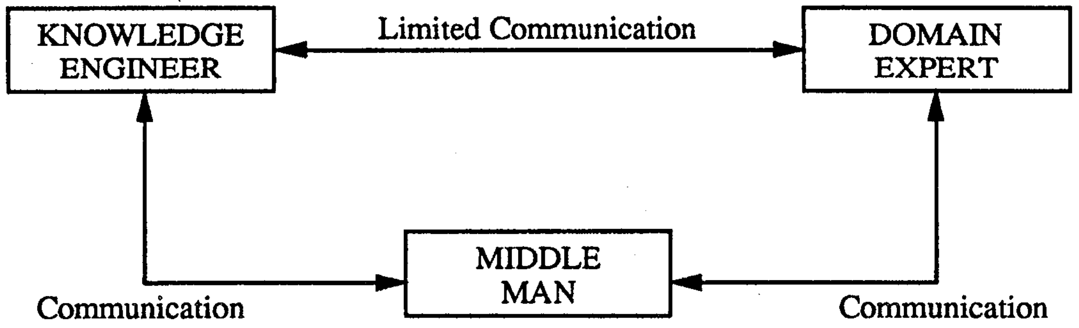


**figure 3**

## A3 Session: Network Applications

# JESNET EXPERT ASSISTANT

ROGER F. HANSEN
LUIS M. FLORES

LOCKHEED ENGINEERING AND SCIENCES CO.
2400 NASA Road 1
P O Box 58561
Houston, TX 77258

## ABSTRACT

The Johnson Space Center Engineering and Science Network (JESNET) supports the JSC Engineering and Space and Life Sciences Directorates by providing a network of interconnected ETHERNETS with a wide diversity of processing and routing nodes throughout the Johnson Space Center and off-site contractor locations. The JESNET provides the primary method of communication between the various computers that support the Directorates. Lockheed Engineering and Sciences Co. (LESC) is tasked with the operation and maintenance of certain primary portions of this network.

Since maintenance of this network depends on the timely identification and subsequent repair of problems on the network, the decision was made to develop a JESNET Expert Assistant (JEA) to assist LESC personnel in the operation and maintenance of the JESNET. The JEA is being implemented in CLIPS, DEC Graphical Kernal System (GKS)™ and C on a DEC MicroVAX™ with VMS™.

# JESNET EXPERT ASSISTANT

ROGER F. HANSEN
LUIS M. FLORES

LOCKHEED ENGINEERING AND SCIENCES CO.
2400 NASA Road 1
P O Box 58561
Houston, TX 77258

## INTRODUCTION

The Johnson Space Center Engineering and Science Network (JESNET) supports the JSC Engineering and Space Life Sciences Directorates by providing a network of interconnected ETHERNETS with a wide variety of processing and routing nodes throughout the center and off-site contractor locations. It also provides access to various gateways and bridges that link JSC to several intercenter and world wide communications networks (SPAN, CSNET, ARPANET), plus providing communication with the JSC administrative network (IBM/PROFS)™.

Lockheed Engineering and Sciences Co. (LESC) is tasked with the operation and maintenance of certain primary portions of this network. As the operator of this network, it is vital that LESC personnel provide timely configuration management information and fault determination, isolation and repair. Since critical skills are in limited supply, new ways to augment these skills must be constantly sought.

## THE PROBLEM

There are two problems inherent in operating and maintaining the type of network under discussion: network configuration management and network fault detection, isolation and repair. Each will be discussed separately.

1. Configuration Management

Any network reconfiguration proposal is driven by network constraints; these are:

a. Naming Constraints - A new address and node name consistent with the existing network must be developed.

b. ETHERNET Constraints - The existing ETHERNET must be examined to determine if a new tap can be placed or if it needs to be extended, what length of new cable and what type of connections are required.

c. Physical Constraints - Where to place the new taps and where to place the new ETHERNET extensions must be determined.

2. Fault Detection, Isolation and Repair

Normally, the detection of a fault depends upon a user of the network to report the anomaly (typically, a node is not reachable). Once the network manager has been notified that a network fault may exist, he must then proceed through a series of diagnostic steps to determine if a problem really exists, e.g., a node may not be reachable because it has been purposely taken off-line for some normal reason. If a network problem is indicated, it must then be isolated and repaired as required.

## THE SOLUTION

It was obvious from the beginning of the project that the solution lent itself to a rule-based system. Since the diagnostic hardware was a DEC VAX and the diagnostic software was implemented on the VAX it was imperative that the expert system shell used to implement the rules could be installed on the VAX.

After a short search, C Language Integrated Production System (CLIPS) was chosen as the implementing expert system shell. CLIPS was chosen for a number of reasons:

- CLIPS could be implemented on any system with a C compiler, i.e. VAX.
- It was available at low cost.
- The authors had previous experience with CLIPS in a PC environment (see the paper "System Control Module Diagnostic Expert Assistant") and therefore knew that CLIPS was easily integrated with external systems and was capable of accessing both internal and external data bases.

The choice of CLIPS subsequently proved to be extremely fortunate for reasons that will be discussed later.

A copy of CLIPS was ported to the VAX, all C source files were compiled and linked to form the executable CLIPS file. With CLIPS now operational on the VAX, development of the JESNET Expert Assistant could proceed.

## METHODOLOGY

The system development philosophy used a modified rapid prototyping methodology where a series of prototypes are developed and delivered to the requestor. Feedback from the experts and users at each prototype stage results in more efficient evolvement of system requirements. This approach also greatly facilitates the knowledge acquisition task and provides a working system for operational evaluation.

The CLIPS environment is ideal for this approach. It enables the developer to demonstrate the latest version of the rules to the user at every stage of development. This encourages user participation in the evaluation and refinement of the rules and leads to a far more useful system.

CLIPS also allows the knowledge engineer to demonstrate his interpretation of the knowledge gleaned from the expert in the form of a working system. This enables the expert to provide a timely evaluation of the knowledge capture process and recommend improvements at every stage of development.

## RESULTS

As was noted previously, the JESNET Expert Assistant has two distinct parts: (1) a Configuration Expert Assistant, and (2) a Diagnostic Expert Assistant. The decision was made to make them accessible from the same program even though they are separate functionally, since they could then share a common data base.

1. Configuration Expert Assistant (CEA)

When reconfiguration assistance is requested, the subsystem accesses the configuration files and the common data area to establish the current network configuration. Once the current configuration is established, the CEA then requests the user to input the desired configuration change. Changes are restricted to adding a node or extending an ETHERNET. After the network change is input, the CEA accesses the network constraints knowledge base to determine the required equipment to be added and check all network constraints. If the requested change can be made, the CEA recommends a plan for accomplishing the reconfiguration.

Initially, the CEA was developed as an interactive, text driven system. Later phases included the development of a graphical capability which produces a drawing of the network on a graphics capable terminal and uses an interactive window for discourse with the user. How this was accomplished will be covered in the following discussion of the Diagnostic Expert Assistant.

## 2. Diagnostic Expert Assistant (DEA)

Phase 1 of the DEA consisted of rules which addressed each element of the network links (routers, bridges, gateways, etc.) as separate entities. CLIPS capability to access external data bases and the CLIPS "system" function proved to be the keys to determining the status of each element under scrutiny.

Phase 2 automatically queried the network to determine the current status of the node under scrutiny, parsed the response, took appropriate actions to determine the cause of the problem, and passed this information to the user along with a recommended remedial action. All of these diagnostic steps were undertaken automatically with no action required from the user. Almost all of the rules were reduced to their generic form, thus removing as much of the individual type-of-node code as possible. Also, the rules were redesigned to depend upon the knowledge base to provide information necessary to diagnose the problem, thus insuring that changes and additions to the JESNET configuration were easily integrated.

Phase 3 introduced a new graphics capability for the JEA. This capability, which required integration of CLIPS code and DEC's Graphical Kernal System (GKS), used CLIPS ability to integrate with external languages and functions. CLIPS main driver was modified using C and GKS to provide the graphics necessary to support the Phase 3 prototype. Phase 3 draws a full screen color representation of the JESNET configuration and then uses the drawing to illustrate either the CEA or DEA reasoning process visually. Any interaction between the JEA and the user is accomplished via a communication window within the graphical representation.

## CONCLUSION

CLIPS capabilities and adaptability made the development of the JESNET Expert Assistant possible. Its ability to communicate with both internal and external data bases enables the use of configuration files and common data bases to simplify the rules and ease maintenance. CLIPS "system" function allowed the rules to use the full capabilities of the operating system and the network diagnostic software implemented on the VAX. Also, CLIPS ability to allow the inclusion of external routines in its code facilitated the development of graphics displays for the JEA. Finally, the CLIPS function "rules-to-C" made possible the development of an easily executed JEA module.

Using CLIPS and proven knowledge engineering techniques has provided LESC with a superior tool with which to perform its tasks of operation and maintenance of the Johnson Space Center Engineering and Science Network.

# The Network Management Expert System Prototype For Sun Workstations

Albert Leigh
McDonnell Douglas Space Systems Company
Houston, Texas

Networking has become one of the fastest growing areas in the computer industry. The emergence of distributed workstations make networking more popular because they need to have connectivity between themselves as well as with other computer systems to share information and system resources. Making the networks more efficient and expandable by selecting network services and devices that fit to one's need is vital to achieve reliability and fast throughput. Networks are dynamically changing and growing at a rate that outpaces the available human resources. Therefore, there is a need to multiply the expertise rapidly rather than employing more network managers. In addition, setting up and maintaining networks by following the manuals can be tedious and cumbersome even for an experienced network manager. This prototype expert system was developed to experiment on Sun Workstations to assist system and network managers in selecting and configurating network services.

# *The Network Management Expert System Prototype for* Sun Workstations

Albert B. Leigh
McDonnell Douglas Space Systems Company

## 1. *Introduction*

Networking has become one of the fastest growing areas in the computer industry. The emergence of personal computers (PC) and workstations make networking more popular because they need to have connectivity within themselves as well as with other computer systems to exchange information. Constructing and maintaining these networks becomes a major chore for system and network managers who are responsible for administering computer systems. Most of the networks are dynamically changing and growing. Therefore, making the networks more efficient and expandable by selecting the network services and devices that fit to one's need is vital to achieve reliability and fast throughput.

Networking plays a major role in UNIX operating systems because UNIX is available from PC's with the Intel 80286 processors and distributed workstations to supercomputers. Configuration of these machines depend on individual site requirements. Some installations have high capacity diskdrives on each node and some have centralized fileservers to serve nodes with smaller disks or diskless nodes. In the distributed environment, sharable software packages and hardware devices such as desktop publishing software, high-quality laser printers, compilers, and on-line manuals do not need to be installed redundantly by sharing the mass-storage media. The user home directories can be configured on fileservers so that personal files can be accessed from the machines on the network. There are variety of network services available on UNIX systems for Local Area Networks (LAN) and Wide Area Networks (WAN). A LAN is a network that transmits a large amount of data at high speeds over limited distances. A WAN network can basically span all over the world via phone lines and transmits data at lower speeds. LANs often need to communicate with other LANs at another location via a WAN.

This prototype project, Network Management Expert System (NETMES), emphasizes LAN and WAN services available on the Sun Workstation. Sun Workstations offer the Berkeley UNIX operating system, open systems networking architecture, powerful RISC (SPARC) architecture, and attractive cost. This expert system provides assistance to system administrators, who are the intended users, in selecting and configuring network services on Sun Workstations. The users should have some background in UNIX system administrations and networking concepts to fully understand the advices and instructions that the expert system delivers. It also features explanation facility that basically allows users to query why the system gives a certain advice or an action.

The expert system provides a menu driven user interface to guide the user through the system prompts required to select and configure network services. When advice is given, the expert system prompts the user with a submenu to continue to next step, ask for explanations, go back to the main menu, or exit to the CLIPS prompt. All rules are coded in CLIPS. The explanation facilitates to answer 'why' type of questions to explain why a particular rules is fired. The detail description of the program is described in section 4.

## 2. *Statement of the problem*

### 2.1 *Description of the problem domain*

Setting up and managing the network on UNIX operating systems by following the manuals and without a friendly graphical user-interface (GUI) can be very tedious. Many companies spend a great deal of time and money sending their personnel to computer training schools or buying installation services from the computer vendors. There are only a handful of system administration and network management software packages available on the market today. These software packages cost at least several thousand dollars. The software vendors claim that they do not have much competition in this market.

The NETMES expert system will assist system managers in selecting and configuring network services. However, the user should have some background in UNIX system administrations and networking concepts because the expert system has no magic and it will not replace the human expert. The expert system emphasizes on Sun Workstations to restrict and simplify the problem domain. There are a variety of UNIX systems such as AT&T System V, Berkeley UNIX, HP-UX, and DEC's Ultrix. Each version may have different ways of managing the system and different sets of networking services. Even when they employ the same network services, some systems have different system configuration files.

Out of all the network services available on the Sun Workstations, only the services mostly used are selected as the rules for the NETMES system. The expert system is implemented in CLIPS and it applies forward-chaining strategy provided by the CLIPS inference engine. This expert system can be considered quite shallow because it does not provide much reasonings or explanations. These deficiencies will be improved as prospects for future work as well as expanding the expert system. The features of the NETMES include network services selection, local area network services such as Network Filesystem, Yellow Page database management, and configuring a wide area network.

The network services selection feature gives advice to the administrators as to why they need to have certain network services and how these services will ease the network management job. The Network Filesystem (NFS) enables machines to share file systems, also known as disk partitions, across a network utilizing Remote Procedure Call (RPC) services, DOD's Transmission Control Protocol/Interface Protocol (TCP/IP), and ethernet. These low level services are completely transparent to users and they see network file systems as if they are local disk drives. NFS actually enables heterogeneous operating systems such as variations of UNIX, DOS and Digital Equipment Corporation's VMS to share the mass storage. This project, however, focuses on Sun OS, which is a derivative of the Berkeley UNIX operating system, as mentioned before.

The Yellow Pages (YP) is Sun Microsystems' distributed network lookup service. It maintains a set of files such as password information, group information and host table files that machines can query as they would in a database. These files are known as YP maps. All YP maps are fully replicated on several systems known as YP servers, each of which runs a server process for the maps. There are two kinds of YP servers: a YP master server and a YP slave server. The master server updates the maps of the slave servers. The YP clients run processes that request data from maps on these machines. YP clients do not care which server is the master, since all YP servers should have the same information. The distinction between master and slave server only applies to where the YP file updates are made. The YP files should always be modified on the master server and modifications are propagated to the slave servers to prevent chaos if there are duplicate sets of system files such as password file on the network. It also ensures consistency of the user and system information.

The aforementioned services are strictly for a local area network. The last feature is how to configure a wide area network. All UNIX systems provide a service called UUCP, UNIX-to-UNIX-Copy. UUCP allows systems to communicate with each other using either dial-up or hardwired communication lines. Once a UUCP line is available, systems can also run commands on remote machines. The physical connection can be used by the Kermit programs to communicate with UNIX and other systems including PC's. Of course, this wide area network does not provide a real-time data transfer unlike the high-speed local area network.

## 2.2 *Appropriate domains for NETMES*

A few questions are raised before building this expert system to see whether the problem domain is appropriate for an expert system. They are as follows:

- Can the problem can be solved effectively by conventional programming?

The problem can be solved by conventional programming but perhaps not very effectively. The problem is very complex and only partial tools can be built easily using a procedural language. However, when the rules change, as they always do because the network grows or a new operating release require changes, the code has to be modified extensively to adapt to those changes. This is probably not feasible or desirable. If it were developed in an expert system language, only the rules that are changed need be modified and leave the logic alone to the inference engine. Consequently, the expert system provides more flexibility.

- Is the domain well-bounded?

The problem domain is well-bounded because the availability of network services or the ways of managing the network is well defined and finite. A dilemma is that there are variations of UNIX are each system has its own "standards" but is avoided by emphasizing on Sun Workstations and the problem domain is well defined within the scope of the Sun OS.

- Is there a need and desire for an expert system?

This question must be answered with a definite 'yes' because network experts are very demanding on the job market today and there are not too many expert systems commercially available for configuring network services. Expertise are available from only a handful of people who are specially trained at computer vendor's training schools or who have a prior experience and a thorough knowledge on the subject. The information is also available from system manuals but they are hard to understand and need to be filtered out.

- Can the expert explain the knowledge so that it is understandable by the knowledge engineer?

The knowledge acquisition is not very difficult because the knowledge engineer himself has some experience setting up the networks and is extremely familiar with UNIX system administrations. It is certainly understandable.

- Is the problem-solving knowledge mainly heuristic and uncertain?

A portion of the problem-solving is heuristic but the majority of the rules are certain and well-defined because there are a set of system configuration files that need be altered and a set of system commands that need be issued. The heuristic part appears in the network selection and tutorial sections.

## 3. *Overview of the approach to NETMES*

The problem is identified as a need in the networking community because there is no systematic way of setting up and managing networks at the local computing facilities. The problem was first analyzed to meet certain criteria such as feasibility and rapid prototyping. Before implementing the prototype, the problem domain analyzed is to focus on UNIX system administration in general. Because of the time constraint, it is realistic to concentrate only on a portion of the chosen domain. As a result, the problem domain is reduced to networking among UNIX systems but again it remains rather complex because there is no standard on UNIX systems. After careful consideration, Sun Workstations are selected because they rely on networking concepts and some of them are diskless workstations. These diskless machines depend on host machines which have mass-storage peripherals.

Once the problem was selected, a brief research was done by reading system manuals to see which useful rules can be extracted out of the lengthy volumes. Then, a set of network services that are heavily used are selected as rule-based information. The first version of the prototype was written in about eight hours in order to show how the expert system can fire certain rules and was demonstrated to my colleagues. It features a menu and a set of queries about selecting network services. The remaining rules are then collected by interviewing the expert, recalling from my experience, and reviewing the system manuals. Most of the rules about YP services are obtained from the network expert. The rules are translated to if-then form and the rest of the rules are implemented in CLIPS.

The review of related papers also provides some hints and clues about building the expert system in general such as knowledge acquisition, selecting problem domain, and planning the network. One of the articles is written by people who have helped build many expert systems for school projects and commercial ventures. Their article was about lessons learned from building expert systems. After completion of the NETMES system, it is tested by three system administrators and they are impressed with the advices and suggest that the system should be expanded to serve the entire system administration task.

## 4. *Detailed description of the program*

The NETMES expert system is designed such that it facilitates a user-friendly interface with menus, queries, and explanations. There are three menus: main menu, YP menu, and NFS menu. The main menu allows the user to select the available services such as 'advice' (Network Services Selection Advisor), 'YP' (Yellow Page Database Lookup Services), 'NFS' (Network Filesystem), and 'exit' (Exit to CLIPS). These menu selections must be entered as they appear on the screen and they are case-sensitive.

The first menu selection 'advice' leads to a set of queries to see what kind of network capabilities the user wishes to facilitate and then the system advices the appropriate network services that should be implemented. Each time a rule gets fired, a NETMES submenu is prompted as the following example.

```
+++ NETMES subshell menu +++
Enter '(w)hy', '(m)ain', '(q)uit' or '(c)ontinue' ->
```

The first choice 'why' is the explanation facility. It can be selected with either the first character or the entire word. This explains why the user receives the fired-rule, a suggestion in this case. Reasons are then asserted by the same rule in the following format:

```
(assert (reason [explanation is inserted here]))
```

The explanation facility picks up this fact by matching the pattern 'reason' and allows the user to view it. More than one reason can be asserted and the explanation facility loops through them until reasons depleted.

The second selection 'main' takes the user back to the main menu without going through the rest of the rules that are waiting to be fired. The third selection 'quit' allows the user to completely exits out to the CLIPS shell and clear the agenda. The last choice 'continue' simply allows the user to proceed without receiving any explanation. This sub-shell menu is always prompted upon firing a rule.

The YP menu permits the user to receive a brief tutorial about the YP services and definitions. The three menu selections, appeared below the tutorial on the screen, allow the user to set up a master server, a slave server and a YP client. One of these menu selections lead the user to a set of queries, similar to the 'advice' from the main menu, which are necessary input data to set up YP services. The last menu selection allows the user to return to the main menu. The user is instructed to enter the first word of a menu entry to make the selection. When a rule is fired, the user receives the sub-shell menu mentioned above to take a desired action.

The NFS menu is very much similar to the YP menu except, of course, it permits the user to select one of the NFS services. It also features a tutorial facility so that the user can familiarize himself with NFS terminology. Sometimes, the expert system informs the user that a system command will be initiated but actually it does not spawn any command because the system is required to run on DOS for grading purpose. Eventually, the system will be corrected to perform real commands on a Sun Workstation. Some user answers can result in failure and the system would display that an error condition occurs. This error facility is tied into the explanation so that the system can explain why the error is flagged.

Finally, there is a Wide Area Network (WAN) facility which allows the user to set up a generic UNIX communication program called UUCP and Kermit lines. These facilities enable the systems to communicate with remote systems as far as such locations overseas. Upon selecting the 'wan' from the main menu, the user is simply prompted with two questions whether he wishes to set up UUCP and/or Kermit. Then, appropriate questions are asked for selected services to proceed. Again, the explanation facility comes in when each rule gets fired.

This expert system is designed to be brought up easily. All the user needs to do is to bring up CLIPS, load the program, reset for the initial-fact to be on the agenda, and then execute the program. Lower-case letters are recommended to select menu picks and answer queries although 'y' and 'n' type of responses are not case-sensitive.

## 5. *Conclusion*

The prototype system gives advice but it does not issue system commands to physically edit the system files or start up daemon processes as it says. This expert system should help the system administrators and network managers configure the network services. The lessons learned from working on this project are lengthy and extremely valuable. The system can be used in real world only to a certain extent because of the system is simplified due to the time constraint. It can definitely be expanded to facilitate network diagnosing capability and to have a better explanation facility. The character oriented user-interface should be replaced with a generic graphical user-interface such as the X-Window System to make the system more meaningful and friendlier. Eventually, the expert system should be generic so that it can be used for all types of UNIX operating systems. This will take some time because the proposed POSIX, a standardization of UNIX by ANSI, is yet to be accepted and implemented by computer manufacturers.

## 6. *References*

1. Dr. J. Giarratano and G. Riley, "Expert Systems - Principles and Programming", 1989

2. Sun Microsystems, "System & Network Administration", Revision A, 1988

3. Artificial Intelligence Section/NASA/JSC, "CLIPS User's Guide", 1989

4. Artificial Intelligence Section/NASA/JSC, "CLIPS Reference Manual", 1989

5. Lindsay Hiebert, "AI and Network Planning", AI Expert, September 1988

6. A. Terry Bahill and Pat Harris, "Lessons Learned, Building Expert Systems", AI Expert, September 1988

# Using CLIPS in a Distributed System
# The Network Control Center (NCC) Expert System

Tom Wannemacher
AFLC/PMXS (CDMS)
Wright-Patterson Air Force Base, Ohio

## Abstract

This paper describes an intelligent troubleshooting system for the Help Desk domain. It was developed on an IBM-compatible 80286 PC using Microsoft C and CLIPS and an AT&T 3B2 minicomputer using the UNIFY database and a combination of shell script, C programs and SQL queries. The two computers are linked by a lan. The functions of this system are to help non-technical NCC personnel handle trouble calls, to keep a log of problem calls with complete, concise information, and to keep a historical database of problems. The database helps identify hardware and software problem areas and provides a source of new rules for the troubleshooting knowledge base.

## Introduction

The Network Control Center (NCC) will be a central contact point for all problems encountered by our users. It will not be necessary for everybody to keep a list of office symbols, telephone numbers, and systems in case of trouble. They will call one central office which will take care of logging problems, determining who is responsible for them, and calling them in.

Our system is composed of many different pieces of hardware and software. Any system with this many moving parts will be difficult to maintain. Difficulties will include the following:

In our environment, one major problem for the users will be in knowing who to call for which problem.

Another will be identifying exactly where and what the problem is. The tendency will be for users to call whoever they know who might be able to solve the problem. The result will be constant interruptions for people involved in the system.

A further problem, related to the above two, is caused by the fact that our system will not be entirely maintained in-house. Many of the critical functions will be out of our control in other organizations. If we are not able to identify problems succinctly and correctly, finger pointing will replace troubleshooting.

## I. Purpose of NCC Expert System.

The NCC personnel can perhaps get along with a set of checklists and phone numbers and a log book in which they can write descriptions of problems when users call in. If we take that approach though, much of the information will be unusable and a lot will fall between the cracks. If as much as possible is automated, our NCC personnel will be able to give quicker and more effective response to users' problems. An additional benefit of the Expert System approach is that, with proper maintenance, the system will improve with use.

## II. System Configuration.

This system is distributed between PCs and an AT&T 3B2 minicomputer. Either component can perform its functions without the other.

### A. PC Component.

The critical parts of the NCC system, the software to log problems and the expert troubleshooter, are hosted on one or more PCs. This way, the NCC can respond as normal to user calls under all circumstances, no matter which components of the production system are down.

### B. Minicomputer Component.

The historical log, tracking information and the associated query and reporting software are hosted on an AT&T 3B2. A menu driven system handles normal queries and reports.

## III. Execution.

To execute the floppy disk based version of the NCC system, insert the floppy in drive A: and proceed as follows:

### A. Load the system:

```
A:
NCC
```

While the system is loading and initializing, a variety of messages are displayed on the screen. The names of the rules of the latest knowledge bases will be displayed on the screen as they are compiled. Finally, the system menu will be displayed.

Use only lowercase when entering commands. CLIPS will not understand if you enter uppercase. If somewhere along the line you inadvertantly enter an uppercase or somehow throw yourself out of the menu system, simply enter '(restart)', with the parentheses but without the apostrophes, to get going again.

## B. NCC commands:

'**up**' causes a rule to kick off that will prompt you for the name of the system component that has come back up. For major components like 'ibm' or 'afcac' you can enter 'reset-ibm' or 'reset-afcac' to bring up that component plus all software running under it. Otherwise just enter the name of the component that just came up, like 'roscoe'. Be sure to use lowercase.

'**down**' causes a logging module to be called. A number of things happen here:

The system asks the operator to log on. The profile should already have been set up. This file contains information like the name of the NCC person, his initials (used as part of the problem log's key), his office symbol and telephone number. Just enter the name of the file, like '1DBM'.

It is not necessary to exit and restart the system when someone new mans one of the NCC PCs. Each time a problem is reported, the system asks for the operator to log on. Whoever is taking the problem call should use their own logon id.

The system asks for the name of the person calling in a problem. Do not use blanks, use underlines or dashes instead.

Next, the office symbol of the person is prompted for.

Then his telephone number is requested.

The system asks for the name of the component that has failed. Enter the number next to that component's name. If you don't know what has failed, select 'unknown'.

The system will prompt you for a text description of what happened, symptoms, etc. Just enter a simple description, hitting carriage return every line or two.

The system will automatically generate a problem log, storing it in file **NCCLOG.CUR**. Then a message will be displayed telling you who to call for this problem.

If you entered 'unknown', you will now enter a **GURU** session. **GURU** will display a list of symptoms. Ask the user on the phone if any of those things sound like the problem he has. If so, enter the number of the rule from the menu. If you select one, you will be told where the problem is, what it is, how to report it and who to report it to.

Finally, you will be returned to the main NCC menu.

'**guru**' brings up the system's troubleshooting rules. Guru displays a list of symptoms, grouped by hardware platform. Next to the symptom is the number of the rule dealing with that problem. Just select the number next to the symptom the caller identifies. If there is nothing like what the caller says is on his screen, enter zero. Anytime a new problem is identified, a new rule should be created to handle it and a new entry made on the main guru menu.

'**status**' causes the current state of the system to be displayed on the screen.

'**exit**' quits the NCC system. If you are at the CLIPS> prompt, you can also enter '(exit)'.

## IV. GURU Troubleshooting.

As the system develops, GURU troubleshooting rules and screens will be layered, from general to specific. The current design consists in two layers, a general rule and many specific rules.

A. The general GURU rule/screen displays messages or error messages or other clear problem symptoms. The NCC operator asks the caller what symptom is displayed on his screen. Many problems can be identified this way. If the symptom is found in GURU's list, the NCC operator selects the number displayed next to it. GURU then kicks off a rule and the specific screen dealing with that problem is displayed.

B. GURU rule/screen displays:

When a problem has been identified, GURU displays its information. A screen header identifies the rule number. Below that is a short paragraph describing the problem. Following is the steps to follow to solve the problem, if it can be solved over the phone. Otherwise, the operator is told who the contact is for that problem, what to tell them, their office symbol and telephone number. The operator should just read the GURU display to the expert over the phone, it will be in his language and will make sense to him.

## V. NCC Rule Maintenance

### A. Adding Rules to the NCC System.

Once the NCC system is set up for a hardware/software configuration, the 'C' code should never have to be changed. New knowledge will occasionally be added however. New knowledge is added in the form of CLIPS rules. These rules are ASCII text and can be added with EDLIN or any PC editor. Before attempting to add new knowledge to the NCC system, make sure the knowledge is appropriate and complete enough.

### B. Situations where rules are appropriate.

An error message or symptom that will be obvious to the caller exists that will uniquely identify the problem.

A solution exists such that:

1. The problem can be solved on the spot.

A solution exists that can be read over the phone and that will be clear to the caller.

2. The problem must be referred to the domain experts.

A description of the problem that will identify it clearly has been obtained from the office in charge of fixing this problem.

## C. Procedure to add rules.

If the situation is appropriate for creating CLIPS rules, proceed as follows:

Using PCWrite or EDLIN or some similar editor, edit file **NCC\NCC.CLP**. Add a line to the GURU menu in rule 'system guru' that displays the following information:

Number of the new rule, this must be unique.

An error message or short description of the symptoms. This short phrase must be clearly visible on the caller's screen. Do not put in anything that will have to be explained at length over the phone.

Using your editor, edit file **A:\NCC\NCCRULES.KB**. Find the rules dealing with the current hardware platform, copy one of them. In the copy, change all occurrances of the rule number to your new rule number. Then change the information that will be displayed, this is inside the double quotes. Add or delete 'printout' lines but change nothing else.

## D. Format of GURU rules.

Model your new rule after the old ones. Be sure to include all relevant information in the same format and as succinct as possible. The necessary information includes the following:

1. Rule Header.
This is simply 'RULE x'.

2. Full description of the problem.

3. Steps to follow to solve.
This is either a series of steps to read to the caller over the phone, or a message to read to the office with expertise in this area. Whichever, it is in the language and terminology of the experts, not the NCC.

4. Office symbol and telephone number of the office with relevant expertise.

## E. The GURU Troubleshooting menus:

There is one menu screen for each platform. Below are screens for two of them.

```
*******************************************************
*           System Problem Troubleshooting GURU        *
*                                                      *
*    IBM:                                              *
*     1 - 'CICS>' prompt displayed                     *
*     2 - 'LOGON APPLID' prompt displayed'             *
*                                                      *
*   Enter the problem number, '0' if none match:       *
*******************************************************
```

If one of the selections on the current GURU menu is entered, that rule is kicked off. Otherwise the next menu is displayed.

```
*******************************************************
*            System Problem Troubleshooting GURU       *
*                                                      *
*   LAN:                                               *
*     200 - 'Connection (n)'                           *
*     201 - No LAN prompt (>>) displayed'              *
*     202 - 'HANGUP'                                   *
*     203 - 'HOST ID not found'                        *
*     204 - LAN timeout at logon                       *
*                                                      *
*  Enter the problem number, '0' if none match         *
*******************************************************
```

## F. GURU Problem solution rules:

If one of the known problems on the above menus matches the current problem, the rule associated with the selected symptom will fire. Sometimes GURU is able to isolate the problem and give specific instructions based on the error message. Rule 203 is one of these:

```
*******************************************************
*                 RULE 203                             *
*                                                      *
*   If the message 'HOST ID not found' is displayed,   *
* there is a problem with the LAN's hostid table.      *
* Call the LAN office and tell them the message and    *
* the host's name.                                     *
*                                                      *
*                     LAN Office tel:  75531           *
*                                                      *
*     Depress <c> to continue                          *
*******************************************************
```

G. **GURU troubleshooting rules:**

In other cases the situation is not so clearcut. Rule 200 gives several options:

```
*********************************************************
*                                                       *
*                   RULE 200                            *
*                                                       *
*    When the message 'Connection (n)' is displayed     *
* during logon, there is something wrong between the    *
* NIU and the mainframe.  The host may be down or the   *
* host's port may be dead.  It is also possible that    *
* the cable between the NIU and the host is bad.        *
*                                                       *
*    Ask the user which host he was trying to connect   *
* to.  Call that office and see if the computer is up.  *
*                                                       *
*    If the host is up, call the LAN office and report  *
* that there is a problem between the NIU and the host. *
*                                                       *
*                        LAN Office tel:  75531         *
*                                                       *
*     Depress <c> to continue                           *
*********************************************************
```

H. **Further Problem Isolation using GURU rules:**

In the 'down' procedure, if the NCC operator cannot isolate the problem, he can log it as an unknown on one of the systems, as in 'unknown-ibm'. If the problem is completely unidentifiable, it should be logged as a generic 'unknown'. At present, GURU's mechanism for handling unknown problems is present only as a stub. In the problem database, 'unknown' is a reminder that further analysis is necessary. Enough information is logged to make that analysis possible. As the 'unknowns' are analysed, it should be possible to add rules to cover some of them and to write new rules to further clarify different situations. Development of a more generic problem solving approach will have to wait until our production system finally comes up.

## VI. Security and System Configuration

### A. Security.

The NCC PC(s) should be manned constantly during working hours and should all be in the same room. If PC security is a problem, the daily log (NCCLOG.CUR) should be uploaded to the 3B2's UNIFY database and deleted from the PC hard disk before shutting down each day.

Access to the 3B2 is thru the PCLogin system using Smarterm 220. PCLogin is a system written in 'C' that asks for the password at PC bootup time. The password is inserted into ST220 scripts that are generated and stored on the ramdisk. For the rest of the day, the user can log on to one or more mainframe or mini automatically, without entering phone numbers, lan commands, userids, passwords etc. As long as the NCC personnel logoff the PCLogin system before leaving the PC unattended, and keep the 3B2's NCC password secret, security between the PCs and 3B2's should not be a problem.

## B. Authorizing new NCC Personnel.

Every member of the NCC who may ever answer problem calls from users should have his own NCC userid on every NCC PC used for this purpose. Create a new userid as follows:

**Create a new file with your PC editor.** The filename should be the operator's last name, in the format of a DOS filename. A userid from another system also works.

The information in the file contains four lines, each line is a alphanumeric string with no embedded spaces. The first line is Name, the second is three position initials, the third is office symbol and the fourth is telephone number. For example,

```
Tom_Wannemacher
TLW
AFLC/PMXS(CDMS/SITA)
257-5941
```

## C. Configuring new NCC PCs.

The procedure to add a new PC to the NCC is fairly simple. Just set up an NCC directory and copy the files from the floppy disk. Then set up an NCC.BAT file to execute it. Be sure to put the NCC.BAT in a directory on a PATH set up by the AUTOEXEC.BAT file.

Edit file **NCCLOG.PC**. Make the ID of the new NCC PC different from the other NCC PCs but follow the same conventions. For example, if you already have NCC PCs 'A', 'B', and 'C', the new one will be 'D'. If numeric, make it the next digit. The ID must be one position, do not make it '10'.

Edit file **NCCLOG.NR**. Start it at 0 (zero).

# VII. Problem Tracking and Trend Analysis

## A. Background.

The production version of the NCC problem database is stored on a UNIFY database on the minicomputer. All historical analysis is done on that platform. If that system is down and version is implemented, the same work can be done manually using the PC based LOG file, NCCLOG.CUR.

## B. Using the NCC LOG File.

Problems are automatically logged onto a file called **NCCLOG.CUR**. Sometime each day, this file will be uploaded to the minicomputer and the information added to a UNIFY database. For occasions when the minicomputer is not available or when something needs to be checked out on the PC, the problem log history will be the PC based file. To examine it, take the NCC floppy to a PC that has a printer, and enter,

```
PRINT A:\NCC\NCCLOG.CUR
```

C. Information contained in NCCLOG.CUR.

Enough information is contained in the LOG file for tracking and trend analysis to be possible. After uploading to the UNIX machine, programs do it automatically. The slots in the file contain the following information about each problem:

1. Caller Name.
This is the name of the person who called in the problem.

2. Caller Office.

3. Caller Telephone.
In case further follow up by the NCC or another office is necessary, the telephone number of the original caller will be available.

4. Hardware.
The platform on which the problem occurred. This is things like IBM, LAN, DDN, and other equipment.

5. Software.
The software that aborted or had problems. There is a set of software for each hardware platform.

6. Key.
The key contains the following information about the problem:

Hardware on which the problem occurred. This is a three position abbreviation of the hardware. The cross reference list is the table, **NCCLOG.HWA**.

Software that had the problem. This is a three position abbreviation of the software. The cross reference list is the table, **NCCLOG.SWA**.

NCC person who logged the problem. This is a three position abbreviation of the person's name. It comes from the profile.

**ID** of the NCC PC being used. This is a '1', '2', 'a', or some other character to identify this PC.

**LOG NUMBER**. There will be a separate series of numbers generated by each NCC PC.

7. **PROBLEM DESCRIPTION**. This is a description of the caller's complaint. It is straight text, free format and has no length limit.

8. **STATUS**. New problems will all have a default status of OPEN.

9. **DISPOSITION.** This is automatically generated by the system. The information comes from the table, NCCLOG.EXP. It is the name of the office in charge of this part of the system.

10. **DATE.** This is the date the problem was logged.

11. **TIME.** The time the problem was logged.

12. **DATE CLOSED.** No date is entered in this field by this system.

**SOLUTION.** No solution is entered by this system.

Of the above fields, only numbers 1, 2, 3, 4, 5, and 7 are input by the NCC operator.

## VIII. Uploading NCCLOG to the 3B2.

At any time during the day, the log files on one or more NCC PC can be uploaded to the 3B2 and stored on the NCC central problem tracking database. To upload the data, exit the NCC system and proceed as follows:

| | |
|---|---|
| **NCCUPD** | This procedure will automatically log on to the 3B2, do some cleanup, and prompt the user when it is ready for the next step. After typing 'NCCUPD', wait until the system prompts for <ESC> and (CTRL-F3). Do not hit any keys until that message is displayed. Hit <CTRL-h> several times to move the cursor to the beginning of the prompt. Then enter, |
| **<ESC>** **<CTRL-F3>** | This will save the file that was uploaded from the PC, reformat the data, generate programs to load the information into the UNIFY database, and run the generated programs. When all is finished, the NCC Menu will be displayed. This gives access to the NCC database. At this point, the data has been successfully uploaded and loaded onto the NCC database. |

The database is designed to reject any duplicate records. If any were uploaded, these will be rejected with error messages. This will not cause any problems and the error messages should be ignored. The duplicate messages should be considered a reminder that you need to delete the NCCLOG.CUR after processing onto the UNIFY database. Otherwise that file will continue to grow.

## IX. 3B2 Problem Tracking Database Menu System.

The NCC menu system has options to run a number of queries against the database. Run as many as you want.

## X. Current Status and Future Plans

The implementation of the system that this expert system was designed to support has been delayed. Many of the critical hardware and software components are not available yet. For that reason, the real heart of the expert system is present only in skeletal form. The troubleshooting rules for the LAN and DDN are fairly complete, we think. Those are the simplest parts of the system to troubleshoot. The real problems will be the IBM mainframe, the AT&T 3B2 UNIX machines, and their respective software components. Troubleshooting rules for them are almost completely absent from the rulebases.

Future development will be in adding rules to cover the IBM and 3B2 and the 'UNKNOWN' areas, narrowing down the machine from the generic 'UNKNOWN', and tracking down the exact software/hardware problem from the specific machine's 'UNKNOWN'.

## B3 Session: Automated Knowledge Acquisition II

# From Data Rich to Information Wealthy - a CASE for CLIPS

Larry Mason
Member, Group Technical Staff
Texas Instruments

Data. Information. Webster's dictionary defines the two terms almost synonymously and in fact, uses each term in the other's definition. More important for this discussion, it associates data with facts and information with knowledge. It is this distinction that this paper will focus upon while describing a CLIPS-based system I have built that processes data into information. The goal is to leverage the power of CLIPS without requiring anyone to be trained in the CLIPS syntax or features. In short, a pseudo-CASE tool for CLIPS (see figure 1 for an overview picture).

**The Map**
Data is available to each of us in massive quantities from numerous sources in different formats. If two people perform the same job function, would it not be reasonable to expect that they access the same data and in a consistent and uniform manner? I believe the volume of data is expanding faster than our capability to effectively utilize or categorize. Therefore, the first step in this system is to map the data to its sources, along with the necessary parameters required for access. For instance, a person's home phone number as a data item has a source of the phone book but requires parameters of last name, first name and perhaps address to locate. Data is named and kept unique within the map but not actual values for data. While a detailed explanation of this mapping is outside the immediate topic of this paper, I mention it to form the basis for the remainder of the paper and as a lead-in to the use of CLIPS.

**The Problem**
Information is defined in this system as the result of some process upon data and/or other information. For example, suppose the outside temperature was recorded on each hour. This would be data and there would be 24 values for any given day. The average temperature for the same period is not data but rather information. The process was an addition of all data values and then divided by the number of values. That is simple information and does not require a very sophisticated system. However, building information from other information, which I will refer to as cascading information, requires the processes to be assembled serially or governed by some schedule for the execution order. This ordering would be trivial if a single thread of information was required. For example, hourly temperature (data) becoming daily average (information) becoming monthly average (information). The next level of difficulty would be a pyramid of information processing with multiple data names at the base with the end result, or peak, being a single point of information. More likely however, would be the same pyramid structure without the peak but instead people perform the reasoning to draw the final conclusions. To support this pyramid structure of data and information, the

system's goals are:

1. Build information through combinations of data and/or other information
2. Reduce raw data volume through summarization
3. Provide data acquisition without knowledge of the source of the data
4. Provide an effective, efficient means of cascading information to become itself a component of information

**The Definer**
Once a map exists for data, the next part of the system interacts with the user to define the components and processing techniques for information. Using the aforementioned example, information would be structured as the following:

```
'temperature_sum'   = sum of 'hourly_temperature'
'temperature_count' = count of 'hourly_temperature'
'avg_temperature    = 'temperature_sum' divided by
                      'temperature_count'
```

The following chart illustrates the format of information:

```
information = [value, field, action] {math [value, field]}
              {condition(s)}
field       = [data, information]
action      = [sum, min, max, count] field
math        = [addition, subtraction, multiplication, division]
condition   = if field relop [value, field]
relop       = [=, !=, <, <=, >, >=, member of a list, not a member
              of a list]

key:  [ , ] means one choice is allowed
      {   } means optional
```

Each information is named by the user in the same manner that names were given to data in the map step. Information can be formed in one of three ways: (1) a simple assignment of a value such as true or false which usually is followed by some condition; (2) an assignment of the value from a component field; and (3) as an action applied to a component field. Components are selected from a list of data and information names. The system continuously displays the definition on the screen as choices are being made. Up to five conditions are allowed. The relational operator within each condition is implemented using CLIPS user defined functions that handle both strings and numbers. This allows for uniform specification without concern to the type of data. Each information can have multiple definitions and presently are not analyzed for any conflicts (for example, hot = true if temp > 90; hot = false if temp > 90). The definitions are stored in a database keyed by information name.

**The Builder**
So far the system has a map of data and definitions for information. The next part produces the logic to retrieve values for data and

builds the CLIPS rules for producing information. The first version of the system only allowed for data to be post-processed into information. In the pyramid analogy, data appeared in only the base. The most recent version now allows for data to be retrieved based upon information. This is possible by scheduling CLIPS to execute within the access to data and requires separate rule files for each invocation of CLIPS. CLIPS is scheduled as few times as possible to maximize the amount of information each execution produces. The pyramid model now still has data as the base, but now allows for alternated layers of information and data.

Information can be built at different levels. In the temperature example, the average temperature may be for each day, for each month, for each year, ... This concept of 'for each' is separate from but related to the definition of the information. The definition of average temperature is constant across all of these time periods. What varies is the volume of data to be processed. This variance is specified at this stage of the system to allow information to be reusable but yet configurable. During the build step of the system, each information is assigned by the user the cross sections of data applicable to the information. These cross sections are in fact other data or information names.

Each piece of information becomes a CLIPS rule. The Left-Hand Side (LHS) of a rule is comprised of patterns for the components of the information. There is one pattern for all components that are data and one subsequent pattern for each component that is information. All data patterns begin with a keyword of "DATA" and all information patterns with "INFO". Any conditions specified for the information are also stated on the LHS using the 'test' feature of CLIPS. Currently if more than one condition has been defined, they are or'ed together. The Right-Hand Side (RHS) contains all math and assignment operations as well as an assertion of a new fact corresponding to the information.

The CLIPS salience feature is used to assure the proper order of rule firings when information is cascaded. The order is particularly important for the actions of sum, min, max and count since the final value must be computed before any cascading can occur. The salience value is calculated during the build step and placed on each rule. The limit of cascade levels is currently 995. Refer to figure 2 for examples.

**The Processor**

The final part of the system is the component that retrieves values for data and builds information. As data values are retrieved, they are stored in a database. When information needs to be built, CLIPS is invoked. The proper rules are loaded into CLIPS. Next, through user defined functions, data is extracted from the database and asserted as facts to CLIPS. Once all facts have been asserted, CLIPS is allowed to run. The CLIPS rule firing mechanism is well suited to the task by (1) assuring all components are present, (2) proper execution order through use of salience, and (3) continues until all possible information is built. At the end of all rule firings, the data facts are removed from CLIPS by activating a single rule as shown below.

```
   (defrule remove_data "remove data facts"
     (declare (salience -998))
     ?f1 <- ("DATA" $?values)
     ?f2 <- ("EOF")
     ->
     (retract ?1))
```

Once purged of data, only facts pertaining to information remain. These facts are stored back in the database that holds the data values. The system can then continue to retrieve more data values and the cycle begins again. The entire process ends when all data has been retrieved and all information built.

**Summary**

Again my goal was to aid the process of creating reusable and manageable information from data. This resulted in a system that utilizes the features of CLIPS, the rules, facts and activation firing mechanism, without requiring training in the syntax or particulars of CLIPS. The focus of the system was to build results not the mechanism by which results were produced. The system is written entirely in C and currently executes on, but is not limited to, MS-DOS and MS OS/2 platforms

By extracting the process of creating logical information from physical data, this system can front-end applications such as executive information systems, decision support systems, spreadsheets, fourth generation tools, and so forth. The applications can then deal with a higher plane of knowledge and rely upon another process for the consolidation. Furthermore, the data and information produced can be used by several of these systems while its creation occurs only once, thus improving efficiency and more importantly consistency.

Terms used:

data - factual information used as a basis for reasoning, discussion or calculation (from Webster's Ninth New Collegiate Dictionary )
named items whose values can be obtained through physical means (as used in the context of this paper)

information - knowledge obtained from investigation, study or instruction; facts, data (from Webster's Ninth New Collegiate Dictionary)
named items whose values are a result of some process(es) upon data and/or other information (as used in the context of this paper)

CLIPS - forward chaining rule system based on the Rete algorithm developed by the Artificial Intelligence Section at NASA/Johnson Space Center

map - database containing data names, source and parameters for obtaining data values cascading - the processing of hierarchical information, that is, information that is a component of other information

definer - system component that interactively builds the definition for information

builder - system component that produces CLIPS rules from the definitions and scripts to interact with data sources

processor - system component that retrieves values for data from the sources and invokes CLIPS to produce information

MS, MS-DOS are registered trademarks of Microsoft Corporation

Figure 1 - system architecture

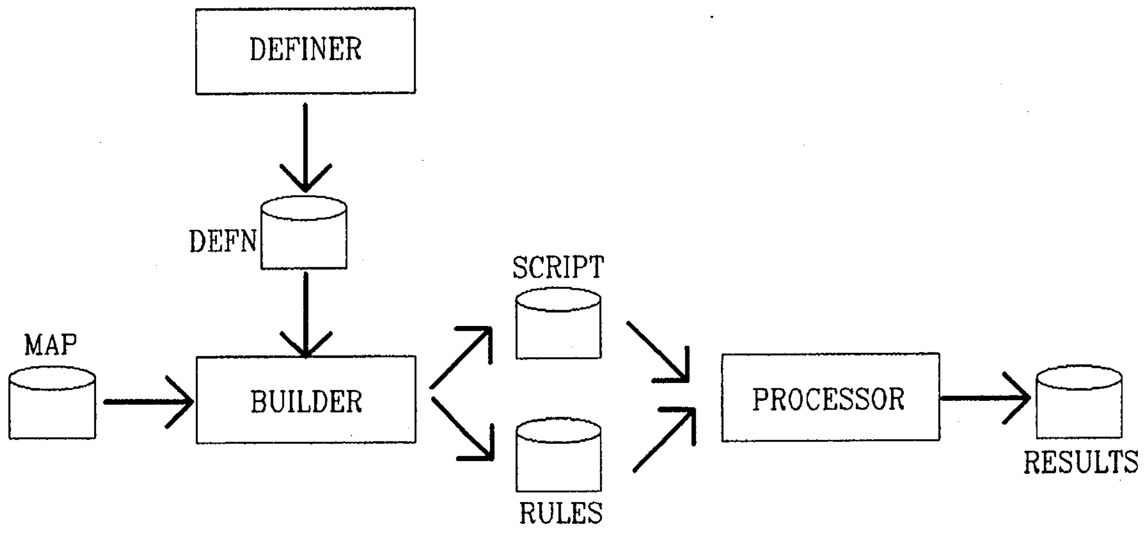


Figure 2 - samples

```
definitions:
 'monthly_salary' = data 'yearly_salary' / value 12
 'monthly_net'    = info 'monthly_salary' - data 'monthly_rent'
 'loan_approved'  = TRUE if info 'monthly_net' > data 'loan_amount'

for each:
 'monthly_salary' for each 'name'
 'monthly_net'    for each 'name'
 'loan_approved'  for each 'name'

generated rules:
 (defrule rule_1 "ASSIGN"
   (declare (salience -2))
   ?f1 <- ("DATA" "name" ?name "yearly_salary" ?yearly_salary)
   =>
   (bind ?math (/ ?yearly_salary 12))
   (assert ("INFO" "name" ?name "$ENDKEY$" "monthly_salary" ?math)))

 (defrule rule_2 "ASSIGN"
  (declare (salience -3))
  ?f1 <- ("DATA" "name" ?name "monthly_rent" ?monthly_rent)
  ?f2 <- ("INFO" "name" ?name "$ENDKEY$" "monthly_salary"
         ?monthly_salary")
  =>
  (bind ?math (- ?monthly_salary ?monthly_rent))
  (assert ("INFO" "name" ?name "$ENDKEY$" "monthly_net" ?math)))

 (defrule rule_3 "ASSIGN"
  (declare (salience -4))
  ?f1 <- ("DATA" "name" ?name "loan_amount" ?loan_amount)
  ?f2 <- ("INFO" "name" ?name "$ENDKEY$" "monthly_net" ?monthly_net")
  (test (lpm_gt ?monthly_net ?loan_amount))
  =>
  (assert ("INFO" "name" ?name "$ENDKEY$" "loan_approved" "TRUE")))
```

# Supplemental Knowledge Acquisition through External Product Interface for CLIPS

**Tim Saito**
Computer Sciences Corporation
16511 Space Center Blvd
Houston, Tx

**Stephen Ebaud**
University of Houston (Downtown Campus)
Houston, Tx

**Dr. R. Bowen Loftin**
University of Houston (Downtown Campus)
Houston, Tx

## 1. Introduction

Traditionally, the acquisition of knowledge for expert systems consisted of the interview process with the domain or subject matter expert (SME), observation of domain environment, and information gathering and research which constituted a direct form of knowledge acquisition (KA). The knowledge engineer would be responsible for accumulating pertinent information and/or knowledge from the SME(s) for input into the appropriate expert system development tool. The direct KA process may (or may not) have included forms of data or documentation to incorporate from the SME's surroundings.

The differentiation between direct KA and supplemental KA (indirect) would be the difference in the use of data. In acquiring supplemental knowledge, the knowledge engineer would access other types of evidence (manuals, documents, data files, spreadsheets, etc.) that would support the reasoning or premises of the SME. When an expert makes a decision in a particular task, one tool that may have been used to justify a recommendation, would have been a spreadsheet total or column figure. Locating specific decision points from that data within the SME's framework would constitute supplemental KA. Data used for a specific purpose in one system or environment would be used as supplemental knowledge for another, specifically a CLIPS project.

## 2. The CLIPS Environment

With the advent of new hardware and software technologies, CLIPS should be able to transform its facilities and functionality for the future. In order to work with real world applications, CLIPS must prove that it can somehow link with existing products as database management systems (DBMS), spreadsheets, word processing documents, hypermedia and hypertext, etc., for application support. For example, production expert systems for realistic engineering applications must process voluminous data and could require access to large distributed DBMSs [1]. Another project, the ANASTASIL system, used a knowledge base (KB) system for the identification of different regions of document image using a hybrid, modular knowledge representation (KR) called a geometric tree to produce an internal, editable description of the document and its contents [2]. .In this case, supplemental knowledge could be derived from this type of process into the CLIPS environment.

When outside data sources are incorporated into an expert system environment like CLIPS, the perceived dichotomy between the CLIPS-type system and other software environments like spreadsheets, DBMSs, and fourth generation and conventional languages fades. If systems only implement simple query-and-match qualified data search applications, the system would likely contain no real expertise and be difficult to control. However, if the project did demand data lookup capabilities and actual subject matter knowledge, the integration facilities of a CLIPS system could provide a reasonable environment to design systems demonstrating discernible intelligence with easy access to knowledge that already exists.

At this point, CLIPS does provide some facility for reading files from various outside sources in its "open" and "close" I/O file commands aided by the "format" instruction. However, more support would be needed to make it an easier process. Within reason, CLIPS also does offer a set of functions for text processing and hierarchical lookup facilities consisting of "fetch" and "toss" commands bounded by "entry_file_format" and "print_region" instructions.

## 3. Issues to Consider for CLIPS with External Product Interfaces as Sources of Knowledge

Whether in database, spreadsheet, hypertext, or other forms of data sources as well as existing knowledge bases (intelligent DBs, rule/fact bases, etc.), the decision as to the appropriateness of these sources to supply additional knowledge or support to existing premises by the expert would have to be determined by those in the knowledge engineer and/or expert roles. Consultation values residing in data files that already exist within the environment of the SME, in machine readable form, could serve as effective supplemental information for a CLIPS application. The capture of supplemental knowledge through existing external sources may not comprise automated KA in the formal sense, but takes advantage of work already accomplished.

The implications of one data item to a set of related data should also be determine suitability for integration purposes. Since most DBMS designs are constructed around a "record" data structure, the records become groups of logically related data items for control. Where a payroll record of an employee would include name, social security number, salary, and other information, each item is still pertinent relating employee information. CLIPS could similarly require groups of related data to conduct a consultation. In essence, although importing external files to augment current knowledge may not be suited to all applications, those that require groups of related knowledge based on a key could benefit most from supplemental KA. The process has the capability of minimizing effort in creating or modifying redundant files into the KB.

Whether or not interactive questioning is appropriate should also be determined. An application requiring sensitive data, such as personnel records and payroll to be handled by the DBMS interface rather than the user, could be considered an appropriate candidates. Enabling CLIPS to access data through a data file interface, the user could have a reasonable mechanism for consultations without requiring sensitive information from the user. However, security or authorization levels of each user should still be verified when needed.

Hardware environments play key roles in the levels and types of products interfacing possibilities. Although products like Oracle, dBASE, 1-2-3, and Word runs on various machines, there are other packages that are not so widespread or portable. Hypercard/hypermedia products like SuperCard, HyperTMon, etc., are reserved for operation in the MacIntosh domains. Digital Equipment Corporation (DEC) also has some specialized products that run on VAX equipment only. Clearly, picking products to interface with CLIPS requires some research as well as preparation, at this point.

Technical issues such as file access constraints or limitations, outside access security, password protection features, and record locks or other protection protocols in multi-user or network configurations should also be addressed when deciding, designing, and deploying interfaces for supplemental KA purposes.

## 4. Methods for Implementation - Roles of Data File Incorporation

The Selection Menu approach utilizes lists of selection items from which users choose, ranging from yes/no selections to expandable or cascading menus. Constantly changing items could be managed by putting choices in DBMS files and building menus by file links. The search strategy would target records matching specific conditions where values for building the menu are taken only from records satisfying the search requirements. For example, a CLIPS statement would create a selection menu based on a search of the dBASE DBMS file STAFF.DBA. Records would be chosen where the field "salary" would have a value greater than the value found in the CLIPS variable "minsalry". A menu would be created whose values come from the field name from each record which satisfies the search.

A Question/Answer (Q/A) Shortcut would use outside data files to minimize excessive user questioning. Answers to questions would be derived from a designated information file regarded by the SME as relevant to the application. CLIPS would be prepared to interrogate the data source as opposed to the user's data entry in appropriate cases. The user would enter one key element where CLIPS would perform the DBMS search matching the key and key fields contents. The record(s) and contents would then become assertions within CLIPS environment. Furthermore, depending on requirements and with some preparation, certain rules or facts could also be derived from the data files. However, not all systems need depend on an index method. CLIPS would be able to access a 1-2-3 ASCII or EBCDIC (mainframe) file to retrieve starting values.

Another approach, which could be considered an extension of the Q/A Shortcut method, would be the Audit and Reason approach. Rather than accessing data files, the primary product interface would be the main point of access. These types of expert systems, capable of intelligent analysis, would run in batch mode behind transaction processing systems providing "smart" data summaries. Unfortunately, the integration process would be intricate and require more sophistication in interpreting or predicting trends/changes in the data source through CLIPS. However, simplifying the reasoning would make it easier by accumulating and examining factors like averages, statistical deviations, totals, etc., on condensed versions of the data. The record-by-record stream summarized by one or more values would be managed more effectively.

In some cases, applications might require certain lines of reasoning to be processed by the inference engine before relevance is determined. This strategy of "hypothetical reasoning" refers to the segments of values that would be stored in an external source and loaded to test the relevance of the model. The scenario justifies the consultation if results of the test match the case of the user. If not, the values of the model are retracted and another is tested or the consultation is terminated. In the example of a medical diagnostic system, even where only partial evidence or facts are available, the system should be able to arrive at or suggest condition "A". The keys containing all other values implied by condition "A" would reside in an external data source. These could be loaded into the KB and appropriate attributes instantiated with values from that source. The diagnostic system would process the model in several options:

- Generate lists of additional symptoms, problem characteristics, or problem implications. The user could also be queried to verify whether the lists matches the current scenario to confirm their hypothesis.

- Use the values to suggest certain remedies and poll the user as to the suggestions effectiveness in improving the problem.

- Investigate contradictions between other user observations and those implied in the hypothetical model.

The DB front-end approach would make use of the expert system for inferencing in order to develop a profile of requirements. Although various applications can take advantage of the CLIPS rule and fact base, managing basic issues like family income and/or student work status meshed with factors like academic performance, equal opportunity regulations, and special scholarships, would serve appropriately as an intelligent front-end system. Still, careful consideration of the process should be given to use of the CLIPS facilities as a DBMS front-end. CLIPS would need the capability to examine all of the data in a file to find records matching user requirements and have common interface protocols set up. Even so, the remote possibility of CLIPS serving as a bonafide DB front-end without true DB import and export capabilities might be somewhat disappointing.

## 5. Embedded Systems

In some applications, procedural processing addresses problems satisfactorily. However, where inferencing would make sense, CLIPS and systems like it could be embedded in existing "conventional" software. Analogous to the "tail wagging the dog" syndrome, external programs would drive CLIPS as a subprogram. Adding intelligent behavior and establishing monitoring or process control programs where inference processing speed is critical would be considered key features.

In some cases, it could be debated that a software program would contain data that an embedded CLIPS routine could acquire as "knowledge on a need-to-know basis". However, the mechanics of obtaining knowledge for the embedded CLIPS functions would remain the same with the exception of the approach taken. Embedding CLIPS in C language programs is perhaps the most common practice among those developers who do embed, although the Ada language is also another candidate host language. A benchmark example of embedding CLIPS in forward-chaining mode in C for an intelligent computer-aided training (ICAT) system called Payload-assist module Deploys PD/ICAT was developed for NASA to teach mission oriented tasks [3]. There was no supplemental KA attempted in this project.

Another form of supplemental KA could come from the Computer Aided Software Engineering (CASE) approach of receiving design knowledge rationale from the C or Ada language so that the CLIPS system embedded within could more easily accept the heuristic information passed from the main software drivers. Ada or C language developers could use the inferencing components when needed. For example, a CASE tool would be able to reengineer existing C code to derive some design information which could be applied to the CLIPS code embedded within a designated application to maintain consistency of values passed between various modules in the system.

## 6. Recommendations to Consider for External Production Integration

Employ established systems analysis techniques with the CLIPS KA process to more effectively incorporate supplemental knowledge. Since CLIPS or other expert systems technologies should already be acknowledged as hybrids of knowledge-based and DP techniques, supplementing the KA process would still be appropriate.

Incorporate the KA and subsequently supplemental KA process within existing DP environments where applicable. Since CLIPS programming techniques represent just one style of software development, the verification and validation process of the CLIPS code, just like other software code, should be performed as well.

Evaluate future as well current technologies for integration needs. The design, analysis, implementation, and integration process will affect success points in existing and future projects.

Not only with KA, but issues concerning object oriented methodologies, neural networks, hardware platforms, etc., as they relate to CLIPS implementation could have large impacts on future projects.

On the other hand, vendors developing and marketing software products would also do well to establish compatible format where data or information could be more easily transferred into a CLIPS or similar expert system environment. Since CLIPS is primarily rule-based and parenthesis constrained, translation devices from a wide range of products could help to the benefit of both.

## 7. Examples of Interface Activities with External Products

Of the software products out on the market, the dBASE and Lotus packages seem to be the most popular out in the CLIPS field. The CLIPS Help Desk at Johnson Space Center, Houston, receives numerous questions concerning CLIPS file access capabilities of the more popular file management packages like dBASE and 1-2-3.

Currently, for knowledge input from external files, the CLIPS developer needed to use either the "open/close" commands within CLIPS for ASCII file access or the "fopen/fclose" commands within C for binary file access. One project interfacing CLIPS with Lotus has the developer modifying the BLOAD modules to create a smoother file loading process. One significant roadblock was how the interface would resolve the right parenthesis as delimiter. Apparently, CLIPS is not able to weed out header information left in files created by dBASE, Lotus, Oracle, etc., products.

Real-time data captured from analog-to-digital converters into database or spreadsheet format would be considered reasonable sources to be acquired for knowledge. Given the amount of scientific data that would support a physician's premises, supplemental knowledge could be exported from one system for use by CLIPS. Researchers at the Baylor College of Medicine have combined the use of CLIPS to manage specific theories with real-time data stored in Clipper DBMS databases. The overall approach was to test certain theories against specific patient profiles.

Another technology just past its infancy would be the hypertext/hypermedia technologies. Since the Macintosh world seems to have some monopoly on hypercard development, most of the interface activities have been limited to the Mac environments. Developers at the Jet Propulsion Lab have released their first cut at a hypercard system with CLIPS that can be used for input and output even with CLIPS' global data and 32K Apple hardware size limitations.

## 8. Conclusion: Issues for New Technologies

As data management product technology evolves, more features that will aid expert system development, like CLIPS, should materialize. Vendors will have to adjust to technological change as much as CLIPS developers. Where their current customers are first trying to make sense and then make use of rule based, object oriented, and other methodologies, vendors should heed such activities and produce some features or functions that help facilitate such accommodations. Some vendors like Aion Corporation, MDBS Inc., and Neuron Data have established some types of translation ports between their expert system and other expert systems and data source facilities.

More advanced technology means more adaptation on the part of the CLIPS developers, vendors, and user communities. Object oriented products and methodologies will take a stronger hold in planning, design, KRs, programming approaches, and implementations. Object oriented packages such as Ontos and Gemstone should allow more sophisticated manipulation of the data from CLIPS. Since a beta version of an object oriented version of CLIPS, designed and written at JSC, has been released April 1990, there may and should be more impact and flexibility in the ability in which CLIPS interfaces with external products and knowledge representations.

Interface with not only other KBs within the CLIPS systems but other expert system development environments like ART, ADS, VP-Expert, etc., will have to be eventually addressed as the pressure to standardize affects the expert system development environments. Even such issues as the AD/Cycle repository of IBM, much less rule base and fact formats, will have to be addressed for more constructive and sophisticated interface and knowledge sharing.

Expert system development environments will no longer be islands of activities affecting only themselves. Although expert systems were primarily scientific and engineering oriented, the realm of application will have become more widespread in other applications like medical and business environments by the 1990s [4]. An increasing number of technologies, including artificial neural systems, genetic algorithms, machine learning and induction, case-based reasoning and analogy, fuzzy logic and uncertainty, and nonmonotonicity and truth maintenance systems are also evolving requiring various levels of knowledge infusions.

## 9. Product/Vendor List

| PRODUCT marketed by: | VENDOR |
|---|---|
| Oracle | Oracle Corporation |
| dBASE | Ashton-Tate |
| 1-2-3 | Lotus, Incorporated |
| Word | Microsoft, Incorporated |
| Macintosh | Apple Computer, Incorporated |
| Ontos | Ontologic, Incorporated |
| Gemstone | Gemstone Corporation |
| ART (Automated Reasoning Tool) | Inference Corporation |
| ADS (Aion Development System) | Aion Corporation |
| VP-Expert | Paperback Software, Incorporated |

## 10. Bibliography

1 Howard, H. C., and Rehak D. R., "KADBASE: Interfacing Expert Systems with Databases", IEEE Expert, Fall 1989, pp 65-76, 1989

2 Dengel, A., and Barth, G., "ANASTASIL: A Hybrid Knowledge- based System for Document Layout Analysis", Proc. Eleventh International Joint Conference on Artificial Intelligence, pp 1249-1254, 1989.

3 Loftin, R. B., Wang, L., Baffes, P., Hua, G., "An Intelligent Training System for Space Shuttle Flight Controllers", Innovative Applications of Artificial Intelligence, H. Schorr and A. Rappaport (eds.), AAAI Press, pp 253-265, 1989.

4 Giarratano, J. C., Culbert, C., Savely, R. T., "The State of the Art for Current and Future Expert Systems Tools", ISA Transactions, Vol. 29, No. 1, pp. 17-25, 1990.

# Rule Induction Techniques

Gary B. Young

SofTech Inc,
1300 Hercules Suite 105
Houston, Texas 77058

## INTRODUCTION

Expert System technology is receiving more and more consideration these days when it comes time for selecting an implementation paradigm. As people become educated as to which **domains** Expert Systems technology can be appropriately utilized and to what extent this capability can be made useful, they will want to extend this ability to encompass greater responsibilities of their everyday routines.

**Knowledge Acquisition** is one of the most time-consuming phases of an Expert System development process. One must extract the essential knowledge from a source by whatever means, and transform this knowledge into a representation that can be demonstrated.

**Rule Induction** is one technique that can be applied to the knowledge acquisition process for certain example-based problems. Spread Sheets and Data Bases for example, may contain columns of information (*criteria*) to be used in some sort of decision making policy. These criteria could automatically be converted (*represented*) into production rules that output an appropriate decision dependant upon a given set of criteria. To generalize this concept, we could state that Rule Induction is the process of generating a **Knowledge Base** from a given set of examples.

Through the use of Rule Induction, I will explain how I took information from a database and generated rules encapsulating the information contained within the database.

## BACKGROUND

There have been numerous methods developed to automate the knowledge acquisition process in Expert Systems development without any method being the clear implementation choice. The following methods/algorithms have been identified as possible implementation paradigms: **Expertise Transfer System**, **ID3**, and **Version Space Search**. Other methods could be equally valid.

The **Expertise Transfer System** is based upon George Kelly's Personal Construct Theory. [3] gives a very high level, abstract, introduction to this theory. I found all articles leaving a little to be desired when it came to giving instructions for the construction of the initial Knowledge Base Foundation. Every article started the acquisition process from an existent knowledge base without specifying what is sufficient for an initial environment. Each article was very weak when it came to determining what the initial set of questions for the Expert would be.

The Expertise Transfer System (ETS), interviews the Expert to come up with an initial set of heuristics or rules which make up the Knowledge Base. ETS performs the initial interviewing, which Boose believes to be the most time-consuming and painful. In essence, this process elicits conclusion items from the Expert and tries to extract as many parameters supporting these decisions as possible. This is sort of like starting off with the answer and trying to come up with an adequate question. The next phase of this system was to set up a sort of ratings grid that classified the supporting criteria in relative importance to the corresponding decision.

This ratings grid technique seemed to interest a great many people due to the various combinations of this technique that are available. In essence, the ratings grid uses psychology based reasoning to analyze and classify each criteria associated with a decision.

Once a grid of parameters is established, a graph of the relationships between constructs is generated. The expert is now required to walk through each of these relationships to ensure correctness. By doing this, the expert can spot holes in the coverage as well as determine if the representation corresponds to his or her perception of the world. This is an iterative process, with the expert being required to restructure constructs that are either ambiguous or inconsistent.

This approach seems to be capable of generating rules at a high level, but the complexity of generating rules with DEEP knowledge needs to be examined.

The **ID3** algorithm builds a decision tree based upon given examples. The main feature of this algorithm over others is that it induces an optimal decision tree. That is, it uses a minimal set of traits or criteria to distinguish one example from another. The limitations of this algorithm is that the expressiveness of a decision tree is not sufficient for highly complex representations.

**Version Space Search** is an algorithm that overcomes some of the limitations of the ID3 algorithm. [4] describes an algorithm that learns general concepts given a set of positive and negative examples. These examples are used to construct a set of "versions" that correspond to the most general and most specific examples of a given rule. Thus all versions of a set of criteria that fall between the most general and most specific set of criteria, will correctly identify the action associated with these conditions.

Due to the nature of the search space, large patterns will require intense processing capabilities. I believe that being required to give a negative example for every positive one will overload the system will extraneous information that will unnecessarily degrade performance as well as productivity. Noisy data and the inability to learn disjunctive concepts could also be limiting factors.

## OVERVIEW OF APPROACH

After I finally understood the basis of the examined algorithms, I decided to implement a somewhat simpler approach. This approach is related to all of the given techniques in that it learns by example.

I decided that I would examine a database and extract the criteria used by the database to make a decision and place these criteria into production rules. Each row of tuples in the database is unique which easily translates into a 1 to 1 transformation of rows of the database to left hand side patterns of a CLIPS rule. Once these rules are generated, the user must supply the action that is to be performed. This action could be as easy as displaying what decision was made or it could modify the Knowledge Base in some way. Once the user has supplied their choice of actions, the action is loaded into the environment along with the necessary criteria. Depending upon the input criteria, the knowledge base will output the corresponding decision.

## DETAILED DESCRIPTION OF PROBLEM SOLUTION

I had to start out by inputting the tuples of the database into a data structure that could be used to transform this data into left hand side patterns of a CLIPS rule. The major obstacle to overcome here was determining how the database product structured their files.

Having deciphered the file format of the database program, I had all the knowledge with which I needed to build my system, I next needed to write a routine that translated this into CLIPS rules. Each criteria of a given tuple is transformed into a single atom of a left hand side pattern. The decision these atoms support is the basis of the right hand side of the rule. The right hand side of the rules outputs the corresponding decision of the specified criteria.

The next obstacle to overcome was how to automatically load this information into CLIPS. I went in and added functions that bypassed the command line and automatically loaded the file containing the generated rules and then reset the environment.

The last major obstacle was designing the user interface to allow various interactions with the generated knowledge base. This was probably the most difficult part to accomplish. Even though you have knowledge within the system, you must provide facilities to let users access and manipulate this information as their needs dictate.

## RESULTS

This project turned out to be more work than planned. I had planned on a very quick prototype that input the knowledge from the data base and transformed this into CLIPS rules. Once this was done, I would spend the majority of the time, creating a user interface.

What actually happened was that the majority of the time was spent converting the database into a data structure that could then be used to create rules. Once the database had been input into a structure, it was a simple matter to transform this information into CLIPS rules.

## CONCLUSIONS

Automated rule induction can be a very useful and time-saving measure when applied correctly. Whether one acquires knowledge directly from spread sheets and databases, or if a system interacts with an Expert, the outcome is the same. Automated knowledge acquisition has found it niche in the programming community.

The important question to answer is which technique to use for a given situation. My program for example, is a straight forward translation. A system with highly complex representations will require a much more sophisticated technique.

As with every other system, you must have a system that will satisfy the users requirements.

## References

[1] Baim, P. W., "*A Method for Attribute Selection in Inductive Learning Systems*" IEEE Transactions on Pattern Analysis and Machine Intelligence, Nov 88, Vol 10, #6.

[2] Bergadano, F., Giordana, A., Saitta, L., "*Automated Concept Acquisition in Noisy Environments*" IEEE Transactions on Pattern Analysis and Machine Intelligence, July 88, Vol 10, #4.

[3] Boose, J. H., "*Personal Construct Theory and the Transfer of Human Expertise,*" in Proceedings of AAAI-84, Austin, Texas, 1984 .

[4] Davis, R., "*Interactive Transfer of Expertise: Acquisition of New Inference Rules*" Stanford University, California.

[5] Gaines, B. R., "*An Overview of Knowledge Acquisition and Transfer*" Proceedings of the AAAI Knowledge Acquisition Workshop, Banff, Canada, 1986.

[6] Mitchell, T., "*Version Spaces: A Candidate Elimination Approach to Rule Learning*".

[7] Parsaye, P., "*Acquiring & Verifying Knowledge Acquisition*" AI Expert , May 1988.

## A4 Session: Verification and Validation

# Enforcing Compatibility and Constraint Conditions and Information Retrieval at the Design Action

**Ravi M. Rangan,** *Graduate Research Assistant*

*George W. Woodruff School of Mechanical Engineering*
*Georgia Institute of Technology, Atlanta, GA 30332*

## ABSTRACT

The design of complex entities is a multidisciplinary process involving several interacting groups and disciplines. There is a need to integrate the data in such environments to enhance the collaboration between these groups and to enforce compatibility between dependent data entities. This paper discusses the implementation of a workstation based CAD system that is integrated with a DBMS and an expert system, CLIPS, (both implemented on a mini computer) to provide such collaborative and compatibility enforcement capabilities. The current implementation allows for a three way link between the CAD system, the DBMS and CLIPS. The engineering design process associated with the design and fabrication of sheet metal housing for computers in a large computer manufacturing facility provides the basis for this prototype system.

## INTRODUCTION

Computer aided Design (CAD) systems have chiefly developed as geometric modeling tools and have failed to adequately recognize the needs of the *engineering design process.* The design process demands collaborative capabilities that seek to externalize the information needs of the designer as well as mechanisms that ensure compatibility between various design agents. As a result, there is a need to enhance the capabilities of existing CAD systems to cater to such design process requirements. Such capabilities are dependent upon the design organization, and it is therefore required that these organizations be well studied preparatory to such development efforts. Design organizations consolidate several active design agents that may work in a concurrent or serial mode that may or may not be independent. Pahl and Beitz [1] stress that the need for interdisciplinary collaboration is imperative to design in large scale organizations. Furthermore, they note that the ready availability of a wide range of comprehensive and problem-oriented information is of utmost importance in the design process.

A basic criteria in designing such integrated CAD systems is the availability of several *design process oriented* capabilities at the source of the *design action.* (we use the term *design action* to refer to the interaction between the designer and the CAD system and to signify the point of influence of the designer in generating the design definition). In other words, the CAD systems must support the demands of the engineering design process in multidisciplinary environments in addition to routine geometric modeling functions. In this paper we shall discuss how this has been accomplished in a prototype CAD system by integrating functionalities such as a database management system (DBMS) and an expert system (CLIPS [2]) with a geometric modeler.

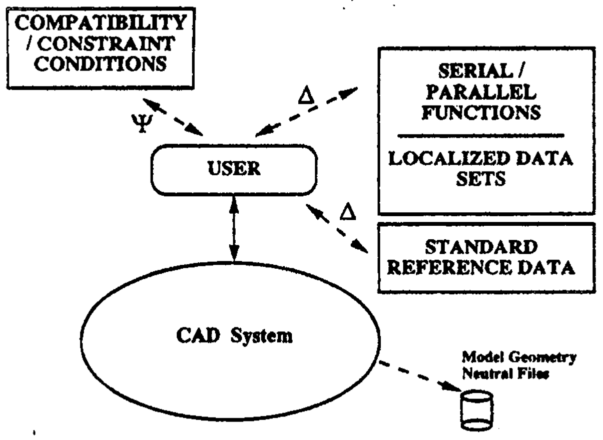

COMPATIBILITY / CONSTRAINT CONDITIONS
Ψ
Δ
SERIAL / PARALLEL FUNCTIONS
LOCALIZED DATA SETS
USER
Δ
STANDARD REFERENCE DATA
CAD System
Model Geometry Neutral Files

*Figure 1 : Manual Design System and Environment*

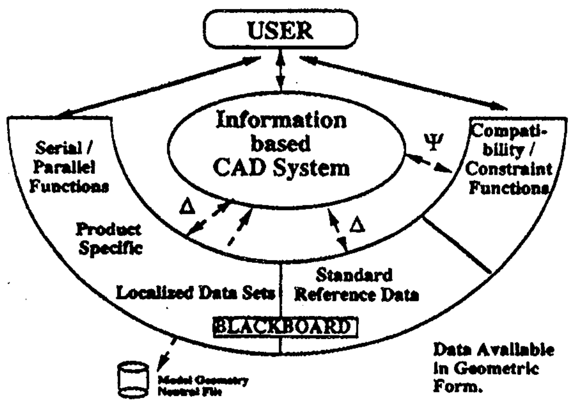

USER
Information based CAD System
Serial / Parallel Functions
Ψ
Compatibility / Constraint Functions
Δ
Δ
Product Specific
Localized Data Sets
Standard Reference Data
BLACKBOARD
Model Geometry Neutral File
Data Available in Geometric Form.

*Figure 2 : Integration of Design Action with Environment*

Figure 1 and figure 2 illustrate the nature of interaction between the user and the design environment. Figure 1 assumes that the designer is forced to interact with the environment in a manual mode - it is apparent that the lack of data integration with respect to the upstream and downstream functions may result in repetition and data inconsistency.

The integrated system shown in figure 2 accounts for this inconsistency by integrating the design action with a blackboard. In this environment, the user interacts with a CAD system, which in turn is intimately integrated with both the geometric database as well as a blackboard. The life cycle data is maintained by several interacting disciplines and is indicative of the current state of the design. Such a configuration ensures that the design action has access to up-to-date design data originating from different participants or teams in a distributed design environment. It must also be possible to manipulate these databases from the design action. The compatibility requirement between the life cycle data and the geometric database dictates that changes in one system trigger changes in the other system as appropriate. This configuration significantly externalizes the data communication and consistency requirement that is required of design processes involved in multidisciplinary design by teams.

The information management problem as related to design organizations may then be stated as follows:.

- The need to provide collaborative capabilities across the design life cycle and facilitate interactions with participating disciplines from the source of the design action.
- The need to decrease design uncertainty by incorporating compatibility and constraint conditions (across discipline and function boundaries) at the source of the design action.
- The need to provide capabilities to manipulate design objects in the context of the above two requirements.

The prototype CAD system has been developed in a case study environment involved with the design and manufacture of computers in a large computer manufacturing facility in the U.S.

## THE CASE STUDY ENVIRONMENT

Figure 3 shows a typical IDEF 0 model [3] of the activities at the case study site at NCR, Wichita.

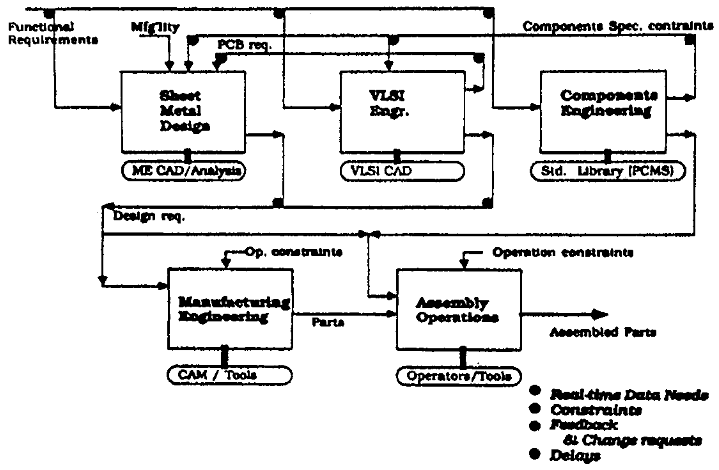


*Figure 3 : Activity Model of Typical Design / Manufacturing Operations*

The sheet metal design function receives a requirements specification as its input data and also receives configuration data from the VLSI design group, the components group, the systems engineering group (not shown) on a continued basis during the early stages of design. This is a dynamic process and there is a high incidence of data interactions and design iterations. On completion of the design, the detail design data is communicated to the various serial functions (manufacturing and systems assembly). Thus, the mechanical design functions are implicitly constrained by the capabilities of the manufacturing groups. The exact nature of these interactions is shown in data flow diagrams in [4].

## Scope :Mechanical Design

The designers are assigned the task of packaging electro-mechanical and electronic components while maintaining constraints originating from process limitations ( assembly and manufacture) and electro-magnetic compatibility requirements and human factors. In the context of this case study, the functionalities of the CAD system have been analyzed from the mechanical design perspective. The disciplines involved in this research effort included Mechanical Design, Manufacturing Engineering, Components Engineering, Drive Engineering and Sub-Systems Engineering.

## The Case Part

The rear panels of several disk file sub-systems were identified as the 'case part'. These parts are rich with features and they imply several modes of interaction for associated data. By studying the interactions between these disciplines in the design and manufacture of the *case part*, dependencies between design features and macro definitions from the various disciplines were determined. Figure 4 shows a typical rear panel used in processor and disk file sub-systems. These parts have several features which are defined by design parameters specified in related serial and parallel groups. For example, consider the slot for the fan and its mounting hole - this feature is defined based on standard component specification sheets available from component databases. Furthermore, the fasteners and screws used to mount the fan on the panel are specified by internal standards documents maintained by the manufacturing groups. Figure 5 depicts a part showing the related group interfaces required to determine the feature definition.

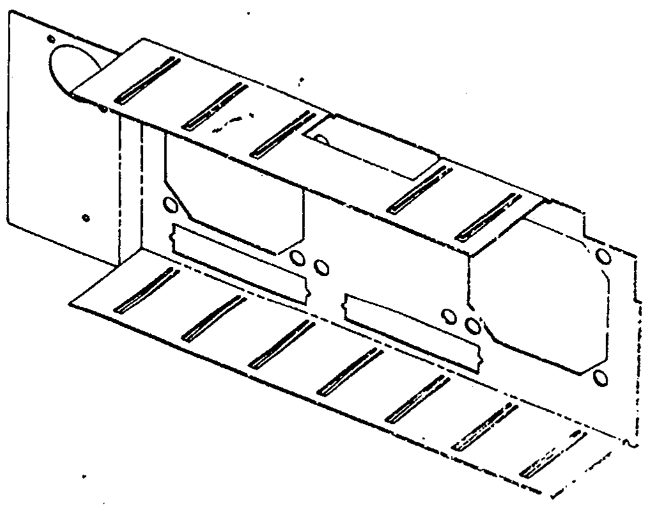
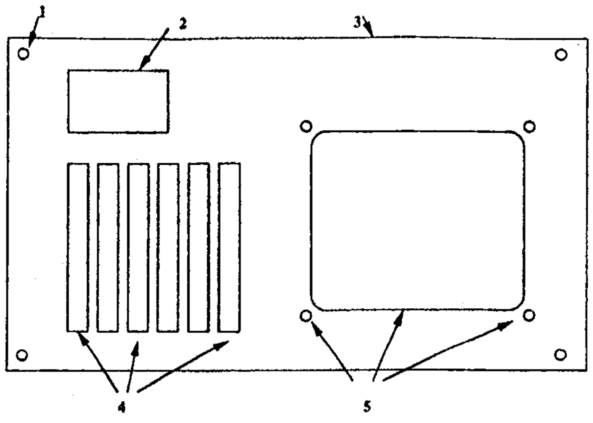


*Figure4 : Typical CASE PART : REAR PANEL*



| Interacting Group | Dependent Data |
|---|---|
| 1 . (Manufacturing, Sys Assy), | Standard Punch Size |
| 2. (Components Engr., Sub-SYS Engr.), | Switch Mount |
| 3. (Manufacturing) | Material Gauge |
| 4. (Hardware Board Design) | Controller Adapter Boards |
| 5. (Components Engr., Sub-Systems Engr.) | Fan cutout & Mounting holes |

*Figure 5 : Typical Data Interactions in REAR PANEL*

Certain compatibility and constraint conditions must not be violated. For example if the fan specifications are changed, the intra-hole spacing must be within the allowable limits. The need for multidisciplinary data as well as the enforcement of compatibility conditions is apparent. The data structures needed to support these requirements are complex since additional data representative of functional requirements and compatibility parameters must be added to the form definition ( the geometry). We see that the significance of the *case part* is far more profound when viewed against the engineering design process as a whole.

Two themes emerge from this discussion :

Integration of data originating from varied sources and in different forms

Enforcement of compatibility and constraint conditions between different design agent objectives.

## DATA INTEGRATION

As a means of providing data integration, the EMTRIS (Engineering, Manufacturing Technical Relational Information System) system [4] was developed to manage, coordinate and control design and manufacturing information in such design organizations. EMTRIS attempted to provide a collaborative mechanism by allowing different design agents to interact during the various stages of the design process and by managing and controlling the design life cycle data. However, this system is not well integrated with the design action. As a result, the designer is forced to move from the CAD system to access EMTRIS. In order to eliminate this inconvenience, it was decided to develop facilities within the CAD system, that permits the integration of EMTRIS-like capabilities with the design action.

Within the context of this prototype, EMTRIS substitutes as a blackboard for several active design agents. A blackboard is essentially a shared database that defines the state of the product design to an expandable community of design agents. This state is an aggregation of the contributions from several design agents. The blackboard therefore provides a forum for tracking the status of the design at any point in time. In addition to this, the blackboard provides control strategies to ensure consistency between data originating from several design agents.

Figure 6 depicts the blackboard architecture in this prototype system which also provides conceptual centralization of data. Data from previous designs, change requests, notes and related product information are available. Also available are a series of standard reference catalogs maintained by different groups. (e.g. tools, fasteners, standard components). The blackboard objects are stored in a bill of material (BOM) structure and as standard reference data sets.

To capture such representations within the CAD system, and to allow blackboard interactions, these entities are modeled as geometric entities.

*Geometric Entity* ( Pointer, Type, Positional_attributes, ER attributes, Descriptive_attributes)
Attributes define an entity's description while functions define an entity's behavior. Within our context, we shall use textual values (numeric or text) to characterize an entity's attributes. The actual geometry representation is referred to as the the geometric object. A geometric entity is denoted by the geometric object including the attributes. Figure 7 illustrates a typical geometric entity and differentiates the geometric object from the attributes.

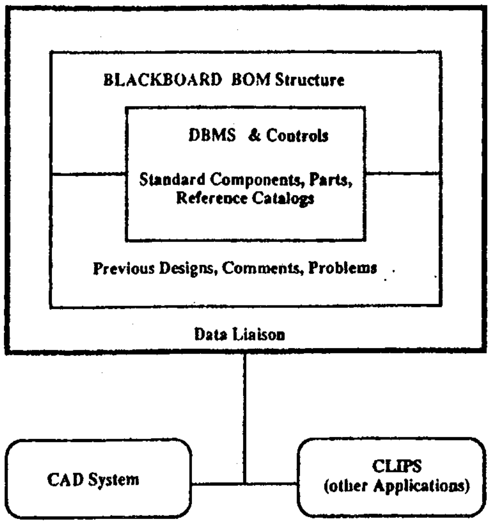
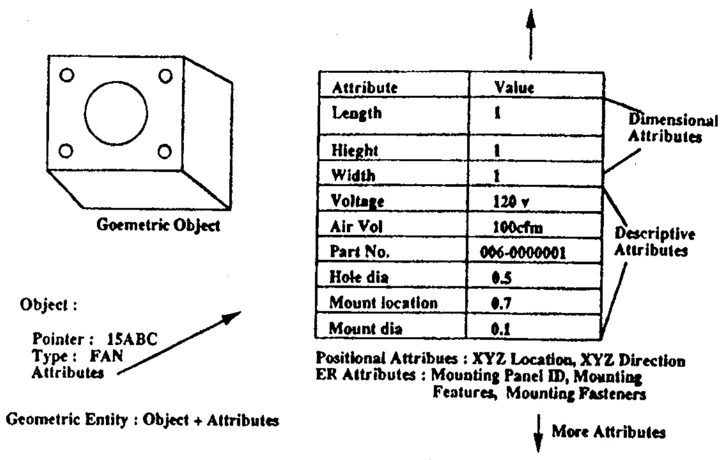

BLACKBOARD BOM Structure
DBMS & Controls
Standard Components, Parts, Reference Catalogs
Previous Designs, Comments, Problems
Data Liaison
CAD System
CLIPS (other Applications)

*Figure 6 : Conceptual Blackboard Architecture*



| Attribute | Value |
|---|---|
| Length | 1 |
| Hieght | 1 |
| Width | 1 |
| Voltage | 120 v |
| Air Vol | 100cfm |
| Part No. | 006-0000001 |
| Hole dia | 0.5 |
| Mount location | 0.7 |
| Mount dia | 0.1 |

Positional Attribues : XYZ Location, XYZ Direction
ER Attributes : Mounting Panel ID, Mounting Features, Mounting Fasteners

*Figure 7 : A Typical Geometric Entity*

The *pointer* is used to access the geometric object while the *type* broadly classifies the entity. The pointer is primarily used to manipulate the geometric object based on the designer's needs.

The geometric entity is an aggregation of *attribute values* and a *geometric object.* We shall classify these attributes into three types :

The *positional attributes* define the actual geometry of the product and control the rendering of the geometry in space.

The *ER attributes* establish the connectivity between different geometric objects.

The *descriptive attributes* assign functional and descriptive meaning to the geometric objects. In addition to providing the infrastructure for enforcing the constraint network, these attributes also provide the designer with information about the object that would usually appear in the form of engineering notes.

**Instantiation of Geometric Entities**

Geometric entities may be instantiated in any one of three ways :
*External Instantiation* : These instantiations result from interactions with the blackboard either through the BOM structure or the standard reference catalogs. These objects are usually defined by serial or parallel design agents. The positional attributes are defined by the designer at the time of instantiation while the descriptive attributes are imported from the blackboard structure. For example, the PCB and the fans are external instantiations for mechanical design.

*Local Instantiation* : These objects are defined by the designer during the design process. The positional and dimensional attributes are defined at the time of instantiation. Descriptive attributes are either defined locally or may use values from the blackboard. For example, the instantiation of the object: "sheet metal panel" may use descriptive values from standard stock items ( gauge size) maintained by manufacturing engineering.

*Dependent Instantiation* : Objects are sometimes defined with relation to some other objects. The positional, dimensional and ER attributes may be defined from constraint mappings. The descriptive attributes may either by manually defined or may be predefined by process limitations or other constraints. For example, the location and dimensions of the fan cutout feature is dependent on the location and orientation of the fan in space.

## COMPATIBILITY ENFORCEMENT

The entity relationship (ER) model [5] is used to model data dependencies between data originating from different design agents. The relationships between geometric entities in the CAD systems are modeled using constraint networks [6, 7]. The ER diagram for the rear panel identifies relationships between classes of objects (Figure 8). Figure 9 shows a constraint network for the class of rear panels. The relationships identified within the constraint network are modeled as functional relationships and are largely rule based. These relationships rely on data from the geometry definition, data obtained from the blackboard (EMTRIS) and may be implemented within the CAD system or using a rule based expert system such as CLIPS.

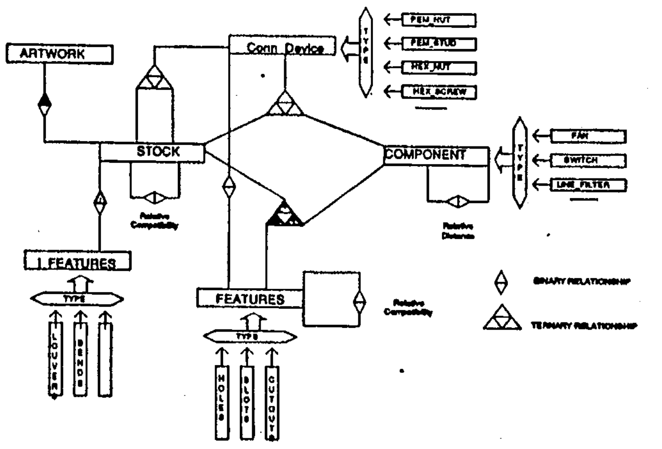


*Figure 8: ER Model for Rear Panel Objects*

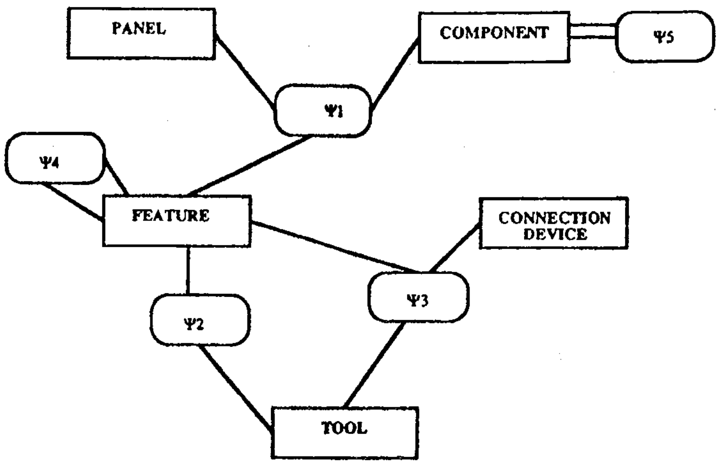


*Figure 9 : Rear Panel Constraint Network*

The panel object is a local instantiation with material properties imported from the standard stock database defined in the blackboard. This database is maintained by manufacturing engineering.

The component object is an external instantiation either from the BOM structure in the blackboard ( as in the case of the PCB definition by the hardware board group) or the standard components database maintained by the components engineering group.

The *features* may be local instantiations from a standard set of features established from process capabilities stored within the CAD database or may be automatically instantiated based on the relation $\Psi_1$.

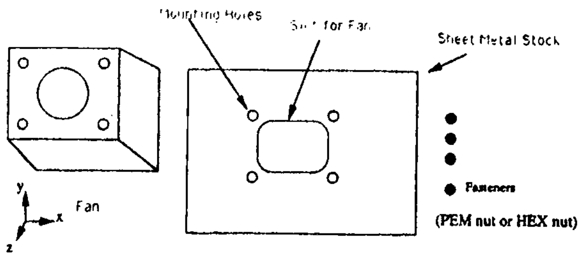
$\Psi_1$ : Verify that Component normal vector is parallel to panel normal vector
Determine intersection of *component* normal vector with *panel* plane
Project *component* dependent feature onto *panel* plane.

The dependency between the tools and the features limit the number of viable features

$\Psi_2$ : Establish viable feature set based on available tools - The available tools are stored in a database maintained by manufacturing engineering.

The *connection devices* may be local instantiations from a standard set of connection devices maintained by the manufacturing and system assembly groups or may be instantiated following $\Psi_3$ .

$\Psi_3$ : given Mounting specification for component
select connection device compatible with mounting hole
determine hole requirements for connection device
select tool to create hole and define mounting hole feature



**User Operation** : Pick Fan and Stock from CAD system
**System Operation** :
Determine Location, orientation and type of feature
Determine Mount Hole Dia from Fan attributes
Choose Screw Type with MAX (major_dia) where
major_dia < Mount Hole Dia from Database
Select appropriate PEM nut or HEX nut from Database
Choose Punch Tool where Punch Dia = PEM hole dia from Database
INSERT "slot cutout" and "mounting hole" as per these
attributes

***Figure 10 : Automated Feature Insertions and Selection of Connection Devices***

The following constraints are enforced using CLIPS and facilitate functional and manufacturability considerations:
$\Psi_4$ : feature to feature spacing constraint based on machine tool capabilities
$\Psi_5$ : component to component spacing constraint based on electro magnetic compatibility and functional requirements

## Constraint Propagation

Let us consider constraint propagation effects: they may occur due to local changes initiated by the designer or by changes posted on the blackboard. Changes in stock parameters, connection device parameters, component parameters or types, and tools will result in appropriate actions as defined in the constraint network. For example, a change in cooling requirements may call for a fan with a higher air flow capacity. This will in turn cause changes in the slot feature, the mounting hole features, the fasteners and the tools (punches, etc.). In addition to these changes, the updates must be posted on the blackboard's BOM structure. Similar changes occur when a component is deleted - all associated features must also be deleted from the geometry model as well as the blackboard. This is implemented by keeping track of the ER attributes and applying the constraint relationships on the geometric objects.

## THE PROTOTYPE SYSTEM

Figure 11 illustrates the layout of the hardware and the accompanying software. The CAD system[8] runs on a PS/2 compatible workstation while the blackboard is implemented on a UNIX based mini-computer (NCR-Tower). These mini-computers are distributed and are linked through the factory network system. The CLIPS expert system shell and the ORACLE database management system [9] are operational on the mini-computer. The link between the CAD system and the blackboard is achieved using the TCP/IP [10] network protocol.

The functional Model of the prototype system is shown in Figure 12. The user interacts with the system through a user interface which is essentially menu driven and supports pop up menus, dialog boxes and menus. The primary user interface is the CAD system. The user may also access the blackboard structure and the EMTRIS database directly from the CAD system. The communication between the blackboard and the CAD system is controlled by the user. The user is intimated whenever relevant changes are made on the blackboard. The user is allowed to post notes on the blackboard directed to different groups - these notes are referenced by the product ID.

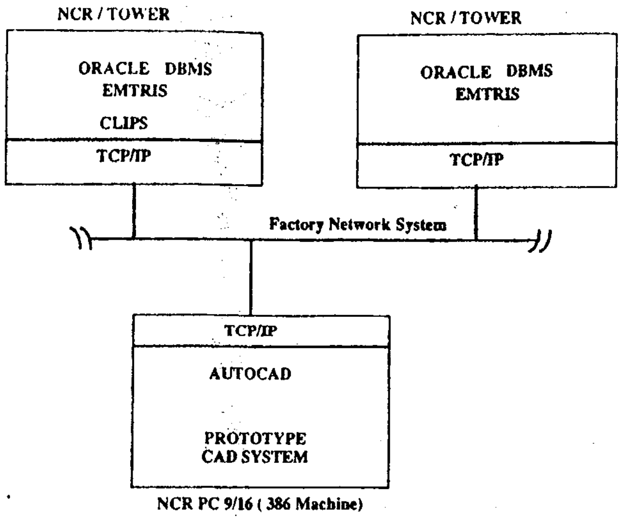


*Figure 11 : Prototype Hardware / Software Configuration*

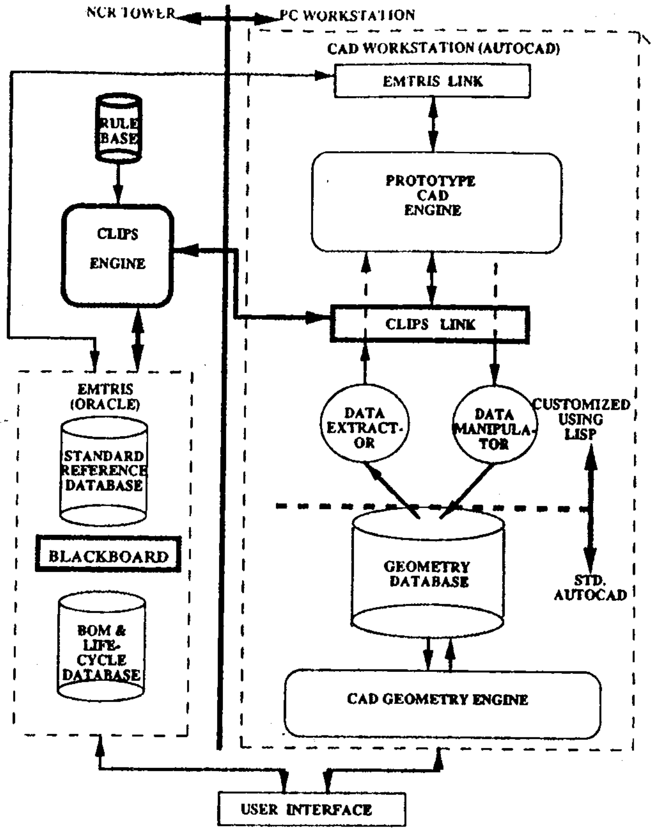


*Figure 12 : The Prototype CAD System*

The CAD data defined by the user is stored in the geometry database. The user may invoke several functions to act on this data such as typical geometric transformations and the creation and manipulation of geometric entities. The data from this geometric database is allowed to interact with the blackboard databases using routines defined in the prototype CAD engine and the communication network. This is a two way link - data may be imported from the blackboard to the geometric database or may be exported from the geometric database to the blackboard. The prototype CAD engine contains routines to automatically operate on this imported data. The blackboard also has routines to act on the exported data to ensure consistency, etc. For example, the *PCB* parameters may be imported from the blackboard and rendered on the CAD system: on updating a *fan* from the blackboard, the fan cutout and mounting hole features as well as the connection devices (standard fasteners) are also automatically updated by these routines, thereby updating the relevant segments of the geometric database. Data from the geometric database is processed by CLIPS in a similar way. The CLIPS system has also been enhanced to communicate and post data on the blackboard. The output information from CLIPS may then be used to update the geometry database.

Any update made to the geometry database is instantaneously rendered on the screen. The functions of the system are geared to respond in real time. Local functions that do not involve the blackboard or the CLIPS expert system may also change the geometry database and these changes are posted on the blackboard as appropriate under the control of the designer. These programs are written in LISP, C, UNIX shell scripts and PC batch files.

### Prototype System Capabilities

Figure 13 shows the menu structure for the prototype CAD system.

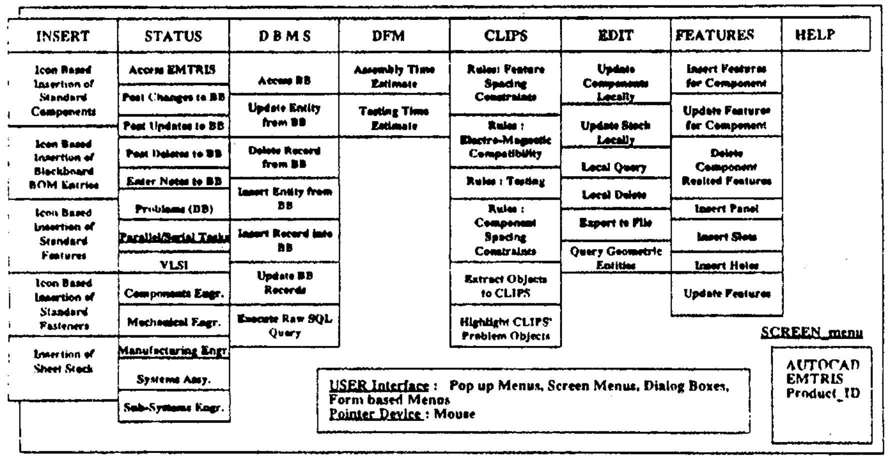


*Figure 13 : Prototype Menu Structure*

Some of the key functionalities of the CAD system are enumerated below

- *Attachment of textual attributes to geometric entities. Encapsulation of form and function.*
- *Query facility for geometric entities*
- *Blackboard interaction with CAD system:*
  *Automated information transfer between serial and parallel groups :*
  *insert geometric entity from standards DBMS ; insert geometrical entity from product specific Data (automated UPDATES, DELETES, INSERTS)*
- *Direct query link between CAD system geometric entities and Blackboard DBMS attributes*
- *Access to EMTRIS from the Design Action*
- *Maintenance of geometric associations and consistency*
- *Automated form feature generation*
- *Feedback on Assembly/ Test time (DFA/DFM)*
- *Manipulation of design features*
- *Verification of geometry by CLIPS*

A detailed account of these functions is given in [11].

## The CLIPS Module

Let us consider the design of a rear panel to support certain components such as fans, controller boards, switches, etc. Several manufacturable features become apparent. Figure 14 shows such a part which has been designed using the prototype CAD system. Using the pull down menu in the CAD system, the user selects primitive *Panel* elements and features such as *slots* and *holes*. The designer specifies the location of the panel in space, thereby automatically specifying the positional attributes. The descriptive attributes are automatically imported from the blackboard.

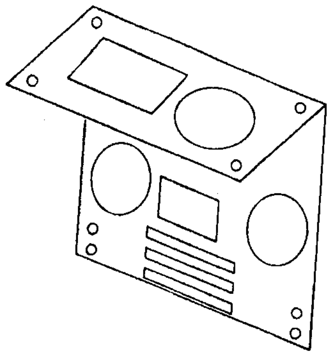


*Figure 14 : Panel Example*

The features may be inserted automatically based on the constraint network or may be manually specified by the designer. In all cases, the attribute values are displayed in a dialog box on instantiation. They may subsequently be queried or modified. Whenever a feature is inserted in a panel, the following checks are performed within the CAD system :

1. The features must be defined within the extents of the panel.
2. The feature dimensions must be greater than the stock thickness. For example, if the hole diameter is less than the stock thickness, it violates this condition. Physically there is danger of tool breakage if this condition is not maintained.

These conditions ensure that the features are instantiated with respect to the sheet metal stock. The user is given the option of repeating the operation if these conditions are violated.

Consequently, the user may invoke the CLIPS module on the NCR-Tower using the pull down menu option. This results in the extraction of the positional, ER and descriptive attributes from the geometry database in the CAD system. These extracted values are passed on to CLIPS over the network. A typical extract file is shown below - the format of this file has been designed to conform to the *deftemplate* structure defined within CLIPS.

```
FEATURE     "PANEL"      "92"       -9.19403e-17 -1.53957e-17 0.0       -1.0         0.0
            -1.60787e-16 9.0        5.0          0.0478       "92"      "18 Ga. CRS AISI 1010"
            "0.0478"     "9"        "5.000"      "0.004"      "0.004"   "0.004"      "348000002A"
            "none"
FEATURE     "P_HOLE"     "182"      6.98179      3.04243      0.0       0.0          0.0
            1.0          0.569      0.569        0.0478       "5C"      "0.569"      "0.0478"
            "182"        "Tool #"   "Yes"        "None"

FEATURE     "P_SLOT"     "176"      7.92125      3.67639      0.0       0.0          0.0
            1.0          1.15       0.257        0.0478       "5C"      "0.0478"     "1.150"
            "0.257"      "0.004"    "0.004"      "176"        "Tool #"  "Yes"        "None"
```

***Extract File Format***

This data is submitted to the CLIPS module, which loads the appropriate knowledge base. The data is then asserted into the fact base as per rules defined in the knowledge base. The following rules show how these values are asserted as template facts:

```
(deftemplate FEATURE "Template to store clips.in from ACAD drawing"
  (field description (type STRING) (default ?NONE))
  (field handle (type STRING) (default ?NONE))
  (field xins (type NUMBER) (default ?NONE))
  (field yins (type NUMBER) (default ?NONE))
  (field zins (type NUMBER) (default ?NONE))
(field xdir (type NUMBER) (default ?NONE))
(field ydir (type NUMBER) (default ?NONE))
(field zdir (type NUMBER) (default ?NONE))
(field xscale (type NUMBER) (default ?NONE))
(field yscale (type NUMBER) (default ?NONE))
(field zscale (type NUMBER) (default ?NONE))
(multi-field attributes (type ?VARIABLE) (default ?NONE)))

(defrule read-template " Read Template values from CLIPS.IN file & assert into fact base"
  (initial-fact)
  =>
  (system "rm acad_dat.clp")
  (open "clips.in" mydata)
  (while (neq (bind ?input-str (readline mydata)) EOF)
    (fprintout t ?input-str crlf)
    (temp_assert ?input-str)) ; temp_assert function to assert template facts
  (close mydata))
```

---

```
int temp_assert() ; user defined function
      {
      char *arg1;
      int num_passed;
      num_passed=num_args();
      arg1=rstring(1);
      assert (arg1);
      return (1);                }
```

---

*Read Template Rules*

Consequently, the relevant rules are activated. Currently the knowledge base contains rules to check for sheet metal manufacturability. Figure 15 illustrates the feature to feature spacing requirement and the feature to edge spacing requirement [12].

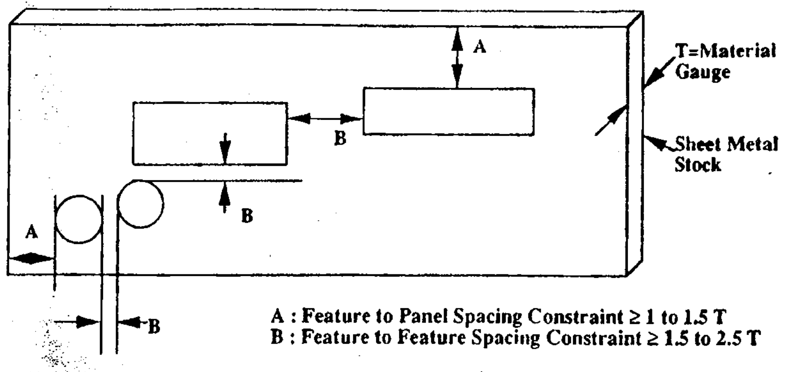

A
B
B
A
B
T=Material Gauge
Sheet Metal Stock
A : Feature to Panel Spacing Constraint ≥ 1 to 1.5 T
B : Feature to Feature Spacing Constraint ≥ 1.5 to 2.5 T

*Figure 15 : Spacing Rules*

Data may be retrieved from the blackboard (the DBMS) using the user defined function *sql_link*. ***sql_link*** submits any SQL statement to the DBMS and returns the result of the statement. In the rule shown below, the fact: (STOCK ?stock_no) has the effect of activating the *SQL-stock-assert* rule. i.e. whenever stock data is not available, the values are extracted from the blackboard (DBMS) and asserted as stock data.:

---

```
(defrule SQL-stock-assert "Retrieve Stock data from ORACLE"
      (STOCK ?stock-no)
```

```
        (not (?stock-no $?stock-data))
=>

; Retrieve Panel sheet stock data from ORACLE

        (bind ?stock-qry (str-cat "select * from stock_tol where stock# ='" ?stock-no "'") )
        (bind ?stock-tol (sql_link ?stock-qry) )
        (if (eq ?stock-tol "ERROR")
          then
                        (fprintout t "SQL ERROR - no values retrieved" crlf)
                        (fprintout t " PROGRAM  T E R M I N A T E D !! " crlf)
                        (halt)
         else
                        (fprintout t " Thge DBMS value is " ?stock-tol crlf)
                        (str-assert ?stock-tol)
        )) ; end IF
```

*EXTRACT DATA FROM DBMS*

The function **sql_link** is a user defined function written in Pro*C [9] and permits dynamic interaction between CLIPS and the blackboard. A few sample queries and their responses are shown below:

```
SQL> select * from stock_tol
STOCK#          FEATURE_TOL  EDGE_TOL
--------------------   -----------   ----------
348000001A         .08975    .05385
348000002A          .1195    .0717
348000003A          .1495    .0897
348000004A         .18675    .11205
348000005A         .22425    .13455

5 records selected.

SQL>select punch.tool#, screwlt.part#, major_dia, pem_screw.screw#, pem#,pem_screw.hole,
pem_screw.min_thk, pem_screw.ctr_edge from pem_screw, punch , hexscrew, screwlt
where punch.hole = pem_screw.hole and pem_screw.screw# = hexscrew.screw# and
pem_screw.screw# = screwlt.screw#
and screwlt.length = 0.5 and min_thk > 0.05
```

*Sample SQL Queries*

Having obtained values from the DBMS, these values may be used as normal facts. In this example, the stock tolerance values are used to check the distance parameters. If these conditions are violated, the corresponding handles are logged into a file and are subsequently passed to the CAD system.

```
(defrule write-handles "write problem handles to file"

                (?e PROBLEM ?h1 ?h2)
=>
                (open "acad_dat.clp" clipout "a")
                (fprintout t "writing values to file ..." crlf)
                (fprintout clipout "\"" ?e "\"" ?h1 ?h2 crlf)
                (close clipout) )
```

*file : acad_dat.clp:*

```
"SLOT-FEATURE""7D""182"
"SLOT-FEATURE""7D""176"
"SLOT-FEATURE""DD""FB"
```

*Write Data to Files*

Control is then transferred to the CAD system which highlights all the problem *object pairs* in the geometry model as identified by CLIPS - allowing the designer to take appropriate action.

```
(defun clips_err ()

	;
	;Reads handles in acad_dat.clp output file from CLIPS
	; and selects appropriate geometric objects from Model
	; Function readclp() reads into list: clipsdat

	(readclp "acad_dat.clp") ; values in CLIPSDAT
(prompt " The Following pairs of Entities do not satisty SPACING TOLERANCES" )
(terpri)
(prompt "Press ENTER to Continue")
(read-char)

	(setq clips_ent (car CLIPSDAT))

(while clips_ent
	(command "SELECT" (handent (nth 1 CLIPS_ent)) (handent (nth 2 CLIPS_ent)) pause)
	(setq CLIPSDAT (cdr CLIPSDAT))
	(setq clips_ent (car CLIPSDAT))
))
```

*AUTOLISP File : Read CLIPS output*

By this process, the designer is able to invoke CLIPS to evaluate the design against potential manufacturability conflicts. The primary purpose here has been to demonstrate the integration of geometry data and blackboard database values with rules in CLIPS. By integrating CLIPS with the DBMS and by exploiting the template structure within CLIPS, it becomes possible to allow the CAD system to function integrally with both the blackboard as well as CLIPS. Although this example has focused on a simple *panel-feature* problem, the principle of integration of functions (CAD, DBMS, expert systems) is apparent. The distributed configuration provides the relevance in multidisciplinary design and manufacturing environments.

## CONCLUSIONS

This paper describes briefly, the capabilities of a CAD system that supports the engineering design process by providing information retrieval and compatibility enforcement capabilities at the source of the design action. Such facilities become particularly significant in large design organizations where the design tasks are decomposed into several interacting sub-tasks. By providing such functions at the source of the design action, it is possible to create a design environment that externalizes the designer's information retrieval and compatibility verification functions.

The CLIPS application, which runs on an NCR-Tower interacts with the blackboard DBMS (implemented using ORACLE on an NCR-Tower) for relevant design data and provides feedback to the designer based on the geometry in the prototype CAD system(implemented using AUTOCAD on an NCR PC/916). This three way link allows the designer to apply rules on the defined geometry and to check for consistency with data defined by other design groups. Currently, the rules reflect manufacturability and electromagnetic compatibility guidelines for a class for *rear panels* used in computers and disk file sub-systems.

By treating geometric entities as object-attribute-value triplets and by supporting relationships between different objects and attributes, using entity relationship models and constraint networks, it is possible to impose functional and geometry based constraints within the CAD environment. By integrating such systems with applications like CLIPS, it becomes possible to add a new dimension to the CAD system. The prototype CAD system described in this paper provides access to a common design-manufacturing blackboard structure and an interface where geometric data in the CAD system is derived from parametric text data stored within the blackboard. Consistency between the two systems (updates, inserts, deletes) is also maintained.

The prototype CAD system described here was well received by the various participating groups at the case study site, and is considered to be a critical link in providing concurrent engineering capabilities.

## REFERENCES

[1] **Pahl, G., and Beitz, W.**, (1984) "*Engineering Design*". Design Council. London.

[2] CLIPS Version 4.3, NASA Johnson Space Center, Houston, TX, 1989.

[3] Integrated Computer Aided Manufacturing (ICAM) Architecture, Part II, Vol. IV - Functional Modeling Manual (IDEF 0). Softech, Inc, Waltham, MA, 1981.

[4] **Rangan, R., M., Fulton, R., E., & Woolsey, T.,** (1989). EMTRIS Controlling Design and Manufacturing Information in a Discrete Part Manufacturing Environment. *ASME Computers in Engineering Conference,* Anaheim, CA. 1989.

[5] **Teorey, T., J., Yang, D., Fry, J., P.** , "A Logical Methodology for Relational Database Using the Extended Entity-Relationship" Model, *Computing Survey,* Vol. 18, No. 2, June, 1986.

[6] **Sriram, D., et. al.** (1989). "Knowledg-Based System Applications in Engineering Design: Research at MIT". *AI Magazine,* fall, 1989.

[7] **Rinderle, J., R.,** et. al.(1988). "Form- Function Characteristics of Electro-Mechanical Designs". Design Theory '88. *Proc. of the 1988 NSF Grantee Workshop on Design Theory and Methodology.*

[8] **AUTOCAD** Release 10, Reference Manual. Autodesk, 1989.

[9] **ORACLE DBMS** Reference Manual. ORACLE Corporation, California, 1987.

[10]. **FTP TCP/IP** Reference Manual, FTP Software, 1988.

[11]. **Rangan, R., M., Fulton, R., E., & Woolsey, T.,** (1990). *An Information based CAD Design Environment to Support Multidisciplinary Design.* ASME Computers in Engineering Conference , Boston, MA. 1990.

[12] **Bralla, J., G.** (Ed.). (1987). *Handbook of Product Design for Manufacture.* McGraw-Hill Book Co.

## B4 Session:
## Enhancements to CLIPS – General I

# DECOMPOSING A COMPLEX DESIGN PROBLEM USING CLIPS

James L. Rogers
NASA Langley Research Center
Hampton, VA, USA

## INTRODUCTION

Many engineering systems are large and multidisciplinary. Before the design of such complex systems can begin, much time and money are invested in determining the possible couplings among the participating subsystems and their parts. For designs based on existing concepts, like commercial aircraft design, the subsystems and their couplings are usually well-established. However, for designs based on novel concepts, like large space platforms, the determination of the subsystems, couplings, and participating disciplines is an important task. Moreover, this task must be repeated as new information becomes available or as the design specifications change. Determining the subsystems is not an easy, straightforward process and often important couplings are overlooked. The design manager must know how to divide the design work among the design teams so that changes in one subsystem will have predictable effects on other subsystems. The resulting subsystems must be ordered into a hierarchical structure before the planning documents and milestones of the design project are set. The success of a design project often depends on the wise choice of design variables, constraints, objective functions, and the partitioning of these among the design teams. Very few tools are available to aid the design manager in determining the hierarchical structure of a design problem and assist in making these decisions.

Recently Sobieski (ref. 1) showed the value of multilevel optimization as an approach to solving complex design problems. But to use this approach, a novel design problem must be decomposed to identify its hierarchical structure. Although much work has been done in applying AI tools and techniques to problems in different engineering disciplines (refs. 2,3), only recently has the application of AI tools begun to spread to the decomposition of complex design problems (ref. 4). Steward (ref. 5) developed a project management tool to organize and display the couplings among tasks in an NxN matrix format using matrix manipulations. A new tool, DeMAID (Design Managers Aid for Intelligent Decomposition, ref. 6) has been developed to implement a decomposition scheme suitable for multilevel optimization. It displays the data in an NxN matrix format and replaces the matrix manipulations with a CLIPS knowledge-based system (ref. 7) to provide more flexibility.

This paper discusses a proposed model of the design process and describes the functions DeMAID uses to decompose a novel, complex design problem into a multilevel structure. The knowledge-based aspects of DeMAID, as well as a sample problem showing its application, are also presented.

## A PROPOSED MODEL OF THE DESIGN PROCESS

DeMAID incorporates only one model of the many possible models of the design process. This model parallels Steward's (ref. 5) model of a system which defines the *structure* of a system as the way in which some parts of a system affect other parts of a system. These effects differentiate a system from just a collection of parts. The *semantics* of the system describe how and why these effects occur. The structure and semantics together completely describe the system. To attain a desirable structure, the design manager needs more formal tools to gain understanding of both the structure and the semantics of the system.

The design process is divided into tasks which are called *modules* in DeMAID. Modules (1) take some input data, (2) perform one or more functions, and (3) generate some output data. Modules are linked when the output from one module is the input to another module. Feedback links imply that information is required before it is available which, in turn, implies that guesses must be made to initiate the process and iterations are necessary. Typically, a desirable structure has a limited number of feedback links because each feedback link increases the cost of the solution. One method of reducing feedback links is multilevel decomposition where the modules and their couplings are ordered in such a way that a number of smaller uncoupled optimization problems can be identified.

DeMAID partitions the modules of a system into circuits which represent subsystems where each module is simultaneously dependent on all of the other modules within the same circuit. Feedback links are contained within the circuits indicating that an iteration is required. Circuits are connected to each other only by feedforward links. This indicates that there is no iteration among circuits and they can be ordered in a multilevel format. Thus a complex design process can be decomposed into a hierarchical set of tasks.

## FUNCTIONS OF THE DEMAID

DeMAID performs several useful functions to aid the design manager in attaining a desirable structure. These functions are (1) planning, (2) scheduling, (3) displaying the modules and their couplings in an NxN matrix format, (4) displaying the subsystems in a multilevel format, and (5) displaying the dependency matrix. Each of these functions is contained in a subroutine of the main program (figure 1). The planning function is always done first followed by the scheduling function. Calling the other functions depends upon the needs of the user. After each function is completed a file is written containing the current list of modules. This allows the user to restart the process without having to go back to the start each time.

The functions of DeMAID are discussed in the remainder of this section using a generic design problem as a sample problem. The problem has 45 modules. These modules perform one of the following tasks: (1) set the value of one or more design variables, (2) evaluate one or more constraint functions, (3) calculate intermediate results and behavior variables, and (4) evaluate the objective function. The problem is defined in terms of relationships among these four design tasks. The dependency of the objective and constraint functions on the design and behavior variables can be defined explicitly by mathematical equations. The same is true for defining the dependency of the behavior variables on the design variables. However, the question of whether new values of the design variables can be set without knowing the outcome of the function evaluations depends on the design manager's view of the problem, therefore engineering judgement is required when determining these dependencies. The main requirement is that a design variable can only depend on a function evaluation if that function is dependent on the design variable.

## Planning

The term *planning* within the context of DeMAID means determining which modules contribute to the solution of the problem. The user begins with a list of modules. This list should contain all modules that might possibly be used in the problem. The first step in the planning function is to determine whether or not a module contributes to the problem by checking the output of each module against the input requirements of the other modules. If the output of the module is contained in the input list of at least one other module then that module contributes to the solution of the problem. If a module is found not to be a contributor to the solution of the problem, then it is removed from the list of modules, but saved for possible use later.

In the second step, the planning function examines the input lists of all the modules to determine if all input requirements are satisfied by the output of other modules. Some modules have no input requirements. These modules are used for initialization purposes and represent external inputs. If an input requirement to a module is not satisfied, then the user must interactively add a new module to the list or remove the input requirement. If a new module is added, its input requirements are also checked. If one or more of its input requirements are not met, then the modules removed from the list earlier are checked first to determine if they satisfy the requirement, if not, then another module must be added. This step continues until all input requirements are satisfied.

## Scheduling

The scheduling function is the heart of DeMAID. Within the context of DeMAID, *scheduling* means the ordering of the modules into a meaningful solution sequence while limiting the number of feedback links among the modules. The scheduling function reorders the modules based on their couplings. If the modules and their couplings are placed into the matrix without any regard to their ordering, then very little information regarding the desirable structure of the system is available to the design manager. For example, the couplings of the initial data for the sample problem are very disorganized and contain a substantial number of feedback links (figure 2). Limiting the feedback links among the modules is accomplished by examining the couplings and grouping the modules into circuits. DeMAID also orders the modules within the circuits and orders the circuits within the design process. While Steward (ref. 5) implements the grouping into circuits with matrix manipulations, DeMAID follows the same steps but replaces the matrix manipulations for grouping by applying rules contained in a CLIPS knowledge base.

One of the advantages of using a knowledge-based tool over matrix manipulations is the ease with which new rules can be added. This gives the knowledge-based tool more flexibility. Additional rules that were not in Steward's (ref. 5) procedure were developed in conjunction with the design problem of Padula et al (ref. 8) and have been added to control the ordering of the modules within circuits and the ordering of circuits within the design process. The ordering is done based on the weight assigned to the modules. This step reorders the modules within a circuit by moving the modules with the highest weight to the beginning of the circuit. The modules with ever decreasing weights are moved to be below but near the top priority modules to which they are coupled. Using this method, tasks can begin as soon as possible but the modules with the highest weights are given priority.

## The NxN Matrix Display

Once the scheduling function is completed, the design manager can examine the NxN matrix display (see figure 3) and manipulate the modules and couplings to meet the requirements and semantics of the problem. DeMAID's display of the modules and their couplings is slightly different from that of Steward (ref. 5). The modules of the problem are placed on the diagonal of the matrix. The couplings are lines connected horizontally to a box to indicate an output from that module and vertically to indicate an input. A circle at the juncture of the horizontal and vertical lines indicates a coupling between two modules. A circle below the diagonal indicates a feedback link.

### Multilevel Organization

The circuits and their couplings can also be displayed in an NxN matrix format by DeMAID (figure 4). By examining the circuits, it is apparent that there are no feedback links among the circuits, therefore there is no iteration among the circuits. The only iterations are contained within the circuits. Thus, once the circuits have been found during the scheduling function, it is simple to achieve a multilevel organization of the problem. A list of circuits is input to the knowledge base to determine the multilevel hierarchy. As circuits with satisfied input requirements are found, they are placed on a level. A circuit is placed on the level below the lowest level containing a circuit which generates input for the circuit being placed (figure 5).

### Dependency Matrix

Another function of DeMAID is to build the dependency matrix of the problem. The usefulness of this matrix is described by Barthelemy (ref. 9). It is an ordered table that identifies the functional dependence between constraints and independent design variables. Behavior variables can be evaluated using design variables, therefore each behavior variable can be replaced by a list of independent design variables. Each constraint is examined to determine its dependency on design and behavior variables. Whenever a constraint depends on a behavior variable, the dependency of that behavior variable on the independent design variables is substituted. This produces a rectangular matrix with constraints listed along the rows and the independent variables along the columns (figure 6). An X marks the dependency. Building the dependency matrix after the planning and scheduling functions reveals dependency patterns that may prove advantageous when developing multilevel optimization algorithms.

## THE CLIPS KNOWLEDGE-BASED SYSTEM

CLIPS (C Language Production System, ref. 7) is a knowledge-based system that was developed at NASA Johnson Space Center. It is written in C, performs forward chaining based on the Rete pattern matching algorithm, and has a FORTRAN interface. There are three main parts to this knowledge-based system, the facts, the rules, and the inference engine.

## Facts

*Facts* are the basic form of data in the knowledge base and are contained in a facts-list. A fact is composed of one or more fields with each field separated by a space. A field can contain a number, a word, or a string. In DeMAID, facts are asserted into the facts-list by an assert command in the calling program before the inference engine is executed, and during execution as the action caused by executing a rule.

The primary function of DeMAID is to manipulate a list of the modules which contribute to the solution of the problem. A fact about a module has the following format:

```
(module ?number ?name ?weight ?time ?output ?status $?input-list)
```

The number field is for "moving" or reordering modules during the inferencing process.
The name field is a unique identifier for each module.
The weight field is a number defining the element of the design process: 4 for an objective function, 3 for a design variable, 2 for a behavior variable, and 1 for a constraint. The ordering of modules within a circuit is based on the weight assigned to the modules. The modules with the highest weights are moved to the beginning of the circuit, then the modules with ever-decreasing weights are moved to be below but near the top priority modules to which they are linked. Using this method, tasks can begin as soon as possible, but the modules with the highest weights are given priority.
The time field contains the amount of time required for a module to complete processing. This information is valuable if the designer desires to some of the processing in parallel.
The output field contains a name of the data that is output from that module. The output may represent a single value or multiple values.
The status field maintains the status of the module to prevent it from being applied more than once in a rule.
The input-list field is a list of the modules required to be available before the given module can be processed.

## Rules

The knowledge base also contains *rules*. A rule states that specific actions are to be taken if certain conditions are met. An action may be to return data to the calling program through the FORTRAN interface or assert a new fact into the facts-list. A rule executes based on the existence or non-existence of facts in the facts-list.

A typical rule in DeMAID looks like:

```
(defrule links
          (module ?no1 ?name1 ?wt1 ?tm1 ?out1 ?stat1 $?inlist1)
          (module ?no2.?name2 ?wt2 ?tm2 ?out2 ?stat2 $?inlist2)
          (test (neq ?name1 ?name2))
          (test (member ?out1 $?inlist2)) =>
                   (KBANS1 LINK null ?no1 ?no2))
```

This is interpreted to read as follows: If there are two different modules where the output of one is the input to the other, then a link exists between those two modules. The action, based upon the conditions being satisfied, is to return to the calling program via the KBANS1 parameter the fact that a link exists between modules ?no1 and ?no2. The KBANS1 parameter is a pointer to subroutine KBANS1 in the calling program. The LINK parameter serves as a pointer for storing the numbers in the KBANS1 subroutine. The null parameter is a place holder and not needed in this rule. The ?no1 and ?no2 parameters are the two link numbers to be passed to the calling program. If, for example, module 7 has the output field of DV07, and module 13 has the input-list field of DV01 DV07 DV20, then the rule would execute and the module numbers 7 and 13 would be returned to the KBANS1 subroutine to indicate a link between the two modules.

Currently there are 156 rules divided among seven files. The rule files correspond to the major subroutines in DeMAID. The division of the rules into seven files was done primarily for efficiency. For example rules pertaining to the planning function need not be considered during scheduling and vice-versa. After each subroutine is called, the rule file is loaded, the facts are asserted, and the CLIPS inference engine is executed.

## Inference Engine

The *inference engine* applies the knowledge (rules) to the data (facts) by pattern matching the facts in the facts-list against the conditions of the rule. The basic execution cycle begins by examining the knowledge base to determine if the conditions of any rules have been met. All rules with currently met conditions are placed on the *agenda* which is essentially a push down stack. Once the agenda is complete, the top rule is selected and its actions are executed. As a result of the action(s) of the rule, new rules may be placed on the agenda and rules already on the agenda may be removed. This cycle repeats until all rules that can execute have done so. The calling program passes control to CLIPS for execution of the inference engine and CLIPS returns control back to the calling program after all the rules have been executed.

As mentioned above, passing data from the execution of a rule to the calling program uses subroutine KBANS1. This subroutine has four parameters, two alphanumeric and two floating point. The first alphanumeric parameter serves as a pointer in KBANS1 to indicate in what variables the other parameters are to be stored. The other alphanumeric parameter is used to store data such as the name of the module. The two floating point parameters are used to pass data like module numbers, times, or weights. If one or two of the parameters are not used in a rule, a dummy value is passes, such as the null in the above example.

## AN APPLICATION OF DeMAID

DeMAID was applied to the design of the Control of Flexible Structures (COFS) mast problem by Padula et al (ref. 8). The COFS mast was a truss structure, attached to the space shuttle, and used to study techniques for system identification and active vibration control. The system contained about 50 modules for decomposition. For this system, a module represented calculating system behavior variables such as the vibration frequencies, selecting a value for one or more design variables, or evaluating a constraint function. A diagram of the decomposed system is shown in figure 7. (Note: DeMAID requires about 10 minutes to complete the planning and scheduling functions for a problem of this size. The amount of time required to decompose a problem increases dramatically with the number of modules.) After the system was decomposed, it was divided among design teams such as *structures* and *controls* as indicated in the figure. The reference explains the changes made to the model by the design manager to arrive at this particular decomposition.

## SUMMARY

DeMAID is a newly developed CLIPS-based tool for decomposing complex design problems into a suitable multilevel structure based on the multilevel optimization approach. DeMAID requires an investment of time to generate and refine the input for each design module. This investment may not be justified for a small, well-understood problem, but should save a significant amount of money and time in organizing a new design problem where the ordering of the modules is still unknown. The decomposition of a complex design system into subsystems requires an interaction with the judgement of the design manager. DeMAID can aid the design manager in making decomposition decisions early in the design cycle.

DeMAID aids the design manager by reordering and grouping the modules based on the couplings among the modules. The modules are grouped into circuits (the subsystems) and displayed in an NxN matrix format. The feedback links, which indicate an iterative process, are limited and restricted to be within a circuit. Since there are no feedback links among the circuits, the circuits can be displayed in a multilevel format. Thus, a large amount of information is reduced to one or two displays which can easily be stored, retrieved, and modified. The design manager and leaders of the design teams are given a visual display of the design problem and the intricate couplings among the different modules so that they can determine how a change in one subsystem will affect other subsystems. It also helps reduce the possibility of overlooking important couplings.

In addition to decomposing the system into subsystems, DeMAID examines the dependencies of the problem and creates a dependency matrix. This matrix shows the relationship among the independent design variables and the dependent objective and constraint functions.

Since DeMAID is based on AI knowledge base techniques, it has proven to be very flexible in adding new capabilities. Given its current capabilities, DeMAID can provide the design manager a great deal of insight when decomposing novel, complex design systems into more manageable subsystems; thereby saving considerable time and money in the total design process.

## REFERENCES

1. Sobieszczanski-Sobieski, J.: "A Linear Decomposition Method for Large Optimization Problems - Blueprint for Development." NASA TM 83248, February 1982.

2. Sriram, D.: A Bibliography on Knowledge-based Expert Systems in Engineering. SIGART Newsletter, No. 89, July 1984, pp. 131-136.

3. Sriram, D.; and Joobbani, R.: Special Issues, AI in Engineering. SIGART Newsletter, No. 92, April 1985, pp. 38-144.

4. Rogan, J. E.: and Kolb, M. A.: Application of Decomposition Techniques to the Preliminary Design of a Transport Aircraft, NASA Contractor Report 178239, February 1987.

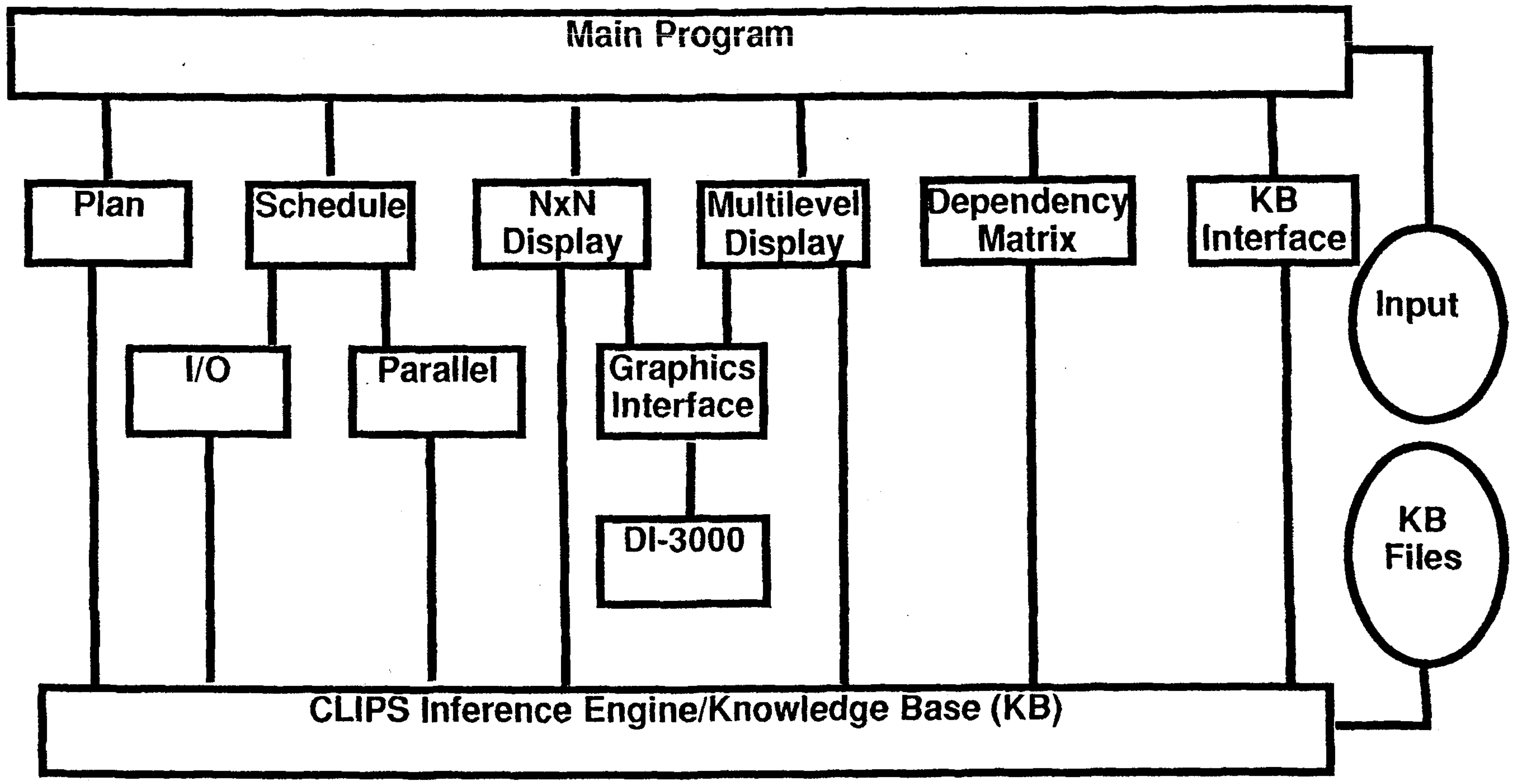
5. Steward, D. V.: Systems Analysis and Management. Petrocelli Books, Inc., New York, NY, 1981.

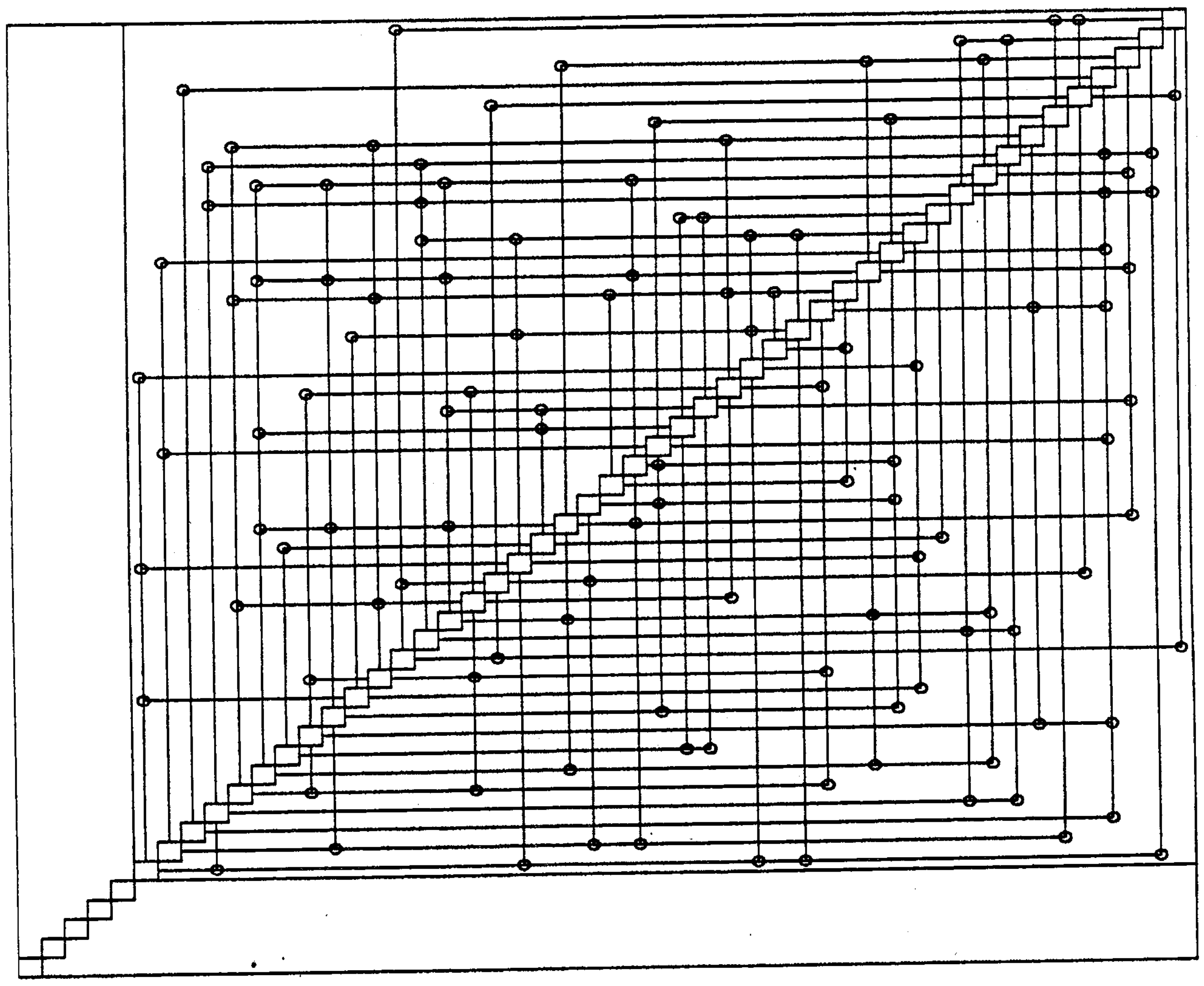
6. Rogers, J. L.: "A Knowledge-based Tool for Multilevel Decomposition of a Complex Design Problem." NASA TP 2903, May 1989.

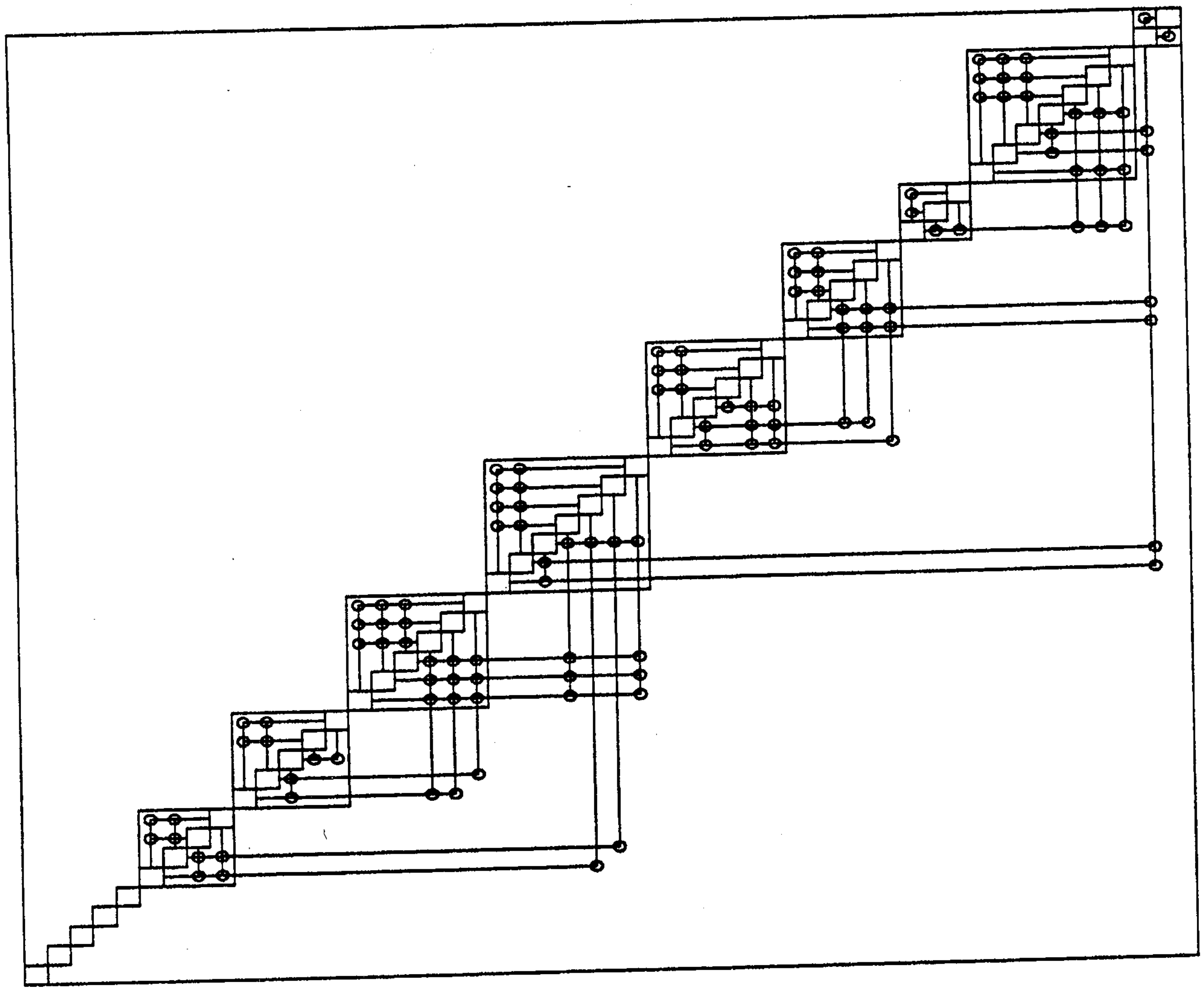
7. Riley, G.; Culbert, C.; and Savely, R. T.: "CLIPS : An Expert System Tool for Delivery and Training." Proceeding of the Third Conference on AI for Space Applications, November 1987.

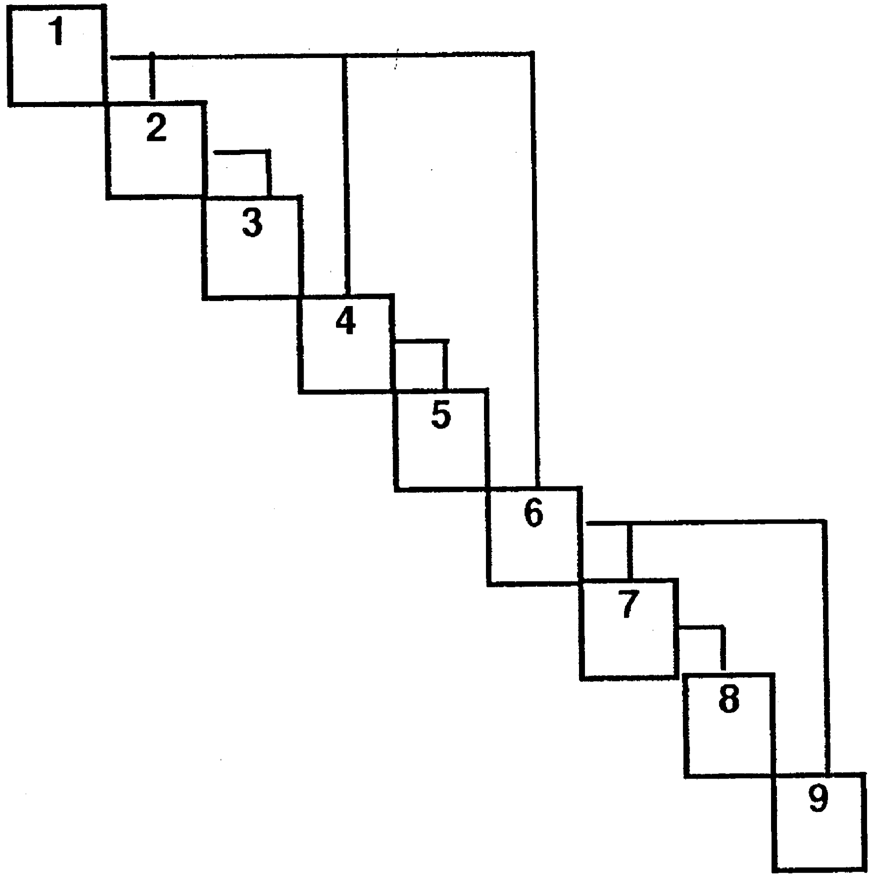
8. Padula, S. L.; Sandridge, C.; Haftka, R. T.; and Walsh, J. L.: "Demonstration of Decomposition and Optimization in the Design of Experimental Space Systems." Preprints of the Second NASA/Air Force Symposium on Recent Advances in Multidisciplinary Analysis and Optimization. September 28-30, 1988, Hampton, VA, pp. 7-27.

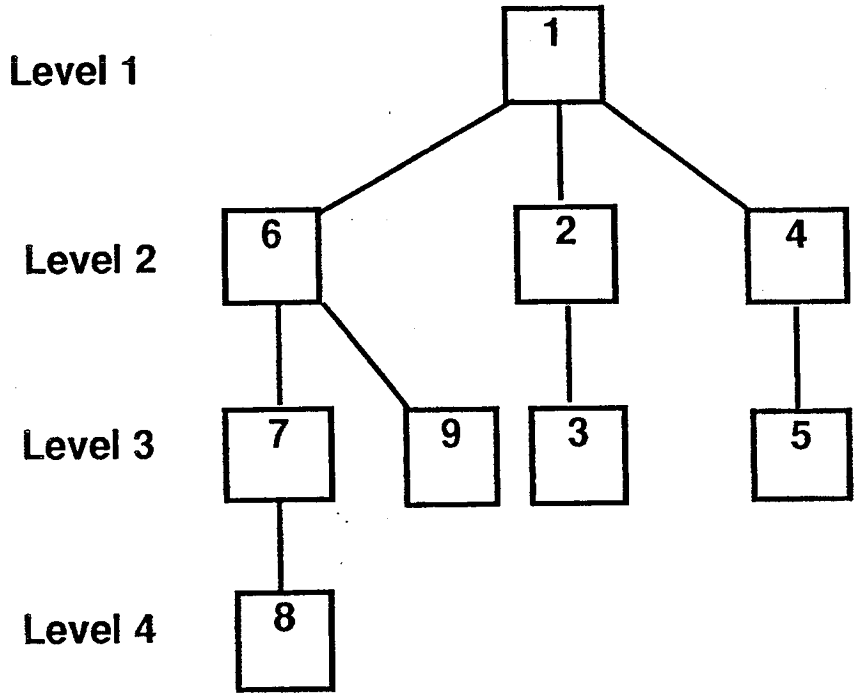
9. Barthelemy, J.-F. M.: "Engineering Applications of Heuristic Multilevel Optimization Methods." NASA TM 101504, October 1988.


Main Program
Plan
Schedule
NxN Display
Multilevel Display
Dependency Matrix
KB Interface
I/O
Parallel
Graphics Interface
DI-3000
Input
KB Files
CLIPS Inference Engine/Knowledge Base (KB)

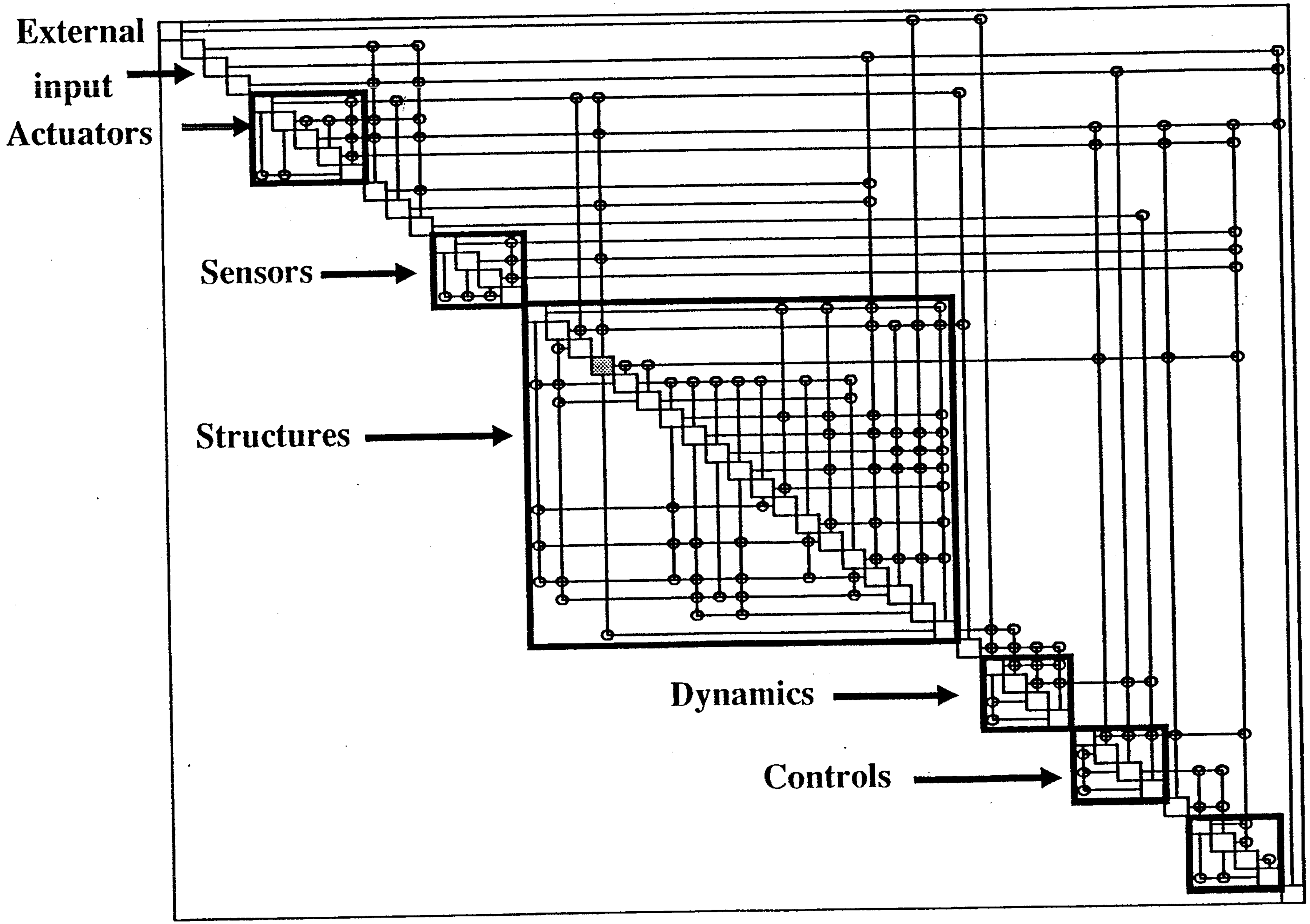





1
2
3
4
5
6
7
8
9


Level 1
Level 2
Level 3
Level 4
1
6
2
4
7
9
3
5
8

| Module | 2 | 3 | 4 | 5 | 10 | 11 | 13 | 14 | 15 | 18 | 19 | 20 | 24 | 25 | 26 | 27 | 31 | 32 | 33 | 37 | 38 | 42 | 43 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 7 | X | X | X | X | | | | | | | | | | | | | | | | | | | |
| 8 | X | X | X | X | | | | | | | | | | | | | | | | | | | |
| 9 | | X | X | X | | | | | | | | | | | | | | | | | | | |
| 12 | | X | X | X | X | X | | | | | | | | | | | | | | | | | |
| 16 | X | | | | | | X | X | X | | | | | | | | | | | | | | |
| 17 | X | | | | | | X | X | X | | | | | | | | | | | | | | |
| 22 | | | | | | | | X | X | X | X | X | | | | | | | | | | | |
| 23 | | | | | | | X | | | X | X | X | | | | | | | | | | | |
| 29 | X | | | | | | | | | | | | X | X | X | X | | | | | | | |
| 30 | X | | | | | | | | | | | | X | X | X | X | | | | | | | |
| 34 | | | | | | | | | | | | | X | | | X | X | X | X | | | | |
| 35 | | | | | | | | | | | | | X | | | X | X | X | X | | | | |
| 36 | | | | | | | | | | | | | X | | | X | X | X | X | | | | |
| 40 | | | | | | | | | | | | | | | | | X | | | X | X | | |
| 41 | | | | | | | | | | | | | | | | | | X | X | X | X | | |
| 44 | | | | | | | | | | | | | | X | | | | | | | | X | X |
| 45 | | | | | | | | | | | | | | | X | | | | | | | X | X |


External input
Actuators
Sensors
Structures
Dynamics
Controls

# FAULT-TOLERANT, EMBEDDED CLIPS APPLICATIONS

Jaye Hicks, MCS
John Matthews, MS

Department of Computer Science
Texas A&M University

## OUTLINE

## ABSTRACT

The enhancements to CLIPS4.3 presented in this paper provide an embedded CLIPS application with the ability to continue operation with minimal to no loss of information in the event of a hardware or a software failure. Given an arbitrary failure, the CLIPS application's environment (fact-list, agenda, and pattern matching network) will be reconstructed to the point at which the failure was experienced. The environment reconstruction is based on state files to which the application periodically checks environment information (fact-list and agenda). The routine for checkpointing the state of the application is as efficient as possible so that the overhead introduced to normal execution of the application is minimal. The only assumptions made by the CLIPS application are that it is running under an operating system that guarantees it access to uncorrupt state files and that the application will be automatically restarted should it terminate abnormally.

## 1. INTRODUCTION

Loss of information due to hardware or software failure can be costly for any application. With a production system, such as a CLIPS application, inference chains can be quite long and computationally expensive. Simply restarting the application at "the beginning", should some failure cause the CLIPS application to halt execution before any useful conclusion is reached, is often very undesirable. However, CLIPS does not offer any means to recover from such a failure without the potentially expensive loss of information. Therefore, for a CLIPS application to be made fault-tolerant, the operating system would need to redundantly execute the application, which could very well be an expensive and unacceptable solution.

There are many ways to organize a fault-tolerant system. In differentiating possible organizations, two aspects of a fault-tolerant system can be considered.[1][2] The first aspect is the degree of processor coupling (loose vs. tight); the second aspect is whether or not identical processes are to be executed. The Loosely Coupled Processors and Independent Processes fault-tolerant organization is one in which software provides almost all of the fault recovery and there is no redundant execution of applications. The enhancements to CLIPS discussed in this paper utilize this method to enable the construction of fault-tolerant CLIPS applications. The other fault-tolerant schemes rely mainly on hardware for fault recovery (e.g. voting scheme) and will not be discussed, although fault-tolerant expert systems have been developed using these methods.[3]

With the fault-tolerant scheme of Loosely Coupled Processors and Independent Processes, a system that experiences a fault will be unavailable for a finite, relatively short period of time while it attempts to recover from the fault. This is a plausible method to insure fault-tolerance of flexible, real-time systems (those without hard deadlines) without requiring expensive fault-tolerant hardware. For example, a fault-tolerant embedded CLIPS application that will both diagnose and control a physical system, based on periodic sensor readings, might recover from a failure in the otherwise idle time between the arrivals of sensor data. However, the operating system must automatically restart the application should it terminate abnormally, and guarantee the application constant access to uncorrupt state files. Though this might sound like a great deal to ask for, it is considerably less expensive, in a hardware sense, than the redundant execution of applications on multiple nodes. Furthermore, by minimizing the space requirements and overhead imposed on the normal execution of the CLIPS application, in order to insure fault-tolerance, this fault-tolerant scheme becomes even more appealing.

In order to reconstruct the CLIPS environment after hardware or software failure, it is necessary to have access to previous CLIPS environment state information and to a transaction log. During execution of the application, the CLIPS environment should be saved in state files at checkpoints. The transaction log will contain a record of actions taken since the last checkpoint that affect the CLIPS environment. Two essential additions to CLIPS are: (1) a command which saves the fact-list and the agenda to state files, and (2) an I/O router which intercepts information that the *watch all* diagnostic mode of CLIPS routes to the logical name *wtrace* and enters the diagnostic information in the transaction log. The major addition to CLIPS enabling fault-tolerant CLIPS applications to be created is the restoration algorithm whose execution can be explained with the following two scenarios.

The most straightforward case of environment restoration follows a failure after a checkpoint before anything was entered in the transaction log. This case involves restoring the rules and the fact-list, in addition to reconstructing the agenda. After the rules and fact-list have been loaded into CLIPS, the resulting agenda will be a superset of the agenda saved at the last checkpoint. A possible discrepancy between agendas can occur because CLIPS enforces refraction in determining the activations to place on the agenda. The reconstruction routine will then need to purge any activations from the current agenda that are not on the agenda saved at the last checkpoint.

The most interesting case of environment restoration involves a failure after a checkpoint, where entries have been made to the transaction log. Restoration is identical to the previous case except that the final step would be to process the transaction log. By affecting the CLIPS environment according to entries indicated in the transaction log, the loss of environmental changes since the last checkpoint can be avoided. In this way, minimal loss of information can be insured for low cost since making entries to a transaction log is extremely inexpensive when compared to saving the state of the CLIPS environment to state files.

Using the previously mentioned checkpointing command, I/O router, and restoration algorithm, it is possible to design a fault-tolerant system using production system technology without redundant execution of applications. For example, CLIPS applications can be designed to control/diagnose physical systems as long as a small amount of down-time due to failure followed by recovery, is acceptable. Such controllers/ diagnosers would undoubtedly rely upon historical data to control, diagnose, and anticipate the behavior of systems; a hardware or a software failure should not cause the loss of crucial historical information. Using the restoration routine in the event of a failure these controllers and diagnosers can operate in a continuous fashion with minimal impact from failures.

After a brief overview of the characteristics of a CLIPS application, a more detailed look is taken at the expectations of a fault-tolerant CLIPS application on the operating system it is running under. Next, the process of saving the CLIPS environment to state files will be discussed in more detail followed by an explanation of the actual restoration algorithm. The final section of this paper proposes future expansions and improvements on the work presented in this paper.

## 2. OVERVIEW OF EMBEDDED CLIPS APPLICATION

### 2.1 Embedded vs. Interactive CLIPS Applications

The enhancements to CLIPS described in this paper are geared toward embedded CLIPS applications. Interactive CLIPS applications require continual human interaction with the program, and though the same state saving and restoration techniques can be applied to them as proposed in this paper, this case is not as challenging as the embedded one. This is due to the autonomous nature (e.g. restoration without human intervention) of embedded CLIPS applications along with their possibly lengthy execution times. Hence, any CLIPS application referred to in this paper, except for the applications in the table in section 8, should be assumed to be embedded.

### 2.2 Indivisible CLIPS Operations

During the execution of any CLIPS applications, certain operations must be executed atomically. Therefore, if the CLIPS application is intended to run in a networked environment and will use interrupt handlers to receive messages, instead of busy waiting, certain situations must be avoided. For example, if the CLIPS application was accessing its fact-list and an interrupt handler was invoked that accessed the fact-list, the results could easily be erroneous. This is due to the nonreentrant nature of certain CLIPS operations. Namely, asserting a fact, retracting a fact, and executing activations on the agenda must be done in an indivisible fashion. This is not to say that these operations cannot be interrupted, it is that access to the fact-list and the agenda must be completed before another such access is attempted. Since the save-state routine and the restoration routine discussed in this paper access the fact-list and the agenda, they too must be executed atomically.

### 2.3 CLIPS Internal Environment

To better understand the internal CLIPS environment, it is useful to first define the execution cycle.

"The basic execution cycle is as follows:

a) The knowledge base is examined to see if the conditions of any rules have been met.

b) All rules whose conditions currently are met are activated and placed on the agenda. The agenda is essentially a stack. ...

c) The top rule of the agenda is selected, and its right-hand side (RHS) actions are executed."[4]

The internal environment for a CLIPS application consists of the fact-list, the agenda, and the Rete algorithm-based pattern matching network. The rule set(s) and fact template definitions are assumed to be static and will be restored in the event of failure by simply loading a file(s). Facts and rules are well documented concepts of production systems, however the CLIPS pattern matching network will be elaborated upon.

"Prior to initiating execution, each rule is loaded into the system and a network of all patterns that appear on the LHS of any rule is constructed. As facts are asserted into the fact-list, they are filtered through the pattern network. If the form of the pattern (number of fields and literal fields) matches any of the patterns in the network, the rule(s) with that pattern is partially instantiated. When facts exist that match all patterns on the LHS of the rule, variable bindings (if any) are considered... If the field values for all patterns are consistent with the constraints applied to the variables, the rule(s) is activated and placed on the agenda."[4]

The pattern matching network is reinstantiated automatically by CLIPS as a result of restoring the rules, fact-list ,and agenda in turn, and possibly processing the transaction log. Therefore, the only things that need to be saved at a checkpoint are the fact-list and the agenda. The restoration routine is able to reconstruct the CLIPS environment with these two state files and a transaction log if this is applicable. Even though, this will require some processing, it is a much more economical and efficient solution than to save the entire context of the CLIPS application at checkpoints.

## 3. REQUIREMENTS ON OPERATING SYSTEM

To provide resiliency to the embedded CLIPS application, the application will need to be run under a special operating system component. This operating system component, which will hereafter be referred to as the Fault-Tolerant Monitor (FTM), will guarantee that the application will continually be executing on some processor and that it will always have access to accurate state files (see fig. 1). This means that in the event of a failure that halts the execution of the CLIPS application, the FTM will restart the application. Any state files should be designated as such to the FTM by the application during initialization. One method to insure the availability of state files would be for the FTM to keep a duplicate copy of each state file, and better yet to keep this copy on a different node as the master copy if multiple nodes exist. Correctness of the state files could be insured by disk mirroring, while continual execution of the applications could be insured by making each application a child process of the FTM, in which case the FTM would be notified by the kernel if any of its child processes terminated abnormally.

## 4. SAVING THE STATE

### 4.1 Save-state Routine

Saving the state of a CLIPS application involves saving the fact-list and the agenda to state files and resetting the transaction log to begin logging information that affects the CLIPS environment (see fig. 2). This information saved to the state files is the minimum amount required by the restoration routine to correctly reconstruct the environment of a CLIPS application, and is considerably smaller in size than the entire run-time context of that application. Two state files, one for the fact-list and one for the agenda, will be used to save the state. The fact field names will not be written to the state file when using templated facts in order to conserve space. It is assumed that the rule base(s) will not change during the execution of the application so there is no need to save the rule base at checkpoints.

To avoid single point failure, two separate sets of agenda and fact-list state files are used by the save-state function. This is necessary because a failure during the save-state function could result in the loss of all the old state information in that state file set and incomplete saving of the new state information. The resulting partial state files would be useless during an attempt to reconstruct the CLIPS environment to its state at the point of failure. Therefore, the first priority of the save-state function is to determine which state file set to use. This turns out to be a straightforward task as the last entry of each complete state file will be a time stamp. The time stamp is simply the character sequence "**TS** dddddddddd", where the series of d's will be the number of seconds that have elapsed since January 1, 1970. The larger the number represented by the d's, the newer the state file. Since this time stamp will be placed in the state file only after the entire fact-list or agenda has been written, the time stamp serves a dual purpose. Not only does the stamp give the relative age of the state file, its presence indicates that the entire structure has been written to the state file, which is useful since the size of the agenda and the fact-list will vary. The time stamps are in terms of seconds, this being reasonable since the average embedded CLIPS application would not need to save the state more than once a second. However, should there ever be a need to save the state more than once a second the time stamp mechanism in the state saving routine can be enhanced to mark a file with a stamp that is down to the microsecond level of granularity.

The state file set that will be overwritten in a save-state call will be the oldest if both sets are intact. If one of the state file sets is not intact, it will be chosen as the set to write to since errors might exist in that file. If both of the state file sets are tainted, then a set will be chosen arbitrarily. This is done in order to keep the most recent and correct state file set around in case of failure. After the agenda and fact-list have been saved successfully, the transaction log will be reset and its first entry will be a time stamp corresponding to the time stamp of the state file set just used. This will mark the transaction log thus indicating to which state file set the transaction log corresponds. Since the restoration routine might use the older state file set (if the most recent set is corrupted), the time stamp in the transaction log will enable the restoration routine to determine whether or not it should process the transaction log.

## 4.2 Checkpointing Policies

The restoration algorithm does not assume any one particular checkpointing policy. Greater frequency of state saving enables faster restoration, but increases the amount of overhead for the application. Since an acceptable frequency for checkpointing will vary from CLIPS application to CLIPS application, it is left to the application designer to insert save-state calls wherever needed. A static interval for checkpointing might be rigid and not as efficient as a more flexible dynamic interval for checkpointing. An example dynamic interval checkpointing policy would be to save the state of the CLIPS environment whenever the transaction log reaches a certain size (say the typical size of both agenda and fact-list state file sets). Also, to provide more flexibility to the application designer, the two routines are available which can be invoked from the right hand side of a CLIPS rule, the interactive CLIPS command line, or from a C function that will save the fact-list or the agenda to the file specified by the argument.

## 4.3 Relative Size of Critical Files

In order to gauge the size requirements of the state file scheme, it is useful to look at the size of the executing CLIPS application that corresponds to the state files. The maximum run-time size of a CLIPS application will vary from application to application, and perhaps even from machine to machine for the same application. Therefore, no tight bound on the maximum amount of memory that an application will require during execution can be placed. However, by empirical observation, one can easily calculate the largest size that a particular CLIPS application reaches during its execution.

> "When new memory is needed by any CLIPS function, CLIPS first checks its own data buffers for a pointer to a free structure of the type requested. If one if found, the stored pointer is returned. Otherwise, a call is made to malloc for the proper amount of data and a new pointer is returned."[4]

The size of the CLIPS executable includes all the source code of CLIPS4.3 and the source code of the enhancements proposed in this paper (running on unix-based workstation). A global variable was kept to track the largest amount of memory used by the CLIPS application at any one time. By adding this number to the basic size of the CLIPS executable (450,560 bytes), the maximum amount of memory used by each application during its execution was calculated. The table in section 8 correlates the maximum size in bytes that CLIPS programs reach while executing to the size that the state files reach with a reasonable checkpointing scenario (state file memory requirement is only 1.5% that of working memory). This reasonable scenario is to run the application and checkpoint at about one-third of the way through and then again about two-thirds of the way through. This way, both state file sets and the transaction log are utilized and the state files reflect the entire execution of the application. The size requirement of the state file scheme is then the combined size of both the state file sets plus the transaction log. The CLIPS programs used in this comparison are the examples included in the CLIPS4.3 software package.

## 4.4 Complexity of Save-state

The agenda and fact-list are internal CLIPS structures kept in linked lists. The save-state routine traverses both linked lists once. If the larger of the two linked lists for the agenda and fact-list is of size n and the smaller is of size m, then the save-state routine is of time complexity O(n + m), which is O(n). The fact that each entry in each linked list must be examined in order to correctly and completely save the state of the CLIPS environment, coupled with the fact that file I/O is intrinsically linear proves that the save-state routine is optimal.

## 5. RESTORING THE STATE

### 5.1 Restoration Routine

The general scenario for restoring a CLIPS application is as follows. During the application's initialization, it will learn from the FTM whether it is executing for the first time or is being restarted as a result of some failure. In the case of a restart, it will invoke the restoration routine. When restoring the state of a CLIPS application, the first priority will be to restore the fact-list based on the fact state file from the most recent uncorrupted state file set. The agenda will be reconstructed later with the agenda state file from this set and the transaction log will only be processed if it has the same time stamp that both of the files in this set have. To restore the fact-list, both templated and free form facts are built from scratch, one field at a time. "This method for building facts is much more efficient than using the function *assert*. It should be used by embedded applications which assert numerous facts into the fact-list."[4] While each fact field from an entry of the fact state file is isolated in preparation for the fact assertion, it is tested to make sure it is a valid CLIPS type NUMBER, STRING, or WORD.

Since there is no limit on the length of time that facts can remain in the fact-list, it is entirely possible that a fact is in the fact-list, but any activations on the agenda it was responsible for have long since been purged or executed. Because CLIPS uses refraction for deciding what to place on the agenda, this particular fact, as long as it remains undisturbed in the fact-list, will never cause the same activation to be reintroduced onto the agenda.

> "CLIPS was programmed with a characteristic of a nerve cell called a neuron. After a neuron transmits a nerve impulse (fires), no amount of stimulation will make it fire for some time. This phenomenon is called refraction and is very important in expert systems. ... In the real world, the stimulus that caused the firing eventually would disappear. ... However, in the computer world, once a fact is entered in the fact-list, it stays there until it is explicitly removed... Without, refraction,

expert systems would be caught in trivial loops. That is, as soon as a rule fired, it would keep on firing on that same fact over and over again."[5]

Yet when the fact-list is restored with the fact state file, just such a fact will cause activation(s) on the agenda which are not part of the agenda state file. It is for this reason that the agenda resulting from the assertion of each fact indicated in the fact state file will be a superset of the desired agenda.

These extraneous agenda activations in the newly reconstructed agenda can be detected by checking each activation against the entries in the agenda state file. If an activation is on the agenda but does not have a counterpart in the agenda critical file, it is considered extraneous and should be deleted. In order to avoid the necessity of reading from the state file each time a current agenda activation is checked, the restoration routine will read the agenda state file once, making each entry a member of a linked list.

The last step of restoration is processing the transaction log. The first entry of the transaction log will be a time stamp that is identical to the most recent set of state files. Processing the transaction log should only occur if the state file set used to restore the state has a time stamp which matches the time stamp of the transaction log. The reason for this becomes clear when one considers that the purpose of the transaction log is to log all the actions taken that affect the CLIPS environment since the last checkpoint. Therefore, it only corresponds to the most recent set of state files, though restoration could have taken place with the oldest state file set. It is also important to realize that if the restoration routine processes the transaction log it will only be able to restore the CLIPS environment to the degree of the last complete and accurate entry of the transaction log.

The restoration routine presented in this paper is limited in restorative ability to the types of actions logged in the transaction log. It is assumed that the rule base(s) and fact template definitions will remain static and not change during execution of the application. *Undeffacts*, *undefrules*, *excise*, etc. will not be part of the application. In fact, the only things that will be logged in the transaction log will be fact assertions, fact retractions, agenda activations, agenda deactivations, and rule firings. Assertions and retractions are handled by asserting and retracting respectively. When processing a fact assertion entry in the transaction log, identical error checking and fact assertion methodologies as those used in the restoration of the fact-list are employed. To process a fact retraction entry in the transaction log, the fact number is isolated out of the entry and the *retract_fact* function is called with this number as the argument. Agenda activation and deactivation entries in the transaction log can essentially be ignored as agenda activations and deactivations will result automatically with the assertion and retraction of facts.

Processing rule firings is a bit more complicated than processing the other transaction log entries. Essentially, if a rule has fired then the agenda activation corresponding to the rule should be deleted from the newly constructed agenda. However, care must be taken to insure that all of the right hand side actions of the rule actually executed before the activation should be taken off the agenda. If the

CLIPS application ceased execution during the firing of a rule (the execution of the RHS actions), it is possible that not every action was executed before failure occurred, yet the firing is logged along with none, some, or all of the RHS actions in the transaction log. To remove the uncertainty posed by this particular case, an *exec* function was added to CLIPS that places a special character sequence, "**FE*", in the transaction log after all the RHS actions have taken place (see fig. 2). In this way, if the special character sequence is found, the rule activation can be deleted off the new agenda without fear of losing information. If the special character sequence is not found in the transaction log, processing of the transaction log will be stopped without removing the rule from the new agenda. The reason being that either the point of failure was this rule execution or the transaction log is corrupt at this point.

After restoration is complete the state is saved, thereby precluding the need to process a transaction log again should a failure be experienced before the next checkpoint.

## 5.2 Complexity of Restoration

With n entries in the fact state file, the complexity of restoring the fact-list is O(n), since file I/O is linear. The agenda state file is placed in a linked list which requires O(m) time for an agenda state file with m entries. In the worst case each activation on the newly constructed agenda will traverse the entire old agenda linked list before finding its counterpart. Given a current agenda with p activations this would require O(p * m). It will require O(q) time to process a transaction log of size q in the worst case. Therefore, the complexity of the restoration routine is O(n + m + p * m + q). In the expected case the number of entries in the fact-list will be considerably larger than the number of entries on the agenda ( |fact-list| >> |agenda|), and processing of the transaction log will generally occupy most of the restoration routine's time. Therefore, in the expected case given a sensible checkpointing policy is used (transaction log size is at most that of both critical file sets) restoration will be O(|transaction log|) when a transaction log is processed and O(|fact-list|) when a transaction log is not processed.

## 6. FUTURE EXPANSIONS

While the enhancements presented in this paper provide all the required tools necessary to build a fault-tolerant CLIPS application, further work is still possible. To increase speed, a hash table could be used instead of a linked list to house the old agenda in the routine that restores the agenda. For cases where the old agenda saved in the state file is quite large, the sequential traversal of the data structure could be a hindrance on the restoration routine. Another possible improvement would be to log and process more types of actions in the transaction log that affect the CLIPS environment, such as *undeffacts*, *undefrule*, and *excise*.

To increase the level of confidence for state files, a checksum number could be tacked on to the end of each entry made to these files. To achieve this, any communications protocol technique can be used. The most common scheme used is the cyclic redundancy check.[6] This method is one of the most powerful error-detecting codes. It would also be possible to take this improvement further and attempt to add error-correcting codes to allow recovery even when a file was corrupted. Being able to recover in this situation would take some of the burden off the FTM. However, it may very well be that the overhead introduced to achieve these improvements would be unacceptable.

## 7. CONCLUSION

It was imperative to design the state saving code so that it would be as efficient as possible because the state will continually be saved throughout the execution of the application. Though efficiency is desirable, it is not quite as critical for the restoration routine, as it will only be executed in the event of a failure. Failures should definitely not be as frequent as the number of times the save-state routine is called. The restoration routine is fairly efficient but not quite optimal.

The additions to CLIPS previously discussed employ software to provide fault-tolerance to a CLIPS application. The overhead required to save a CLIPS application's environmental information is minimal and the space it requires is about 1.5% that required by the entire context of the application when it is executing. For CLIPS applications without hard deadlines, which need fault-tolerance but cannot be run on expensive fault-tolerant machines, the fault-tolerance facilities presented in this paper are a viable and efficient option.

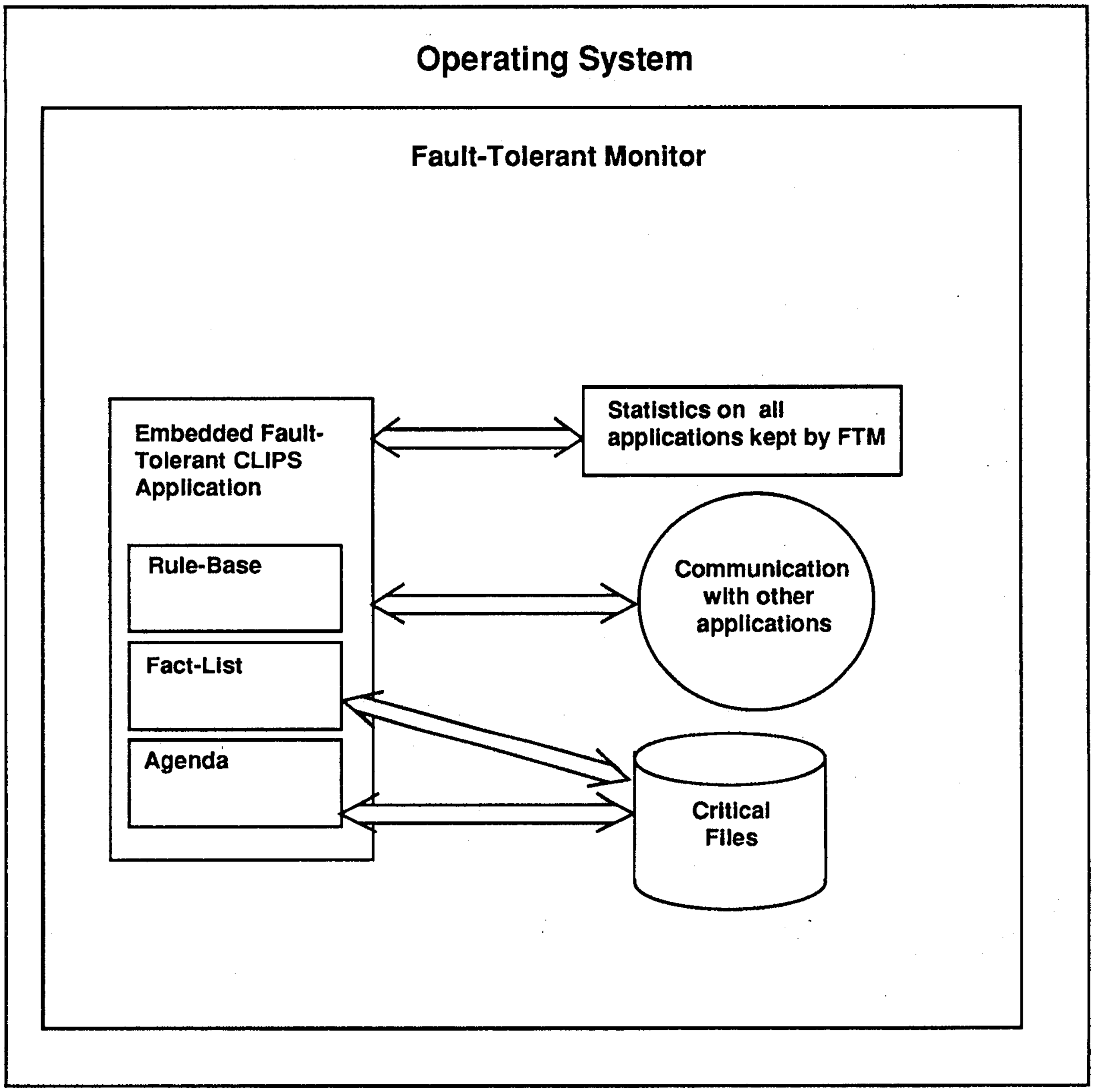
## 8. TABLES

| Application Name | Total Number of Rules Fired | Maximum Run-Time Memory Requirements (bytes) Application | Critical Files |
|---|---|---|---|
| auto.clp | 9 | 490,249 | 2,190 |
| dilemma.clp | 80 | 478,724 | 6,876 |
| mab.clp | 81 | 506,979 | 7,331 |
| ttt.clp | 151 | 500,646 | 6,524 |
| wine.clp | 44 | 526,840 | 4,670 |

NOTE: the CLIPS executable requires 4682 additional bytes to get to the CLIPS interactive prompt

## 9. FIGURES

Figure 1


Operating System
Fault-Tolerant Monitor
Embedded Fault-Tolerant CLIPS Application
Rule-Base
Fact-List
Agenda
Statistics on all applications kept by FTM
Communication with other applications
Critical Files

## Figure 2

*Fact-List State File (Set A)*

```
f-0     (initial-fact)
f-1     (square 1 1 " " corner)
f-2     (square 2 1 " " side)
f-3     (square 3 1 " " corner)
f-4     (square 1 2 " " side)
f-6     (square 3 2 " " side)
f-7     (square 1 3 " " corner)
f-8     (square 2 3 " " side)
f-9     (square 3 3 " " corner)
f-10    (computer is o)
f-11    (human is x)
f-15    (square 2 2 o center)
f-18    (evaluate board for human)
f-19    (potential-win 3 3 corner o)
f-20    (potential-win 2 3 side o)
f-21    (potential-win 1 3 corner o)
f-22    (potential-win 3 2 side o)
**TS** 639171738
```

*Agenda State File (Set A)*

```
0      evaluate-potential-win: f-18,f-15,f-6,f-4
0      evaluate-potential-win: f-18,f-15,f-7,f-3
0      evaluate-potential-win: f-18,f-15,f-8,f-2
0      evaluate-potential-win: f-18,f-15,f-9,f-1
-10    switch-to-move: f-18
**TS** 639171738
```

*Fact-List State File (Set B)*

```
f-0     (initial-fact)
f-2     (square 2 1 " " side)
f-3     (square 3 1 " " corner)
f-4     (square 1 2 " " side)
f-6     (square 3 2 " " side)
f-7     (square 1 3 " " corner)
f-8     (square 2 3 " " side)
f-9     (square 3 3 " " corner)
f-10    (computer is o)
f-11    (human is x)
f-15    (square 2 2 o center)
f-19    (potential-win 3 3 corner o)
f-20    (potential-win 2 3 side o)
f-21    (potential-win 1 3 corner o)
f-22    (potential-win 3 2 side o)
f-23    (potential-win 1 2 side o)
f-24    (potential-win 3 1 corner o)
f-25    (potential-win 2 1 side o)
f-26    (potential-win 1 1 corner o)
f-30    (square 1 1 x corner)
f-33    (cleanup for computer)
**TS** 639171741
```

*Agenda State File (Set B)*

```
0      cleanup-2: f-33,f-19
0      cleanup-2: f-33,f-20
0      cleanup-2: f-33,f-21
0      cleanup-2: f-33,f-22
0      cleanup-2: f-33,f-23
```

```
0      cleanup-2: f-33,f-24
0      cleanup-2: f-33,f-25
0      cleanup-2: f-33,f-26
-10    switch-to-evaluate: f-33
**TS** 639171741
```

*Transaction Log*

```
**TS** 639171741
FIRE    1 cleanup-2: f-33,f-19
<== f-19    (potential-win 3 3 corner o)
**FE*
FIRE    2 cleanup-2: f-33,f-20
<== f-20    (potential-win 2 3 side o)
**FE*
FIRE    3 cleanup-2: f-33,f-21
<== f-21    (potential-win 1 3 corner o)
**FE*
FIRE    4 cleanup-2: f-33,f-22
<== f-22    (potential-win 3 2 side o)
**FE*
FIRE    5 cleanup-2: f-33,f-23
<== f-23    (potential-win 1 2 side o)
**FE*
FIRE    6 cleanup-2: f-33,f-24
<== f-24    (potential-win 3 1 corner o)
**FE*
FIRE    7 cleanup-2: f-33,f-25
<== f-25    (potential-win 2 1 side o)
**FE*
FIRE    8 cleanup-2: f-33,f-26
<== f-26    (potential-win 1 1 corner o)
**FE*
FIRE    9 switch-to-evaluate: f-33
<== f-33    (cleanup for computer)
==> f-34    (evaluate board for computer)
==> Activation -10    switch-to-move: f-34
==> Activation 0      evaluate-potential-win: f-34,f-30,f-7,f-4
==> Activation 0      evaluate-potential-win: f-34,f-30,f-4,f-7
==> Activation 0      evaluate-potential-win: f-34,f-30,f-3,f-2
==> Activation 0      evaluate-potential-win: f-34,f-30,f-2,f-3
==> Activation 0      evaluate-potential-win: f-34,f-15,f-8,f-2
==> Activation 0      evaluate-potential-win: f-34,f-15,f-7,f-3
==> Activation 0      evaluate-potential-win: f-34,f-15,f-6,f-4
==> Activation 0      evaluate-potential-win: f-34,f-15,f-4,f-6
==> Activation 0      evaluate-potential-win: f-34,f-15,f-3,f-7
==> Activation 0      evaluate-potential-win: f-34,f-15,f-2,f-8
**FE*
FIRE   10 evaluate-potential-win: f-34,f-15,f-2,f-8
==> f-35    (potential-win 2 3 side o)
**FE*
```

## 10. SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| $x >> y$ | x is much much greater than y |
| $\lvert x \rvert$ | number of elements comprising x, length of x, size of x |
| O(m) | f(m) is of O(g(m)) for any $c_1$, $c_2$ iff $c_1 g(m) \geq c_2 f(m)$ for $m > m_0$ |
| FTM | Fault Tolerant Monitor |
| LHS | Left Hand Side of a CLIPS Rule |
| RHS | Right Hand Side of a CLIPS Rule |

## 11. REFERENCES

1. Chester, M., Fault-Tolerant Computers Mature, *Systems & Software*, March, 1985.

2. Stallings, William. *Computer Organization and Architecture*, 1987, MacMillian Publishing Company, 418, 428.

3. Theuretzbacher, N. Expert Systems Technology for Safety-Critical Real-Time Systems, *Electrical Communication*, No. 2 (1986), 147-152.

4. *CLIPS Reference Manual*, Version 4.3 of CLIPS, Artificial Intelligence Section, NASA, June 1989, pp I-5, II-56.

5. Giarrantano, Joseph C., *CLIPS User's Guide*, Version 4.3 of CLIPS, Artificial Intelligence Section, NASA, June 1989, p 20.

6. Stallings, W., *ISDN An Introduction*, New York, NY, MacMillian Publishing Company, 1989, p 361.

# CLIPS/Ada
# an Ada-Based Tool for Building Expert Systems

W. A. White

Barrios Technology Inc.
1331 Gemini
Houston, Texas

April 4, 1990

*Abstract. This paper describes the CLIPS/Ada expert system tool. It explains the reason for the tool's development, gives examples of its use, and addresses the tools performance and portability. The author assumes that the reader is familiar with CLIPS.*

## 1 Introduction

Clips/Ada is a production system language and a development environment. It is functionally equivalent to the CLIPS tool. CLIPS/Ada was developed in order to provide a means of incorporating expert system technology into projects where the use of the Ada language had been mandated. A secondary purpose was to glean information about the Ada language and its compilers. Specifically, whether or not the language and compilers were mature enough to support AI applications. The CLIPS/Ada tool is coded entirely in Ada and is designed to be used by Ada systems that require expert reasoning.

### 1.1 Program Development

The first fully functional prototype was completed in August of 1988. A period of redesign followed. Several versions were released to an independent company for verification and validation. Due to the C version's continuing evolution, many new features were also added during this time to maintain compatibility. The first official release of CLIPS/Ada was in October of 1989. This version is equivalent to CLIPS version 4.3 when compiled with the "generic" option (i.e. the setup.h "generic"flag is set to "1" before compilation).

## 2 CLIPS/Ada Interface

CLIPS/Ada has two interfaces. One is interactive and is used for interactive expert system development. The other is embeddable and was designed to allow other programs to easily interface with CLIPS/Ada.

### 2.1 Using the Interactive Interface

CLIPS/Ada's interactive interface is identical to the generic CLIPS interface. It does not support non-generic features such the integrated editor and windowing capability that some implementations of CLIPS provide. Aside from these differences the user should not be able to distinguish between the CLIPS/Ada interactive interface and the CLIPS interactive interface.

### 2.2 Using the Embedded Interface

Using the embedded interface to link CLIPS/Ada to and existing Ada program is straight-forward. Since CLIPS/Ada is all Ada, its procedures and functions can be called like any other Ada subroutines. No knowledge of parameter passing mechanisms or interface pragmas is necessary as are needed when languages are mixed. An example of how one might use the embedded interface is shown in figure 2.1. The names of the subroutines should seem familiar to those with CLIPS programming experience.

```
with Text_Io, Embedded_Clips;
use  Text_Io, Embedded_Clips;

procedure Example is
   Status      : Boolean;
   Fact_Value  : Fact;
   Rules_Fired : Integer;

begin

   Init_Clips;
   Put_Line("Loading rules. Please wait...");
   Status := Load_Rules_File("example.clp");
   If Status then
     Put_Line("Load Completed.");
   else
     Put_Line("Load Failed.");
     return;
   end if;
   Reset_Clips;
   Fact_Value := Assert_Fact("Example-Fact has-value 1");
   Rules_Fired := Run_Clips;
   Status := Retract_A_Fact(Fact_Value);

end Example;
```

**Figure 2.1: Embedded CLIPS/Ada Example**

## 3 Metrics

CLIPS/Ada, as those with Ada programming experience might suspect, runs slower than CLIPS. Two factors that significantly contribute to this slow down are: support of strong type checking by insertion of extra machine code and inefficient handling of dynamic memory. On some machines it was possible to suppress many checks without altering program behavior. This is because the hardware on some machines automatically performs these checks and the checks produced by the compiler are apparently redundant. On other machines this was not the case and neither pragmas nor compiler directives that suppressed checks could be used.

### 3.1 Speed

Run time performance varies depending on the machine, the compiler, and the compiler flags used. By improving memory management schemes and using maximum safe compiler optimization switches the speed gap between the Ada version and C version has been closed from greater than six times to about two times slower. In some test cases involving little I/O and no multi-field variables the Ada version's speed nearly matched that of the C version. Table 3.1 shows the results of informal benchmark. A formal comparison of CLIPS/Ada and other shells is currently underway.

For the comparison a classic AI demo program, "Monkeys and Bananas" was selected. This program was chosen over benchmarking programs that employ artificial sequences of patterns because such programs tend to not accurately reflect the way a real application will perform. For example, a series of test in which the rule's patterns follow the sequence (aaa), (bbb), (ccc)... will cause the discrimination net used by CLIPS to degenerate -- thus the patterns above produce a "worst case" scenario. While the sequence (a a a), (a a b), (a a c)... will produce a near "best case" scenario.

The program was executed on a MicroVAX-II. The "(opt)" in the tables means that maximum compiler optimization was selected. This includes suppressing type checking when safe and applicable. The "(O+M)" in the tables means that the program was compiled with full optimization and the program tested was the version of CLIPS/Ada that manages its own memory.

| CLIPS Version | Compiler Flags Used | Mean Run-Time |
|---|---|---|
| CLIPS 4.3 | CC/Nooptimize | 1.40 seconds |
| CLIPS 4.3 (opt) | CC/Optimize | 1.31 seconds |
| CLIPS/Ada 4.3 | Ada/Nooptimize | 7.17 seconds |
| CLIPS/Ada 4.3 (opt) | Ada/Optimize=Time/Nocheck | 3.80 seconds |
| CLIPS/Ada 4.3 (O+M) | Ada/Optimize=Time/Nocheck | 3.09 seconds |

**Table 3.1a: Comparison of CLIPS and CLIPS/Ada (Mean Run-Time)**

| vs. | CLIPS 4.3 | CLIPS 4.3 (opt) | CLIPS/Ada | CLIPS/Ada (opt) | CLIPS/Ada (O+M) |
|---|---|---|---|---|---|
| CLIPS 4.3 | 1.00 | | | | |
| CLIPS 4.3 (opt) | 0.94 | 1.00 | | | |
| CLIPS/Ada | 5.12 | 5.47 | 1.00 | | |
| CLIPS/Ada (opt) | 2.71 | 2.90 | 0.53 | 1.00 | |
| CLIPS/Ada (O+M) | 2.21 | 2.36 | 0.43 | 0.84 | 1.00 |

**Table 3.1b: Relative Speeds of CLIPS and CLIPS/Ada (Ratio of Mean Run-Times -- Rows / Columns)**

## 3.2 Portability

CLIPS/Ada is portable to any machine that has a reliable Ada compiler. Table 3.2 shows the latest versions of the compilers that CLIPS/Ada has been tested on. It is interesting to note that during development bugs were uncovered in all of the compilers listed below. Most were not serious, however, and were fixed in subsequent releases of the compilers.

| COMPILERS | MACHINES | | | | | | |
|---|---|---|---|---|---|---|---|
| | HP-370 | PC/286 PC/386 | VAX 11/780 MicroVAX II | MAC-II | Sun 3/60 | Harris | Mips M/120 |
| Alsys | - | 4.3.1 | - | - | 4.2 | - | - |
| VAX/Ada | - | - | 1.5 | - | - | - | - |
| Meridian/Ada | - | 4.0* | - | 3.0 | - | - | - |
| HP/Ada | 4.35 | - | - | - | - | - | - |
| Vertix | - | - | - | - | - | 5.41 | 2.10** |

**Table 3.2: Machines Known to Run CLIPS/Ada**

*** Compiler produces unreliable code**
**** Reported to Run on Mips**

## 3.3 Memory Requirements

The CLIPS/Ada executable is larger than the CLIPS executable. One reason for this is Ada 's type checking. Another is Ada's lack of a C-like preprocessor. CLIPS uses the C preprocessor to allow the user to selectively omit segments of code or even whole modules which are not needed. For example, the extended math library will not be included in the executable if the user sets the extended math preprocessor flag to 0. Other reasons for the increase in size are: the inefficient compilation of generic instantiation, and the incorporation of unnecessary run-time library overhead into the program.

## 4 Future Directions

Future plans include improving the program's execution speed, adding a "rules-to-ada" function analogous to "rules-to-c", investigating decomposing CLIPS/Ada into a component library, and continuing to match new capabilities of CLIPS.

### 4.1 Increasing the Program's Execution Speed

Increasing speed remains a high priority. However, major improvements to execution speed will probably depend on the development of better Ada compilers and faster hardware. Certain extensions to the Ada language would also allow programs to run faster. CLIPS depends heavily on the ability to call a function using its address. If Ada had this capability it would greatly reduce the amount of time CLIPS/Ada spends evaluating expression trees, and it would also increase I/O efficiency.

### 4.2 Adding a "rules-to-ada" Function

A "rules-to-ada" function is being completed. When invoked this function will generate Ada code that can replace the rules of an expert system. To use this feature requires three steps. First, the desired rules must be loaded. Then the rules-to-ada command must be invoked. This command will produce several Ada specifications. These specifications contain the information necessary for rebuilding the data structures that were produced during the loading of the CLIPS rules set. The final step is to compile the generated modules and relink the main program. After these steps are done the expert system can be called upon without continually have to load it. This eliminates load time and in some cases reduces memory requirements, since modules that are only needed for compiling the rules can be replaced with stubs.

## 4.3 Decomposing CLIPS/Ada into Components

Since each expert system application is different, each has different requirements of its shell. At on extreme are applications which require great speed but do not depend on powerful features. For example a real-time fault-diagnoses and correction expert system. At the other extreme are those that require powerful features while speed is not as critical. For example, a system that monitors traffic on a computer network and periodically adjust parameters in order to fine tune the system.

Meeting the requirements of both extremes simultaneously may be possible if CLIPS/Ada were to be decomposed into components. The user could then construct a customized tool by combining components. If Ada were to provide a means of conditional compilation it would be possible to, alternatively, customize CLIPS/Ada by using preprocessor flags.

## 5 Conclusions

CLIPS/Ada was created in response to the demand for Ada-based expert system tools. It is portable and has been tested on a variety of machines. CLIPS/Ada is syntactically identical to CLIPS and is similar to other rule-based shells . Porting a production system from CLIPS to CLIPS/Ada is trivial. And in many cases it is not difficult to go from another shell to CLIPS/Ada or visa-versa. CLIPS/Ada is distributed by COSMIC. It is available free for government use and at a small fee for private use.

## A5 Session:
## Space Shuttle Quality Control/Diagnosis

# SYSTEM CONTROL MODULE DIAGNOSTIC EXPERT ASSISTANT

LUIS M. FLORES
ROGER F. HANSEN

LOCKHEED ENGINEERING AND SCIENCES CO.
2400 NASA Road 1
P O Box 58561
Houston, TX 77258

## ABSTRACT

The Orbiter EXperiments (OEX) Program was established by NASA's Office of Aeronautics and Space Technology (OAST) to accomplish the precise data collection necessary to support a complete and accurate assessment of Space Transportation System (STS) Orbiter performance during all phases of a mission. During a mission, data generated by the various experiments are conveyed to the OEX System Control Module (SCM) which arranges for and monitors storage of the data on the OEX tape recorder. The SCM Diagnostic Expert Assistant (DEA) is an expert system which provides on-demand advice to technicians performing repairs of a malfunctioning SCM. The DEA is a self-contained, data-driven knowledge-based system written in the 'C' Language Production System (CLIPS) for a portable micro-computer of the IBM PC/XT class. The DEA reasons about SCM hardware faults at multiple levels; the most detailed layer of encoded knowledge of the SCM is a representation of individual components and layouts of the custom-designed component boards.

In the demonstration phase, which has been completed, knowledge required to troubleshoot the analog components of the SCM, specifically the switching array and its input and output buffers, was encoded.

Since the Symbolics machines present a very powerful development environment, the development of the demonstration phase was done on a Symbolics in ART in a manner to simulate the restrictions imposed by the CLIPS environment. The information included in the demonstration phase's knowledge base was gathered via scheduled question and answer sessions with a senior expert. Subsequent analysis and critique sessions with knowledgeable technicians provided the vital refinement of the prototype system developed on the Symbolics.

Transfer of the first phase of the DEA to the PC CLIPS environment necessitated rewriting the various data base access functions from Lisp to 'C' and required further refinement of the component data base to facilitate its installation in the PC. The DEA successfully demonstrated proof-of-concept of the proposed design and development methodology.

# SYSTEM CONTROL MODULE DIAGNOSTIC EXPERT ASSISTANT

LUIS M. FLORES
ROGER F. HANSEN

LOCKHEED ENGINEERING AND SCIENCES CO.
2400 NASA Road 1
P O Box 58561
Houston, TX 77258

## BACKGROUND

The Orbiter Experiments (OEX) Program was established by NASA to accomplish the precise data collection necessary to support a complete and accurate assessment of Space Transportation System (STS) Orbiter performance the launch, boost, orbit, atmospheric entry, and landing phases of a mission. Prior to each mission the Orbiter is outfitted with instrumentation associated with the suite of OEX experiments to be performed for that mission. During the mission, data generated by the various experiments are conveyed to the OEX System Control Module (SCM) which arranges for and monitors storage of the data on the OEX tape recorder. After return of the Orbiter, the data tape is played back for re-recording on a ground system. Data obtained from OEX program experiments are constantly being utilized, both to improve the design of the orbiter and to reduce its life-cycle costs, by demonstrating ground-based laboratories.

## SCM SYSTEM DESCRIPTION

The SCM provides flexible manual and automatic control of the OEX recorder, OEX experiments and other OEX components. In response to commands received via the Orbiter's Multiplexer/Demultiplexer (MDM, a multi-purpose communications bus), the SCM controls the manner in which the OEX recorder records data from the OEX experiments and the Orbiter's Modular Auxiliary Data System (MADS). The SCM can control the recording of up to 32 data inputs on any of 32 output channels (the OEX recorder's 28 data tracks plus four T-O umbilical lines). It controls all recorder operating characteristics (eg., tape speed, direction) and provides discrete commands for experiment control functions.

The SCM contains a semiautomatic feature in which a complete data recording configuration can be established by a keyword command and an automatic provision whereby the OEX data recording sequence for an entire mission can be initiated by command. The SCM interfaces with the various OEX experiments; the OEX tape recorder; MADS, Orbiter power, instrumentation, mechanical, and thermal subsystems; and (dedicated) ground support and equipment (GSE).

The SCM-to-GSE interface is designed to allow check-out of the SCM and tape recorder in four situations: laboratory check-out and trouble-shooting, qualification and flight level acceptance testing;

pre-installation testing at John F. Kennedy Space Center (KSC); and preflight and postflight testing of the hardware while it is installed in the Orbiter. The GSE is designed so that power, signal levels, timing, command word formats, and digital/discrete signals are identical to those provided by the Shuttle interface.

## SCM DESCRIPTION AND HARDWARE FAULTS DIAGNOSIS

The SCM consists of 6 solid-state component cards, power supplies, and external connectors (for power, data input, and data output). The SCM includes both digital and analog components which function under the control of a model 1802 CPU (connected by auto-switch to a duplicate backup CPU). The analog portion of the SCM handles routing of OEX experiments output data to the OEX tape recorder.

## OBJECTIVE

The ultimate objective of this project is to develop an expert system to assist in diagnosis and repair of malfunctions in the SCM flight hardware. The intermediate objective of this project is to prove that the development of the system can be accomplished incrementally in a non-portable development environment (Symbolics/ART) with the subsequent transfer of the completed system to a portable micro-based computer for ease of field access.

## DESCRIPTION OF THE EXPERT SYSTEM

The SCM Diagnostic Expert Assistant (DEA) is an expert system which provideds on-demand advice to engineers performing repairs of a malfunctioning SCM. The DEA is a self-contained, data-driven, knowledge-based system written in CLIPS and is targeted for delivery on a micro-computer of the IBM PC/XT or AT class. The DEA reasons about SCM hardware faults at multiple levels; the most detailed layer of encoded knowledge of the SCM is a representation of individual components and layouts of the custom-designed component boards.

## APPROACH

The DEA is being developed in two phases. In phase one, knowledge required to trouble-shoot the analog components of the SCM, specifically the switching array and its input and output buffers, was encoded. In phase two, diagnostic aids for the remaining SCM components will be added to the DEA.

A preliminary system that included a sample of the strategy for the representation of the system's components, and the rules for the diagnosis of a part of the output buffer board was implemented in ART to test the feasibility of the proposed system architecture. This system constituted a proof of concept for the system design. In order to gain efficiencies in time and space which were necessary to adapt the system to a microcomputer, some functions had to be written in Lisp in the Symbolics.

An analogous proof of concept for the system's design was built to operate on a micro-computer of the IBM/XT or AT class based on the CLIPS inference tool. A suitable lower level language ('C') was selected for the microcomputer for some selected data base access functions to produce executable programs.

## STRATEGY

Since the Symbolics machines present a very powerful development environment, the development of phase one was done on a Symbolics in a manner to simulate the restrictions imposed by the CLIPS environment. The information included in phase one's knowledge base was gathered via scheduled question and answer sessions with a senior expert. Subsequent analysis and critique sessions with knowledgeable technicians provided the vital refinement of the prototype system developed on the Symbolics.

The feasibility of a system design based on viewpoints, similar to ART, was investigated and a viewpoint mechanism for CLIPS was developed. This strategy allowed the system to reason about an individual component without forgetting knowledge that has been previously gained.

Since CLIPS does not have the capability for representation of objects based on schemata, a hierarchical representation of the components and parts of the SCM was developed and tested.

The feasibility of designing a strategy that would allow the system to reason about generic components whenever possible was investigated and led to the development of a data base containing both generic and actual component names. This minimizes the size of the knowledge base and the number of facts in the knowledge base during a run. The design was done in a manner that is transparent to the user so that the user would refer normally to specific components.

Transfer of phase one of the DEA to the PC CLIPS environment necessitated requiring the various data base access functions in 'C' and required further refinement of the component data base to facilitate its installation in the PC.

## RESULTS AND CONCLUSION

The first phase of the system consists of the knowledge base and rules for the three out of the six boards. It is being used for demonstrations and for obtaining feedback from users prior to final completion of the project. Simulated diagnosis sessions have been held to attempt to cover as many as the possible alternative paths through the system as feasible.

The present system is contained in a bootable low density diskette and can be operated in a computer of the XT class. The system interacts with the user through questions and instructions. In all cases, hardware faults are pursued to the individual component level.

The design that allows the system to reason about generic components while to specific components was successful. The conversion from Symbolics ART to CLIPS on a PC was accomplished with a minimum of difficulty. The system can guide any technician who can read the engineering drawings through the tests leading to the identification of the defective hardware component.

# Prototype Automated Post-MECO Ascent I-Load Verification Data Table.

George D. Lardas

McDonnell-Douglas Space Systems Company
Engineering Services Division
16055 Space Center Blvd.
Houston, Texas 77062

## Abstract

A prototype automated processor for quality assurance of Space Shuttle post-Main Engine Cut-Off (MECO) ascent initialization parameters (I-loads) is described. The processor incorporates Clips rules adapted from the quality assurance criteria for the post-MECO ascent I-loads. Specifically, the criteria are implemented for nominal and abort targets, as given in the "I-load Verification Data Table, Part 3, Post-MECO Ascent, Version 2.1, December 1989." This processor, ivdt, compares a given I-load set with the stated mission design and quality assurance criteria. It determines which I-loads violate the stated criteria, and presents a summary of I-loads that pass or fail the tests.

## Introduction

Originally, the verification and quality assurance of Space Shuttle flight software initialization parameters (I-loads) was the responsibility of the I-load designers. Over the course of time, the rules-of-thumb and flight design experience were distilled into a document, the "I-Load Verification Data Table" (IVDT). Up to now I-loads have been verified manually according to guidelines provided in the IVDT. However, this has manifest drawbacks, since it is difficult for a human operator to maintain constant, close attention to repetitive detail, especially when a large number of parameters are to be examined. Not only does fatigue reduce the effectiveness of manual quality assurance, but even with the formalized procedure outlined in the IVDT, individual differences in the quality assurance process are inevitable.

An automated processor for I-load quality assurance would address these difficulties. Since the criteria for quality assurance are summarized as rules in the IVDT, a rule-based language would be a natural choice for programming the processor. Therefore this processor, ivdt, was developed in Clips. (Henceforth, upper case IVDT signifies the document or the body of requirements for QA, while lower case ivdt signifies the program or processor.)

This processor should be relatively simple to use, and it should be possible to enter the I-loads and whatever simulation data are needed from the designer with little or no manual preprocessing. In fact, I-load QA automation has three parts: automation of the decision-making based on verification criteria, automation of I-load data entry, and automation of the entry of performance data from a simulation program. However, ivdt only automates the I-load QA decisions, while the I-loads and simulation data must still be entered manually into files with a text editor prior to operation.

## I-Loads

The Space Shuttle flight software initialization loads (I-loads) are parameters specific to the particular mission, and are set before each flight. These I-loads are divided into several categories, depending on the phase of flight, the functions they govern, and the contingencies for which they are designed. Here we are only concerned with post-MECO ascent for three types of mission, nominal (NOM), abort-to-orbit (ATO), and abort-once-around (AOA). The I-loads corresponding to these contingencies are listed in the IVDT as belonging to categories A3N, A3T, and A3A, respectively. Post-MECO ascent has two orbital maneuvering system (OMS) burns, OMS-1 and OMS-2. The following parameters comprise the post-MECO ascent I-loads:

| | |
|---|---|
| DTIG | time interval from external tank separation ignition (sec) |
| C1 | intercept of vertical vs. horizontal velocity target line (fps) |
| C2 | slope of vertical vs. horizontal velocity target line (dimensionless) |
| HTGT | the height of the OMS target point above the earth (ft) |
| THETA | in-plane downrange angle from launch site to OMS target point (rad) |

There is one set of the above I-loads for each of the nominal OMS burns, but multiple target sets for ATO and AOA. The total number of post-MECO I-loads is nearly 160.

I-loads are identified in ivdt by FSSR (Flight Systems Software Requirements) name. The FSSR name generally has the following parts: the basic name of the I-load (from the above list), the contingency category (if applicable: AOA or ATO) and either OMS, OMS1, or OMS2. These two or three parts are separated by an underscore, and most of the FSSR names are followed by one or two subscripts in parentheses separated by a comma. Nominal I-loads have only one subscript, which gives the number of the OMS burn. ATO OMS-2 and AOA OMS-1 I-loads have one subscript, which corresponds to the target set number (ITGT) to which it belongs. AOA OMS-2 I-loads have two subscripts, the first of which is related (but not identical) to the ITGT number, and the second pertains to one of the three terms in a quadratic equation dependent on variable-plane targeting. So far this option has not been used, and the linear and quadratic terms (subscripts 2 and 3) have always been set to zero, leaving only the constant value term (subscript 1). The following are examples of I-load names:

```
DTIG_OMS(2)
HTGT_ATO_OMS1
THETA_AOA_OMS2(7,1)
C2_ATO_OMS2(2)
```

Since parentheses are delimiters in Clips, and cannot be input as part of a string by Clips i/o functions, all parentheses must be replaced by square brackets. FSSR names are used as words in the ivdt fact base, and therefore must not contain any internal spaces. Since commas are not delimiters, and can be part of a Clips word, they may be entered unchanged. All inputs to ivdt must be in lower case. Thus the above I-loads should be entered as

```
dtig_oms[2]
htgt_ato_oms1
theta_aoa_oms2[7,1]
c2_ato_oms2[2]
```

## Criteria

The I-loads must be checked against criteria summarized in the "I-Load Verification Data Table." The table specifies three kinds of constraints: the I-load values must fall within given ranges, the I-loads must be consistent with each other, and they must result in simulated flight parameters within acceptable ranges. Three classes of criteria can be distinguished: limit checks, consistency checks, and performance checks. The I-load limits checks combine mathematical, physical, and mission constraints. For instance, THETA must fall within the mathematical limits of zero to twice pi in all cases. HTGT cannot be less than zero for nominal OMS-2, and must be high enough to ensure a stable orbit for the duration of the mission plus a certain margin. An example of a consistency criterion is that for nominal and abort to orbit, THETA for OMS-2 must be approximately pi greater than that for OMS-1. An example of a performance check is that the apogee and perigee of the final orbit must be within a certain margin of the nominal.

## Implementation

The program, ivdt, consists of a rule base operating under the Clips 4.3 interpreter. Since interpreted rules yield adequate speed for the preliminary version of the processor, a compiled version has not been attempted. Only standard Clips is used, and no user-defined functions have been implemented. The source code is contained in seven source code files and can be loaded into the interpreter manually by the operator, or by running the batch file, ivdt, with the command, `(batch "ivdt")`:

```
; batch file:  ivdt

(clear)
(close)
(load "ivdt0")
(load "ivdt1")
(load "ivdt2")
(load "ivdt3")
(load "ivdt4")
(load "ivdt5")
(load "ivdt6")
(reset)
```

To run the program, the operator must then start it with the `(run)` command, or run the following batch file, istart:

```
; istart

(batch "ivdt")          ; load ivdt
(run)                   ; run program
tflg                    ; enter name of test flag file
fout                    ; enter name of output file
(save-facts "fact")     ; save internal state at termination
```

Istart loads ivdt, enters the test flag input file name (which contains all further information the program needs to run) and the output file name. When ivdt terminates, either normally or abnormally, the command, save-facts, saves the internal state (fact base) of the processor. This can be helpful for diagnosis of abnormal operation.

## Inputs

The program, ivdt, requires several input files, whose internal logical names are as follows:

| Designator | Description |
|---|---|
| dsgn | mission design parameters |
| inpt | I-loads |
| iref | I-load reference data, including limits |
| itgt | correlation between OMS-1 and OMS-2 target set numbers |
| shdr | header and title of simulation run pages in QA package |
| sims | simulation run results (performance data) |
| tflg | test flags and file input modes |

The file entries are in free format records, one record to a line, and may contain trailing comments or even entire comment lines introduced by semicolons. The file logical name must be the first datum in these files and is used as a check on the file format and information content. Ivdt may either prompt the operator for the system file name of these files, or they may be specified in the first file opened, tflg. For most of these files, a line of data is entered as a fact by doing a str-cat on the line to the file logical name and then a str-assert. In tflg, however, the facts are entered directly from the file by a simple str-assert.

The file, tflg, contains flags that activate the tests the operator wishes to run on the I-loads, as well as the file logical names, file input modes, and, optionally, the system file names, that the operator wishes to use. The following is an illustration of the contents of this file (not complete):

```
tflg            ; input flag must be first datum in file

; input flags:

; fname dsgn                                ; design criteria
  file dsgn "dsgn35" "r" consistency-check ; direct input of file
  enter-file dsgn                           ; enter entire file

; fname inpt                                ; iload input
  file inpt "inpt35" "r" input-iloads       ; direct input of file
  enter-file inpt                           ; enter entire file

  tflg all                                  ; all tests
; tflg lmts                                 ; limit test
```

Note that semicolons are used not only to introduce comments, but also to comment-out options not used. This file is entered directly by str-assert without prefixing the word, tflg. When the option, fname, is used, the program prompts the operator for the system file name of these files. The option, file, gives the system filename directly, the read/write/append mode, and the phase when the file is to be opened. The option, enter-file, indicates that the facts should be entered from these files with the file's logical name as the first word. The option, tflg, activates some particular subset of tests.

The I-loads are entered in the file, inpt. Inpt contains the I-loads listed by FSSR name with their numerical values, one I-load per line, as illustrated below (not complete). Each input line is asserted as a fact with the word, inpt, prepended. This file is needed for all ivdt test modes.

```
inpt            ; input flag must be first datum in file

; fssr                    value

; category a3n

dtig_oms[1]               102
htgt_oms[1]               1183800
theta_oms[1]              2.610
dtig_oms[2]               1895
htgt_oms[2]               1196995
theta_oms[2]              5.75526
```

The file, iref, contains reference data on the I-loads, and is the same for all flights. Therefore, once this file is created it need not be changed. Each data line in iref contains an I-load FSSR name, its MSID (a unique I-load identifier), the simple I-load name, its contingency category, OMS burn, target set number, and lower and upper limits. The word, iref, introduces all facts asserted from this file. This file is needed for all ivdt test modes. The following illustration is a sample, and not complete:

```
iref            ; file flag must be first datum in file

; fssr         msid       name    cat  oms  itgt lwr      upr

; category a3n

dtig_oms[1]   v97u4367c  dtig    nom  oms1  0   102      102
htgt_oms[1]   v97u4411c  htgt    nom  oms1  0   6.076e5  2.7108e6
theta_oms[1]  v97u4803c  theta   nom  oms1  0   1.6      3.4
dtig_oms[2]   v97u4368c  dtig    nom  oms2  0   1200     2250
htgt_oms[2]   v97u4412c  htgt    nom  oms2  0   6.076e5  2.7108e6
theta_oms[2]  v97u4804c  theta   nom  oms2  0   3.5      7.0
```

The file, dsgn, contains flight design parameters, such as heights of apogee and perigee, inclination, and other data. This file is needed for ivdt performance check modes. The word, dsgn, introduces all facts asserted from this file. A complete example follows:

```
dsgn            ; input flag must be first datum in file

; parameter     value   ; units

insertion       direct
inclination     28.45   ; deg
apogee          190     ; nmi
```

```
perigee          190      ; nmi
var-iy           off
```

The file, itgt, contains the correlation between OMS-1 and OMS-2 target set numbers for nominal and contingency flights. This is sometimes given explicitly in the QA summaries, but sometimes it must be deduced from the notes in the I-load tables, or from the simulation run summaries. This file is needed for ivdt consistency and performance check modes. All fact asserted from this file are introduced with the word, itgt (this example is complete):

```
itgt              ; input file flag must be first datum in file

; itgt1 itgt2

     0    0
     1    1
     1    2
     3    3
     4    4
     5    5
     1    6
     1    7
     0    8
     0    9
     0   10
     0   11
     0   12
```

The file, shdr, contains header information for each page of simulation run summaries, each page corresponds to a single simulated flight. While this file repeats information contained in itgt and sims, it must be entered for ivdt performance checks. Every fact asserted from this file begins with the word, shdr. Here is a sample of the contents:

```
shdr              ; input flag must be the first datum in file

; id  oms1   oms2    title

p16     0      0     dinom
p17     0     10     di-st10
p18     0     10     di-st10
p19     0     11     di-sh11
```

The file, sims, contains simulated flight performance data taken from the QA summaries. The data include the heights of apogee and perigee before and after each OMS burn, the VGO at the beginning of each burn, and the true anomaly at OMS-2. This file must be entered for ivdt performance checks. Facts asserted from this file begin with the word, sims. A sample of the contents is given below:

```
sims             ; input flag must be the first datum in file

; id  name   oms   qual  val

p16   ha     oms1  co    185.72
p16   hp     oms1  co     33.84
p16   ha     oms1  tig   185.72
p16   hp     oms1  tig    33.84
p16   vgo    oms1  tig     5.14
p16   ha     oms2  co    190.27
p16   hp     oms2  co    189.88
p16   ha     oms2  tig   189.93
p16   hp     oms2  tig    35.01
p16   vgo    oms2  tig   277.27
p16   truea  oms2  tig   173.65
```

While the input files can be given any system file name arbitrarily, it is helpful to adhere to a consistent nomenclature to avoid confusion. One possible naming convention is to use the internal logical names with the Shuttle flight number appended, as for instance, I-loads from STS-35 might be stored in the file inpt35. Likewise the other files could be labelled dsgn35, shdr35, sims35, and tflg35. However, since iref is common to all flights, it could simply be named iref.

## Processing

Processing is divided into several phases, with a different set of rules potentially active in each. The phases are as follows:

| **Phase** | **Description** |
|---|---|
| initial | initial displays, initialize facts, set flags |
| input-iloads | load I-loads as facts |
| limits-check | check I-loads against specified limits |
| consistency-check | check I-loads against consistency criteria |
| performance-check | check simulation results against performance criteria |
| accumulate | accumulate tallies of tests passed and failed |
| count | count tallies |
| final | final displays, output results |

These correspond roughly to the seven source code files:

| **File** | **Description** | **No. of Rules** |
|---|---|---|
| ivdt0 | phase change facts, rules common to all phases | 9 |
| ivdt1 | initial phase, begin, open and verify data files | 11 |
| ivdt2 | input I-loads and check limits | 4 |
| ivdt3 | consistency checks | 8 |
| ivdt4 | performance checks | 10 |
| ivdt5 | accumulate and count results | 12 |
| ivdt6 | final phase, close all files and exit | 5 |
| | | --- |
| | **Total** | 59 |

The rules are contained in the source code files are derived from the quality assurance criteria given for the various I-loads in the IVDT document. Most of the IVDT criteria have been implemented as rules. However, some of the stated criteria involve AOA re-entry conditions and landing site targeting for which no information is available in the QA summaries, and so these rules were not coded. A sample rule, taken from source code file, ivdt4, and active under the phase, performance-check, is illustrated here:

```
(defrule orbit-ht-ato             ; check ato final orbit ht
  (phase performance-check)           ; phase flag
  (shdr ?page ? ?it2 ?tl)
  (iload ?fssr htgt ato oms2 ?it2 ?) ; ato oms2 iload
  (sims ?page ha oms2 co ?ha2)        ; oms2 co ht apogee, nmi
  (sims ?page hp oms2 co ?hp2)        ; oms2 co ht perigee, nmi
=>
  (if (and (< (abs (- ?ha2 105)) 5) (< (abs (- ?hp2 105)) 5)) then
    (assert (itst ?fssr ot pass))
    (assert (ptst ?page ot pass))
  else                                ; at least one not nr 105 nmi
    (assert (itst ?fssr ot pass))
    (assert (ptst ?page ot pass))
    (bind ?str1 (format nil
         "Fail consistency (orbit-ht-ato) -- "))
    (bind ?str2 (format nil "Data set %s (%s):" ?tl ?page))
    (bind ?str3 (format nil
         "ATO OMS-2 apogee %5.2f nmi and perigee %5.2f nmi"
          ?ha2 ?hp2))
    (bind ?str4 (format nil
         "should be within 5 nmi of desired altitude 105 nmi"))
    (bind ?str (str-cat "%n" ?str1 ?str2 "%n%n%t"
         ?str3 "%n%t" ?str4 "%n"))
    (assert (message ot ?str))
  )
)
```

This rule is taken from IVDT Part 3, page A3T-1, "Verification Methodology," Section 1, and states that, "The orbit after OMS-2 should be near circular with the apogee and perigee approximately equal to 105 nmi (± 5 nmi)." The first pattern, (phase performance-check) is matched only when the processor is in the performance check phase of operation. The second pattern, (shdr ?page ? ?it2 ?tl), binds the data set page ID and OMS-2 target set number. The third pattern finds an ATO HTGT I-load for OMS-2 whose target set number matches that in that of the data set. The fourth and fifth patterns obtain the heights of apogee and perigee of that data set page after OMS-2 burn. On the right hand side, if the final orbit is within the given limits, the facts that the I-load and the data set pass the test are asserted, otherwise, the fact that they failed, along with a diagnostic message. The other rules are written in the same spirit, for the limit tests, consistency tests, and performance tests.

After all the tests are made, the program accumulates the totals of I-loads and data sets that pass and fail the tests, and generates summary facts, which are used to produce the output.

## Output

The ivdt program produces one output file, whose logical name and system file name are both fout. This contains a detailed summary of the test results. For each I-load that failed a test, the file contains a message giving the name and value of the failed I-load, its desired value, and the name of the test. An I-load summary gives the overall number of I-loads that passed every test, the number that failed at least one test, and the number to which none of the implemented tests applied. For each simulation run in the QA summary that contains non-I-load values that violate some test criterion, fout contains a diagnostic message with the page number of the simulation run, the name and value of the failed parameter, its desired value, and the name of the test that it failed. A data set summary shows the number of pages that pass all stated criteria, the number that failed at least one test, and the number to which none of the implemented tests applied. Finally, a test summary lists each test that was implemented, and how many I-loads and data sets passed or failed the given test. Here we give a sample output:

```
Thu Mar  1 10:08:12 CST 1990

        IVDT v1.0 rev 900223 by gdl

        Automated I-Load Verification Data Table

        Quality Assurance Checker


Number of I-loads processed:  137

        131 passed;    6 failed

Number of data sets processed:  19

         17 passed;    2 failed

Fail consistency (true-anomaly-oms2) -- Data set ato-ato (p31):

        OMS-2 TIG true anomaly 186.97 deg should be
        should be within 4 deg of 175 deg

Fail consistency (delta-v-oms1-ato) -- Data set ato-ato (p31):

        Total delta-v 268.49 fps should be within 3 fps of delta-v
        criterion 261.97 fps based on nominal orbit height 105 nmi

Fail consistency (apogee-oms1-nom) -- Data set dinom (p16):

        Nominal OMS-1 apogee 185.72 nmi should be
        within 0.1 nmi of design apogee 190.00 nmi

Test summary -- I-loads:

          1 passed;    2 failed;  test: true-anomaly-oms2
          0 passed;    3 failed;  test: delta-v-oms1-ato
          0 passed;    1 failed;  test: apogee-oms1-nom
        137 passed;    0 failed;  test: limit-test
         20 passed;    0 failed;  test: aoa-c1-c2
          3 passed;    0 failed;  test: diff-htgt-ato
```

```
                1 passed;    0 failed;  test: htgt-ato-oms1
                4 passed;    0 failed;  test: orbit-ht-aoa
                2 passed;    0 failed;  test: orbit-ht-ato
                1 passed;    0 failed;  test: apogee-oms1-ato
                1 passed;    0 failed;  test: delta-v-oms2-nom
                1 passed;    0 failed;  test: orbit-ht-nom

      Test summary -- Data sets:

                1 passed;    1 failed;  test: true-anomaly-oms2
                0 passed;    1 failed;  test: delta-v-oms1-ato
                0 passed;    1 failed;  test: apogee-oms1-nom
               17 passed;    0 failed;  test: orbit-ht-aoa
                1 passed;    0 failed;  test: orbit-ht-ato
                1 passed;    0 failed;  test: apogee-oms1-ato
                1 passed;    0 failed;  test: delta-v-oms2-nom
                1 passed;    0 failed;  test: orbit-ht-nom
```

In the above summary, we can see that the I-loads that failed fall just outside the specified range. In fact, this set of I-loads passed manual inspection as these deviations were considered insignificant.

## Conclusions

A prototype automated post-MECO IVDT has been developed. While the program does not have all the desirable features of a fully automated I-load QA tester, it does represent a first step. However, the most important part of the process, the reasoning, has been implemented, and this was possible through the rule-based features of Clips.

# An Expert System to Manage the Operation of the Space Shuttle's Fuel Cell Cryogenic Reactant Tanks

Amy Y. Murphey
Rockwell Space Operations Company

**Abstract**

This paper describes a rule-based expert system to manage the operation of the Space Shuttle's cryogenic fuel system [1]. Rules are based on standard fuel tank operating procedures described in the EECOM Console Handbook [2]. The problem of configuring the operation of the Space Shuttle's fuel tanks is well-bounded and well-defined. Moreover, the solution of this problem can be encoded in a knowledge-based system. Therefore, a rule-based expert system is the appropriate paradigm. Furthermore, the expert system could be used in coordination with power system simulation software to design operating procedures for specific missions.

## 1 Introduction

To ensure an adequate and uninterrupted supply of electrical power during all phases of the Space Shuttle's flight, it is necessary to judiciously monitor and control the flow of cryogenic hydrogen and oxygen out of storage tanks and into the fuel cells. To maintain a reliable supply of cryogenic reactants and to provide redundancy and fault-tolerance, the Power Reactant Storage and Distribution (PRSD) tanks must be depleted as evenly as possible, within the constraints of certain guidelines. Therefore, the PRSD system must be configured at certain times during the nominal mission according to, not only the mission profile and initialization specifications, but also an appropriate schedule of tank depletion.

This paper presents a rule-based expert system which may be used for flight design to manage the operation of the Space Shuttle's PRSD system. The expert system provides the user with recommendations on how to configure the PRSD system. That is, for a given state of the PRSD system, the expert system indicates which manifold valves to close and which cryogenic fuel tanks to activate. The knowledge of this expert system is based upon standardized management criteria established by NASA - Johnson Space Center.

## 1.1 Purpose

Flight design analysts of the Space Shuttle's Electrical Power System (EPS) may use this expert system to design PRSD operational plans during preflight analysis. These preflight plans are influenced by the number of tanks flown, the mission profile, and other flight-specific and generic data. As this information is input to this expert system, a PRSD management plan is developed for the entire flight. This expert system will ensure that the analysts use a standard PRSD quantity management plan, which is outlined in the Electrical, Environmental, Consumables and Mechanical (EECOM) Console Handbook [2].

## 1.2 Application

This application of an expert system configures, monitors and controls the PRSD system of the Space Shuttle, according to the analyst's inputs. It is classified as a *configuration* type of application because it must assemble the PRSD components according to a standardized PRSD management plan. These PRSD components consist of the cryogenic network and its controlling switches. Also, this expert system provides a way to *monitor* the Space Shuttle's cryogenic tanks. It queries the analyst for changing tank conditions at certain predetermined times during the mission. If any anomalies are detected, it informs the analyst of this potentially dangerous situation. Furthermore, this expert system is a *control* tool because it regulates how the cryogenic tanks are depleted. It is imperative to maintain strict control of the PRSD system because the uniform depletion of the cryogenic fuel tanks ensures a fault-tolerant and redundant system.

## 2 Rationale for Designing an Expert System

This problem is an ideal candidate for an expert system. Firstly, this problem domain is *well-bounded.* Since the Space Shuttle has a finite number of tank heaters and manifold valves, there are a finite number of valid hardware configurations, which constrain the problem domain. Also, the limits of this expert system's knowledge and its capabilities to manage the PRSD system are *well-defined.* The operation of the cryogenic tank heaters will follow the philosophy outlined in the EECOM Console Handbook [2]. Because this document explicitly outlines what, when and how PRSD configuration decisions should be made, this problem domain is well-defined. Furthermore, no efficient, conventional algorithmic solution exists. Various combinations of facts influence how the tank heaters and manifold valves will be configured. A simple table look-up or decision tree would not be adequate. A *fast pattern matching and pattern recognition algorithm*, such as the Rete Algorithm [3, p.35], is required. Moreover, a rule-based expert system will be *easier to maintain.* If a certain PRSD philosophy is changed or more hardware is added to the Space Shuttle, it may be necessary either to modify the actions of an existing rule or to add other rules. A conventional program may not adapt very well to such changes and may require major modifications to its control structure.

## 3 Problem Domain

During all phases of the Space Shuttle's flight, it is necessary to maintain an adequate supply of cryogenic reactants in each fuel tank. The PRSD system stores and distributes the cryogenic reactants, hydrogen and oxygen, to the fuel cells [1, p.214]. The fuel cells generate electrical power for all the Space Shuttle's electrical equipment during all phases of flight. The PRSD tanks must be depleted as evenly as possible to safeguard against a manifold leak or loss of a tank. Therefore, the PRSD system must be configured at certain times during the nominal mission according to an appropriate, standardized management scheme and the mission-specific data.

## 3.1 Electrical Power System

The Space Shuttle's Electrical Power System consists of three subsystems: Power Reactant Storage and Distribution (PRSD), Fuel Cell Powerplant (FCP) and Electrical Power Distribution and Control (EPDC) [1, p.214]. For the purpose of this expert system, a rudimentary understanding of the FCP and EPDC is necessary. The EPDC distributes electrical power to all the Space Shuttle's electrical components during all mission phases. Three separate FCPs generate the electrical power. Each fuel cell operates independently, producing different amounts of electrical power, as required by its electrical load. The fuel cells require an uninterrupted supply of cryogenic hydrogen and oxygen. Within each fuel cell, a chemical reaction between hydrogen and oxygen occurs, which produces water, heat and electrical power. Therefore, to ensure that enough electrical power is generated during all phases of flight, it is necessary to judiciously monitor the flow of hydrogen and oxygen into the fuel cells.

## 3.2 Power Reactant Storage and Distribution System

The Space Shuttle's Power Reactant Storage and Distribution (PRSD) system consists of a variable number of storage tanks which hold the reactants at cryogenic temperatures and supercritical (compressed) pressures. These hydrogen and oxygen reactants are continuously supplied to the three fuel cells to produce electrical power. Also, the Environmental Control and Life Support System (ECLSS) requires oxygen to maintain the proper partial pressure in the cabin. This system maintains a balance of oxygen and nitrogen to create a breathable environment, which is vital to the crew.

The number of tanks used on a mission will vary, depending on the mission duration and the mission profile. However, there will always be the same number of tanks for each reactant. So, a tank set will refer to a hydrogen tank and a corresponding oxygen tank. The physical specifications of all the oxygen tanks are identical. Likewise, all the hydrogen tanks are identical. Presently, the Space Shuttle can be configured for two, three, four or five tank sets.

The PRSD tanks can be offloaded; that is, the amount of reactant in the tanks can be reduced during prelaunch activities. An offload situation may occur when a full tank is not needed for the mission duration and the Space Shuttle's weight needs to be reduced. Usually, only oxygen tanks are offloaded before liftoff because oxygen weighs considerably more than hydrogen. The oxygen offload requirement should be carefully divided among the tanks, as explained in the Offload Guidelines Section [4]. Moreover, offloaded tanks require special PRSD management criteria because each tank should contain approximately equal amounts of reactant to protect against single point failures.

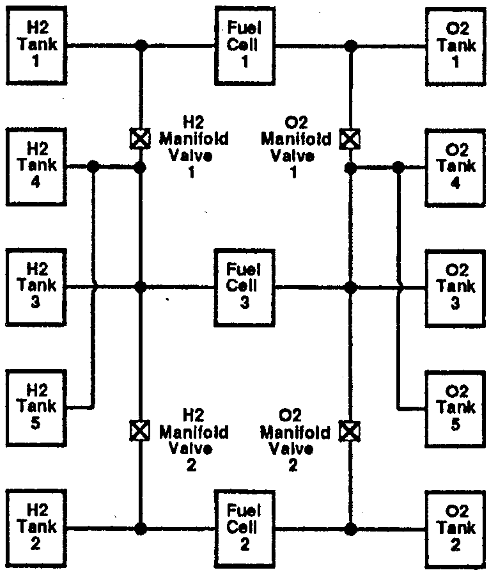


Figure 1: Block diagram of the Power Reactant Storage and Distribution (PRSD) system.

The PRSD system has two manifold valves, which can isolate the system into three separate manifold sections, as seen in Figure 1. Separating the PRSD system into three manifold sections provides a redundant and fault-tolerant system. For example, if a leak occurs within one of the manifold sections, then it can be isolated and a continuous flow of reactants can still be supplied to the fuel cells and the life support system. Also, the standard PRSD management scheme usually involves closing one manifold valve during sleep periods for a single shift crew. During sleep periods, the crew may not respond quickly to a major system leak. If a manifold section was not separated, the cryogenic reactants from all the tanks could be lost before the leak is detected and isolated. This catastrophic leak could interrupt all electrical power production. Therefore, closing a manifold valve ensures that a leak's effects is restricted to that manifold section.

Other components of the PRSD system are cryogenic tank heaters and heater controllers. The tank heaters add heat energy to maintain a constant pressure in the tanks. The cryogenic reactant flows out of the storage tank when the tank pressure is higher than the allowed pressure band. To ensure further redundancy in the PRSD system, there are two heaters, heater A and heater B, for each tank. The heater controller maintains the tank pressure within a certain range. Since certain tanks share heater controllers, the PRSD management plan must account for this coupling of heater operation.

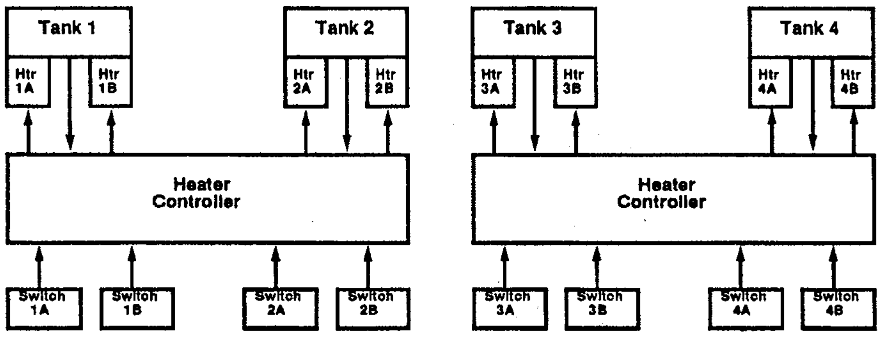


Figure 2: Hydrogen or oxygen heater switches.

To configure which tanks supply hydrogen and oxygen to the fuel cells, it is necessary to decide which tank heaters to activate. However, the heater switches, which set tank heaters to an AUTO, ON or OFF mode, may not operate on the expected tank heater. For each tank, there is a set of switches, which consists of a switch for heater A and a switch for heater B. If four tank sets are loaded, then four of these switch sets are used to control each of the four hydrogen tanks and four are used to control each of the four oxygen tanks. However, the Shuttle can be configured for two, three, four or five tank sets. That is, all of these switches may not be valid when there are fewer than four tank sets. Or, the switches may not operate their respective tank heaters when five tanks are used.

The heater switches are not complicated when two, three, or four tank sets are loaded, as shown in Figure 2. If two tank sets are used, then only the first two sets of switches are used. If three tank sets are used, then the fourth set of switches is ignored (since it is not connected to any tank heater). If four tank sets are being used, then each of the four switches corresponds to the appropriate tank heater.

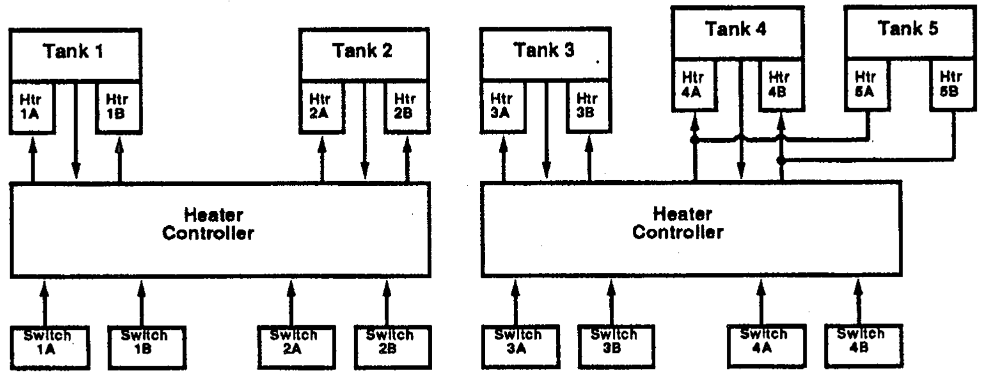


Figure 3: Hydrogen heater switches.

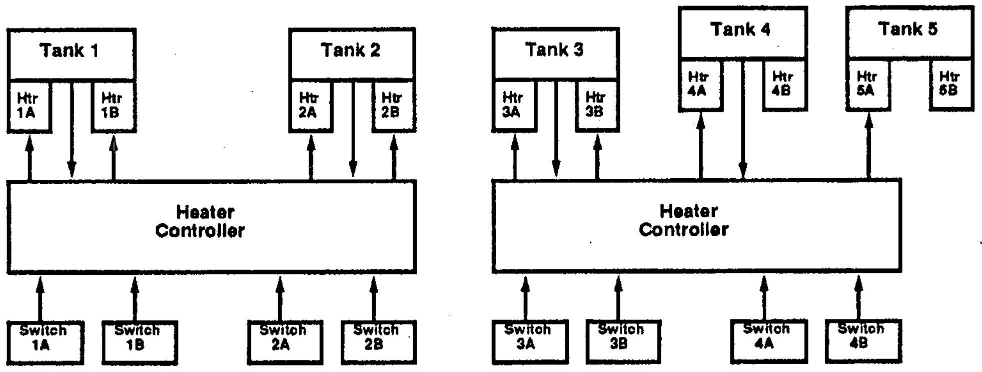


Figure 4: Oxygen heater switches.

However, if five tank sets are used, then the wiring of these switches is not straightforward. The fourth set of switches operates both tanks 4 and 5. But, the operation of hydrogen and oxygen differs. For the hydrogen tanks, switch 4A enables heaters 4A and 5A and switch 4B enables heaters 4B and 5B. This wiring is possible because hydrogen tank 4 will operate alone (in a four tank set configuration) or hydrogen tanks 4 and 5 will operate simultaneously (in a five tank set configuration). Refer to Figure 3. For the oxygen tanks, switch 4A enables heater 4A and switch 4B enables heater 5A. This wiring is possible because the amount of oxygen required from tanks 4 and 5 can always be supplied with A heaters only. Refer to Figure 4.

## 4 Overview of Approach

Expert system technology is an appropriate paradigm for the problem of configuring the PRSD system. More specifically, the decisions of cryogenic management must abide by predefined rules. And, certain rules are activated by the current state of the PRSD system. Therefore, an expert system shell, which separates facts and rules, supports this flexible and robust representation of knowledge. The following paragraphs provide a high-level description of this application, concentrating on how it utilizes an expert system shell.

### 4.1 Structure of an Expert System

A typical expert system is composed of an inference engine, a knowledge base and a working memory. The CLIPS expert system shell provides an inference engine and functions to define the knowledge base and working memory. Also, it provides utility and interface functions such as a debugger and program verifier. How the elements of the CLIPS expert system shell were used in this application is discussed below.

- The *inference engine* uses the facts in working memory to determine which rules should fire. When the fact-list changes, the inference engine checks which rules are satisfied. A prioritized list of rules to fire is put into the agenda. Then, the inference engine executes the rules in the agenda.

  The inference engine provided by the CLIPS expert system uses the *forward chaining* control strategy, which is suitable for solving this type of problem. This strategy reasons from existing facts to resulting conclusions. Typically, forward chaining is more appropriate for monitoring and control applications [3, p.29].

- The *knowledge base* contains the rules governing how to make decisions. The criteria to manage the PRSD system, outlined in [2], were easily translated to CLIPS rules. Since the PRSD management philosophy is influenced by flight-specific and generic data such as the number of tanks flown and the capacity of each tank, fact templates which describe this information form the patterns of the rules. The most volatile patterns pertain to the mission phase and the quantity in each tank. Therefore, these patterns are located near the front of the rule [5, p.90]. The actions of these rules advise the analyst how to configure the PRSD system. A message is displayed to recommend which tank heaters to activate and which manifold valves to open. Simultaneously, the recommendations are written to a text file.

- In this application, the *working memory* or fact-list is divided into three categories of facts.

  First, facts which pertain to *static data* describe unchanging data. This data includes the number of tank sets used on this flight, the duration of the mission and physical specifications of the tanks (such as nominal capacities and volume).

  Secondly, there is *dynamic data.* These facts describe the changing characteristics of the tanks at certain times in the mission. An example of dynamic data is the quantities in each tank. This data will influence how the PRSD system is reconfigured at the appropriate decision time.

  Thirdly, *control facts* are used to divide this problem into phases. This type of fact detracts from the pure opportunistic nature of a rule-based expert system. However, in this problem, they are very useful for controlling the program execution since the problem can be subdivided into certain execution phases. That is, the expert system executes by firing one rule within a set of rules that pertains to the current execution phase.

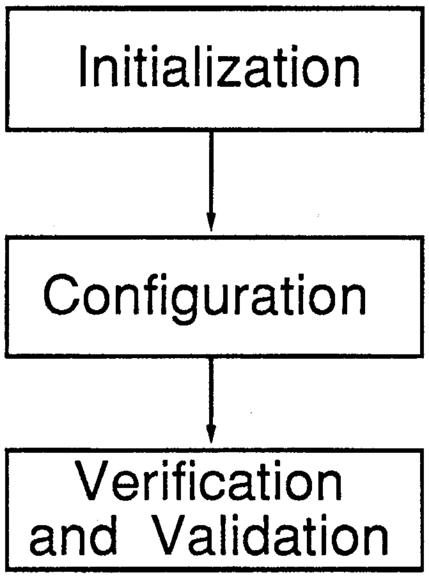


Figure 5: Steps of the PRSD system configuration problem.

## 4.2 Overall System Design

This problem can be divided into three sequential steps, as seen in Figure 5. Each step can be executed independently of the next. The expert system program for a particular step asserts the appropriate fact-file and loads the knowledge base for the next step to be solved. This observation was particularly useful in test and development. Also, since certain subproblems require many rules, it is advantageous to perform some rudimentary data-gathering and, then, load the next program into the expert system shell. This method eliminates the problem of running out of memory or slowing execution speed. Therefore, this expert system is divided into two programs, where the first program performs the data initialization step and the second program performs the remaining steps.

**Initialization Step** The primary objective of the first step is to get initial data from the analyst. This data will provide characteristics of the mission, such as the number of tank sets, any offload conditions, crew sleep shifts, times of other relevant activities and the duration of the mission. Also, generic data, which is not specific to any mission, must be determined. An example of this type of data is the physical specifications of the tanks. Based on this information, the expert system will know which management criteria to follow and when to query the analyst for dynamic information during the configuration step. A secondary objective of this step is to verify the analyst's inputs. For example, if there are two crew shifts, then at least four tank sets must be used.

**Configuration Step** The purpose of the configuration step is to manage the PRSD system according to the changing tank quantities and the mission profile. In the initialization step, the expert system determined the occurrence of the PRSD configuration times. So, in this step, the expert system asks for tank characteristics at these times and it recommends how to set the tank heater and manifold valve switches. The PRSD configuration rules are based on the standard PRSD management guidelines, explained in [2]. Also, this step checks to ensure that none of the tanks have depleted below its residual quantity.

**Verification and Validation Step** This step is used to check the ending quantities in the tanks. It is imperative that a certain amount of hydrogen and oxygen is reserved for mission extension days, the imprecision of the measurement gauges and other requirements. If the total quantities have depleted below this unusable quantity, this anomaly is reported.

## 5 Detailed Design Strategy

This expert system is composed of two CLIPS programs to eliminate the problem of overflowing memory or slowing execution, as discussed previously. The first program obtains the bulk of the data and verifies that the data is acceptable for the next program. The second program loads more data, evaluates the data at certain configuration times and suggests how to control the PRSD system. Lastly, the second program analyzes the final quantities in the tanks.

## 5.1 Program 1 – Load Data

The analyst will be asked to enter some flight-specific data. The characteristics of the Space Shuttle flight will influence the PRSD management, throughout all mission phases. This program will gather the following information.

- The *number of hydrogen and oxygen tanks* used on this mission must range from two to five tanks. The present Space Shuttle can load two, three, four or five tank sets, although an Extended Duration Orbiter (EDO) can hold up to nine tank sets. A general rule of thumb is that the number of days of a mission is approximately twice the number of tank sets, with no offloaded tanks.

- The analyst must enter *any offloaded amounts from each oxygen tank.* Several rules will be activated to check that the analyst entered valid offload quantities. The knowledge from the Offload Guidelines Section [4] was encoded in these rules.

- *The number of work shifts* is also needed. The Space Shuttle missions will either have a single or a dual work shift. In general, if there is a dual work shift, then there are at least four tank sets.

- The analyst must inform the expert system of *the number of flight days.*

- For each flight day and each work shift, the expert system will ask the analyst to enter the *start and end times of each sleep cycle.* These activities will trigger a reconfiguration of the PRSD system for single shift crews, in the next program.

- The expert system will also need to know when the following mission activities are scheduled: cryogenic usage start, post insertion checklist, ascent end, deorbit preparation checklist and cryogenic usage end. These activities will trigger *other cryogenic decision times.* Also, the analyst may enter the times at which any non-standard PRSD configuration activities occur.

The above information is saved in a fact file, which is accessed by the next program. Also, this fact file can be examined with any text editor. It provides a way to validate and document the expert system's inputs.

## 5.2 Program 2 – PRSD Management

This part of the expert system is implemented by the second CLIPS program. It consists of five general phases of execution.

1. First, the expert system queries the analyst for the names of some *input text files*. These text files contain generic tank specifications and mission extension information. The data from these text files is stored in working memory. Also, this phase loads the fact file, which was created by the first CLIPS program, into working memory.

2. The second phase *searches for the next cryogenic configuration time.* To examine each PRSD activity, this search procedure will use the search keys: integer hour, integer minute and integer second.

3. This phase consists of *interactive queries for changing tank quantities.* After these tank quantities are entered, the expert system checks that the quantity in each hydrogen and oxygen tank is greater than its residual quantity.

4. The next phase performs the actual *PRSD management.* The knowledge required to manage the cryogenic fuel tanks was based on the PRSD Quantity Management Section of [2]. The expert system informs the analyst which tank heaters to activate and which manifold valve to close. To determine the PRSD configuration, these rules examine the mission data and the changing conditions of the tank. This JSC document explicitly explains six cases of tank set arrangements:

   (a) Two tank sets.

   (b) Three tank sets.

   (c) Three tank sets, with offload conditions.

   (d) Four tank sets.

   (e) Four tank sets, with offload conditions.

   (f) Five tank sets.

   This phase was implemented by translating the guidelines in the EECOM Console Handbook [2] to CLIPS rules. The PRSD management of all tank set cases (as described above) is driven by certain metaknowledge, which aided in the design of this phase of the expert system. In other words, an understanding of the underlying reasoning behind the guidelines of each tank set case reveals a common philosophy for the entire system. The primary influence is the type of activity which triggers the PRSD configuration. For example, during all sleep periods of single shift crews, the configuration must isolate a manifold section. But, during a flight day, the manifold sections are not separated. This fundamental rule of isolating the manifold sections during night and not during day is followed by each tank set case. Also, another principle relevant to each case is influenced by the following events, liftoff and landing. That is, the management criteria requires that tanks one and two are enabled during liftoff and landing for each case.

5. Finally, the expert system performs a *validation of the ending conditions* of the tanks. The total quantity of the PRSD tanks must be greater than that reactant's total reserved quantity. This total reserved quantity ensures an adequate margin for mission contingencies (which may lengthen the mission duration), for inaccuracies of each tank's measurement gauge and for each tank's residual quantity. The expert system will inform the analyst if there is an adequate margin or not.

   If the aforementioned phases of the second program are successfully completed, then the PRSD management scheme will be saved in an output file. This output file will contain the recommendations of PRSD management at each configuration time. This file can be perused with any text editor to verify and validate the simulation.

## 6 Concluding Remarks

This expert system fulfills its intended purpose, which is to configure the PRSD system during all phases of various Space Shuttle missions. Moreover, this expert system will be flexible to changes and easy to maintain because it is written in a rule-based language, CLIPS. This rule-based system boasts the following advantages:

**Modularity** Because the rules are not interdependent and act opportunistically, it will be easy to add more rules. For example, it will be necessary to add more rules to configure the Extended Duration Orbiter's nine tank sets.

**Granularity** The existing rules break the problem domain into little pieces, so it will be easy to change existing rules. For example, if B heaters, instead of A heaters, should be used during the ascent phase of the mission, only two rules need to be modified.

**Expandability** This expert system could interface or link to other programs. It could be interfaced to a PRSD simulation program to eliminate the interactive queries for the initialization data at the beginning of the simulation and for the changing conditions of the tanks at the PRSD configuration times.

**Portability** This expert system can run on a variety of computers. That is, if the computer has a Kernighan and Ritchie C compiler, the CLIPS expert system shell can be installed. Then, any CLIPS program can be loaded and executed.

## References

[1] Space Shuttle Transportation System. Rockwell International Corp., Houston, 1984.

[2] C. Dingell and F. Ouellette. Standardized PRSD Quantity Management Plans. In *EECOM Console Handbook*, pages 4.2.5–1 thru 4.2.5–18. NASA JSC, Houston, 1988. JSC-19935.

[3] J.C. Giarratano and G. Riley. *Expert Systems: Principles and Programming.* PWS-KENT Publ. Co., Boston, 1989.

[4] C. Dingell. PRSD Offloading Guidelines and Considerations. In *EECOM Console Handbook*, pages 4.2.4–1 thru 4.2.4–4. NASA JSC, Houston, 1988. JSC-19935.

[5] J.C. Giarratano. *CLIPS User's Guide.* NASA JSC, Houston, 1989.

## B5 Session:
## Enhancements to CLIPS – General II

# Rule Groupings in Expert Systems

**Mala Mehrotra**
*Vigyan Inc.*
*MS 130, NASA Langley Research Center*
*Hampton, Va 23665.*
*mm@air12.larc.nasa.gov*

**Sally C. Johnson**
*MS 130, NASA Langley Research Center*
*Hampton, Va 23665.*
*scj@air12.larc.nasa.gov*

**Abstract**

Currently, expert system shells do not address software engineering issues for developing or maintaining expert systems. As a result, large expert systems tend to be incomprehensible, difficult to debug or modify, and almost impossible to verify or validate. Partitioning rule-based systems into rule groups which reflect the underlying subdomains of the problem should enhance the comprehensibility, maintainability, and reliability of expert-system software. In this paper, we investigate methods to semi-automatically structure a CLIPS rule base into groups of rules that carry related information. We discuss three different distance metrics for measuring the relatedness of rules and describe two clustering algorithms based on these distance metrics. The results of our experiment with three sample rule bases are also presented.

## 1 Introduction

Knowledge-based expert systems are playing an increasingly important role in NASA space and aircraft systems. They have potential usefulness in fault diagnostics and recovery, monitoring, planning, scheduling, and control systems, and are already being used as ground-based advisory systems. However, many of NASA's software applications are life- or mission-critical, and knowledge-based systems do not lend themselves to the traditional verification and validation techniques for highly reliable software[1].

Even relatively small rule-based systems contain hundreds of rules, and as the number of rules increases, the number of possible interactions between rules increases exponentially. There is also another dimension of complexity independent of the number of rules in the system. This is the number of potential matches for each pattern in a rule. As the complexity of these patterns increase, the number of possible combinations of facts required for testing these patterns increases exponentially. Therefore, it is infeasible to attempt exhaustive testing of every possible path through the system, much less every possible combination of inputs. Thus, exhaustive testing is impractical for demonstrating high reliability of knowledge-based systems, and less computationally intensive analysis techniques must be employed.

Although the meaning of each rule in isolation may be well understood, the complexity of the rule-based system stems from the interactions and inter-dependencies between rules. Therefore, analysis of the system should focus on understanding those interactions and assuring that they are correct. The approach we have chosen is to investigate the structuring of a rule-based system into a set of groups. Such groups would allow one to abstract away from the *each rule is a procedure call* point of view, and look at the system from a higher semantic level. This should make explicit the underlying subdomains of the problem and also aid in understanding the level of knowledge, (i.e. deep or shallow reasoning) that was applied to solve the subproblems. The groupings are made to aid in comprehension of the rule base and have no effect on the execution of the system.

Most developers already break up their rule bases into separate files to reflect the different run-time phases of their programs. However, there still exists a potential for each file to grow large and incomprehensible. For example, ONAV*[2], an expert system developed at NASA Johnson to help navigate re-entry of a space craft, has 564 rules divided across 13 files. The largest file contains 282 rules. The file decomposition reflects the various stages of navigating re-entry into earth's atmosphere of a spacecraft. Even with extensive comments and a tool such as CRSV†[3], the dependencies of rules across files cannot be easily determined. Our analyses indicate that grouping a rule base according to control aspects of the problem is not sufficient for understanding the problem. The static aspects of the problem can be understood only if domain knowledge can be separated from control knowledge[4, 5].

In this paper we describe our attempts to extract the domain knowledge from a rule base by structuring the rule base into groups of related rules. We present several distance metrics which measure the relatedness of rules, and we describe two different clustering algorithms.

## Related Work

The issues involved in grouping rules have been studied at various levels. Lindell[6] suggests a clustering algorithm based on keywords contained in comments. Clearly this can work only for well-documented rule bases and for those where one can succeed in finding appropriate keywords.

Lindenmayer, et al.[7] have proposed a certain methodology for grouping OPS5 rule bases for the Hubble Space Telescope Project. Different rule couplings are discussed with respect to *make*, *remove* and *modify* constructs in OPS5. However, dependency between two rules is assumed to exist only from the consequent of one rule to the antecedent of another.

A more sophisticated approach, by Jacob and Froscher[8, 9], utilizes more forms of data dependencies of a rule base and groups rules on their informational content. However, Jacob's grouping strategy also assumes that domain knowledge is in the form of a decision tree, in which chaining of rules takes place predominantly from the consequent of one rule to the antecedent of another. It presumes a hierarchical decomposition of domain knowledge, which seems to be well suited for classification problems, such as the mammal classification rule base. However, it does not work well in monitoring or scheduling problems, since here the antecedents of rules generally carry domain information, such as different modes of failure, while consequents usually give only directives for taking the appropriate actions.

## An Overview of Our Approach

We have taken a pattern-matching approach towards grouping of rules. In this approach, the commonality of items in the rules determines the distance between them. Our rule grouping process consists of three stages. First, the distance between each pair of rules is computed and stored in a distance matrix. In the second stage, the computed distances are modified so that all distances satisfy the triangle inequality. That is, we replace the distance between two rules by a shorter distance, if there exists an intermediate rule through which a shorter path can be created. Consider three rules $r_i$, $r_j$ and $r_k$ with inter-rule distances $d_{ij}$, $d_{ik}$ and $d_{jk}$. If $d_{ij} + d_{jk} < d_{ik}$, then we replace $d_{ik}$ by $d_{ij} + d_{jk}$. This method thus extracts the transitive dependency between rules. Finally, we apply a clustering algorithm to form our groups. There are two approaches we consider for clustering. The first one is an automatic clustering algorithm based on graph-partitioning approaches. The second requires the user to designate certain rules as "primary rules" or "seed rules" around which the clustering algorithm will form groups.

Automatic clustering algorithms work, but optimal clustering is NP-complete, and the resulting clusters may not conform, in any reasonable way, to the clusters a user would desire. Thus, we developed a more user-directed approach, in which the user designates some rules as primary rules, and a cluster is automatically formed around each of these rules. These rules typically reflect key concepts from the domain; thus, the resulting clusters correspond closely to what the user desires.

*Onboard Navigation Expert System

†CLIPS Cross Reference Style Analysis and Verification Tool

The rest of the paper is structured as follows. In the second section we discuss the design issues in developing different distance metrics. In section 3, the two different algorithms for clustering are given. Section 4 describes our experiments on three different rule bases. This section also discusses various quality measures for determining the "goodness" of groupings. Based on these, a performance evaluation is done of the different distance metrics on three sample rule bases. The final section presents our conclusions.

## 2 Design of Distance Metric

In order to allow our system to adapt to different types of rule bases, we have designed different distance metrics for measuring the relatedness of rules. Different metrics capture different kinds of information from the rule base. In this section, we present the motivation for our different distance metrics and the criteria used in designing them.

Experience with using Jacob and Froscher's grouping algorithm on rule-based systems, such as ONAV and MMU-FDIR[‡], suggests that different application domains require different distance metrics. Finding a single universal metric appears to be impossible, since different expert systems require one to capture different information.

In our discussions henceforth, we follow the terminology of Giarratano and Riley[11]. A rule base is made up of rules with antecedents and consequents. Each antecedent or consequent is composed of patterns which match facts in working memory at run time. Each pattern is further divided into fields or tokens, which in CLIPS can be a word, string or number. We refer to these as *items.*

Only certain items are relevant for grouping. We ignore all run-time aspects of the rule base, since the use of a particular item in a rule appears to be more important to proper grouping than the way in which it is used. Also, analysis of the run-time behavior of a rule base could be prohibitively expensive, amounting to direct simulation of all logic paths. Our grouping does not functionally alter the rule base at all; once the grouping is performed, the user sees rules in their original form. In effect, we are trying to automate a user's first steps in attempting to comprehend an unknown rule base.

In order to do effective grouping, we need to suppress all information conveyed by reserved words; they serve only as run-time directives. For example, a rule asserting a fact A, and another one testing for the absence of A, would not be temporally grouped together. But statically, since both rules are referring to the same information, their meaning may be more clear if they are present in the same group. If the fact A gets modified, then it is important to show that the validity of the rule testing for the absence of A still holds. A similar justification holds for ignoring keywords such as "retract", "bind", "test" and so forth. We do, however, include numeric and word type fields in our metrics, since they frequently contain domain-specific information relevant to static grouping. However, we suppress all strings and variable names. Strings rarely occur outside of print statements and convey little domain information. Variables are local to each rule, and hence cannot carry information when statically relating rules. Their bindings take place at run time, and therefore no values can be assigned to them *a priori.*

Given these fundamental design choices, we can now state a generic definition of the distance between two rules, $r_1$ and $r_2$, as

$$d(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ r_1\ and\ r_2}{no.\ of\ \text{``}common\text{''}\ items\ in\ r_1\ and\ r_2}$$

where different definitions of "common" give rise to different distance metrics. When there are no common items between $r_1$ and $r_2$, $d(r_1, r_2)$ is replaced by the maximum number of patterns allowed.

The nature of the domain knowledge enforces a certain programming methodology on the developer of a rule base. Classification systems, such as the mammal problem, have a hierarchical structure which yields easily to a data-flow grouping like Jacob's and Lindenmeyer's. Classifying disease hierarchies also

[‡]Manned Maneuvering Unit - Fault Detection, Isolation and Recovery System [10]

```
( defrule r1
        (animal gives milk)
  =>
        ( assert(animal is-a  mammal))   )

( defrule r2
        (animal is-a  mammal)
        (animal has hoofs)
  =>
        ( assert(animal is-a  ungulate))  )

( defrule r3
        (animal is-a  ungulate)
        (animal has blackstripes)
  =>
        ( assert(animal is-a  zebra))  )
```

Figure 1: Example Rules from Animal Rule Base

```
( defrule cea-a-gyro-input-pitch-pos-2
        (aah on)  (gyro on)
        (gyro movement pitch pos)
        (side a on) (side b on)
        (rhc roll none pitch none yaw none)
        (thc x none y none z none)
        (vda a ?m on)
  =>
        ( assert (failure cea))
        ( assert (suspect a))  )
```

Figure 2: Example Rules from MMU-FDIR Rule Base

falls in the same application type. The fundamental characteristic of such systems is that the flow of data takes place from the consequent of one rule to antecedent of other rules. Three rules from the mammal classification rule base are presented in Figure 1 to show the data-flow aspects of the system. For this type of rule base, it is appropriate to use a distance metric, $d_{df}(r_1, r_2)$, between two rules, $r_1$ and $r_2$, defined as,

$$d_{df}(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ consequent\ of\ r_1\ and\ antecedent\ of\ r_2}{no.\ of\ \text{``}common\text{''}\ items\ in\ consequent\ of\ r_1\ and\ antecedent\ of\ r_2}$$

A monitoring system issues different commands depending on the status of different components of the system being monitored. In such systems, the antecedents of the rules usually search for special values of flags in the component system, and the consequents assert actions to be taken when different components fail. Example rules from MMU-FDIR given in Figure 2 illustrate this point.

The bulk of domain information required for grouping is usually present in the antecedents of rules in a monitoring system. This gives rise to the antecedent distance metric:

```
( defrule starter_ok
        (starter cranks_engine yes)
        (lights dim slightly)
=>
        ( assert(battery problems no))
        ( assert(engine is_tight no))     )

( defrule battery_not_ok-
        (starter cranks_engine yes)
        (lights dim considerably)
=>
        ( assert(battery problems yes))
        ( assert(engine is_tight yes))    )

( defrule engine_misfires
        (battery problems no)
        (engine is_tight no)
        (engine misfires constantly)
=>
        ( assert(cylinder problems yes))  )
```

Figure 3: Example Rules from Car Diagnostics Rule Base

$$d_{ant}(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ antecedents\ of\ r_1\ and\ r_2}{no.\ of\ \text{“}common\text{”}\ items\ in\ antecedents\ of\ r_1\ and\ of\ r_2}$$

If one wanted to group on different component failures asserted by the consequent of such rules, a consequent distance metric could be defined as well. The antecedents in the above distance metric would be replaced by the consequents.

In a diagnostic system, a data-flow aspect is present together with a monitoring aspect. A hierarchical search space reflects the different sub-domains of the problem. Rule-interdependency is from consequent of some rules to antecedent of others. Certain other rules are related to each other through antecedents alone, since they detect similar symptoms from the domain. An example is given in Figure 3 where the antecedents of the first and second rules need to be fired by a situation match from working memory. Other rules in the rule base do not assert these patterns. However, there is data-flow dependency between the consequent of the first rule and antecedent of the third rule. Since such is the case for most diagnostic systems, both sides of the rule deserve consideration. Therefore, we have the following metric to define distance in such a case:

$$d_{all}(r_1, r_2) = \frac{Total\ no.\ of\ items\ in\ r_1\ and\ r_2}{no.\ of\ \text{“}common\text{”}\ items\ in\ r_1\ and\ r_2}$$

According to the distance metrics defined above, relatedness of two rules is inversely proportional to the distance between them. That is, the more related two rules are, the less the distance between them.

## 3 Clustering Algorithm

In order to be useful, a clustering algorithm must break up a rule base into meaningful groups. However, which particular clustering will prove "meaningful" is strongly dependent on the nature of the rule base. To address this issue, we perform clustering based on the different distance metrics. By carefully choosing the right metric, we hope to be able to achieve a clustering well suited to each particular rule base.

### 3.1 Automatic Clustering Algorithm

The automatic clustering algorithm starts with each rule in its own group. Groups are then merged based on the minimum inter-group distance. Here, we define inter-group distance, $D(i,j)$ as follows:

$$D(i,j) = \frac{\sum_{r_k \epsilon G_i} \sum_{r_l \epsilon G_j} d(r_k, r_l)}{n_i * n_j}$$

where $n_i$ and $n_j$ are the number of rules in groups $G_i$ and $G_j$, respectively.

Using this definition of inter-group distance, we form an automatic clustering algorithm as follows. In this algorithm, the user provides the total number of groups, $M$, to be formed, which serves as a stopping criterion. A high level view of the algorithm is given below:

```
Initialize each rule into its own group

While ($number\ of\ groups > M$)
  Find groups $G_i$ and $G_j$ with minimum inter-group distance $D(i,j)$
  Merge groups $G_i$ and $G_j$
```

We experimented with various other stopping criteria which did not need any user input. However, none of them seem to work consistently across rule bases. For example, we calculated the mean distance between the rules in the rule base, and when the intergroup distance exceeded the mean distance, the algorithm stopped clustering. A similar experiment was done taking the standard deviation of rules as the stopping criterion. In some instances, the rule base would fragment into very small meaningless groups; at other times, one would obtain a very skewed grouping with some very large groups and some very small ones. Since the purpose of our study is to ascertain which distance metric gives the right grouping for a rule base, penalizing the distance metric for not responding well to a stopping criteria did not seem justified. Hence, we chose the option of requiring the user to input the number of groups desired and then seeing which distance metric fares best.

As we shall discuss later, it is very difficult to define an optimal grouping for a rule base. Even if there were, implementation of a guaranteed optimal grouping solution for the rule base would reduce to an optimal graph partitioning problem, which is known to be NP-complete.

### 3.2 Clustering through Primary Rules Selection

In this strategy, more input by the user is required. Not only is the number of groups provided by the user, but for each group, a primary rule must be given which captures a concept from the rule base. These rules then form the seed or context around which clustering can take place.

A high level view of the clustering algorithm through primary rule selection is as follows:

```
For each rule $r_i$
  Find primary rule $r_p$ for which $d(r_i, r_p)$ is minimized
  Merge rule $r_i$ into G($r_p$)
```

In this algorithm, rules that are equidistant from two or more primary rules are skipped over in the first pass when all the other rules are clustered. In the second pass, the equidistant rules are resolved based on the criterion that they minimize the average distance between rules in the group that they are pulled into.

## 4 Experimental Results

In order to assess these different approaches to clustering, we tested our algorithms on three sample rule bases. The largest one is the MMU-FDIR, written by McDonnell Douglas Astronautics having 104 rules. Correct thruster configurations for the two sides of the MMU and gyroscope in the primary and backup modes are checked by 73 rules. An additional 14 rules deal with failure recovery for the two control electronics assemblies, and seven rules deal with tank/thruster tests. The remaining ten rules do printing and demonstration. These rules were removed before we began grouping, as mentioned in Section 2. The majority of the rules in this system have a similar structure. The antecedents consist of tests for automatic attitude hold, gyroscope readings, rotational and translational hand controller values, and valve drive amplifier outputs. The consequent then declares the faulty subsystems.

The second rule base is a car diagnostic expert system we wrote. It has 60 rules to diagnose problems in 12 subsystems of the car. Typical subsystems include the distributor, carburetor, and ignition systems. This rule base traverses a search tree which branches first at the root, based on the status of the head lights and whether or not the starter can crank the engine. The search is guided by various observations of ammeter readings, lights, spark-plug reversals and others. One of the problems in this rule base is that rules assert several potential problems. This feature greatly complicates grouping, but seems typical of fault diagnostic systems. Such complicated couplings mirror the complexity of the underlying domain.

Our last rule base is the animal classification program[12], a well-known problem in artificial intelligence. It contains 14 rules, which classify an unknown animal as either an ungulate, carnivore, or bird. Then, having placed a given animal into one of these broad categories, it makes a more precise classification, deciding, for example, whether an animal is a cheetah or a tiger, based on detailed characteristics.

### 4.1 Evaluation Criteria

It is difficult to judge the quality of a rule base grouping; judgement of quality is, in general, highly subjective. Nonetheless, it appears essential to assess the value of our grouping strategies in a comparatively rigorous manner. Otherwise, we would be unable to make definitive statements about the comparative merits of these strategies and of the value, more generally.

In order to measure the quality of groupings, we have developed two independent measures. The first of these measures is based on an "ideal grouping" which we generate ahead of time by hand. It measures the deviation of the computed groupings from our ideal grouping. The second of these measures the "stability" of groupings obtained by the primary rule algorithm, where we measure stability by counting the number of rules which migrate between groups as we vary our choice of primary rules.

While some attempt at measuring the quality of a grouping is necessary, there are difficulties with both the approaches here. In the first measure, there is obviously some subjectivity inherent in our choice of an "ideal grouping;" different researchers may produce different "ideal groupings." However, for rule bases as simple as those used here, people would tend to generate fairly similar "ideal" groups. The second measure of cluster quality avoids subjectivity, but still begs the question of the quality of a grouping; highly stable groupings can still be quite poor. In our view, finding a better way of assessing grouping quality should be as high a priority as that of finding better grouping strategies. Solving either problem would help solve the other.

## Measuring Cluster Quality

In many cases, it is easy to see which rules should be grouped together. However, in other cases rules cannot be classified unambiguously, since they relate to several key concepts. For example, in the rules given for the car diagnostic system in Figure 3, we can say that the second rule relates only to battery problems. However, in the first rule, a faulty battery relates to both starter and light problems; there is no best way to group this rule.

Mathematically, we use the following representation. The rule base consists of a set of rules

$$R = \{r_1 \ldots r_n\}$$

and a set of concepts

$$C = \{C_1 \ldots C_l\}.$$

Rule $r_i$ has associated concepts

$$Concepts(r_i) = \{C_{i_1} \ldots C_{i_k}\}$$

where

$$\{C_{i_1} \ldots C_{i_k}\} \subseteq C.$$

We define *unique* rules as those rules which contain only a single concept. Rules which have multiple concepts associated with them are called *ambiguous* rules.

To deal with the issue of *ambiguous* rules, we take as our ideal grouping a grouping of the *unique* rules only. This is easy to do, and removes some of the subjective component of this approach. Thus our ideal grouping has the form:

$$I = \{\mathcal{I}_1 \ldots \mathcal{I}_m\},$$

where each $\mathcal{I}_j$ is a disjoint set of unique rules.

Now suppose the observed or computed grouping is denoted as:

$$O = \{\mathcal{O}_1 \ldots \mathcal{O}_m\}$$

Our first step in measuring deviation is to pair each ideal group $\mathcal{I}_j$ with the "closest" observed group, where we measure closeness using only unique rules. Thus, for each index j, let $k_j$ be chosen to maximize:

$$\mid \mathcal{I}_j \cap \mathcal{O}_{k_j} \mid$$

Now we define the deviation $dev(A, B)$ for groups $A$ and $B$ as:

$$dev(A,B) = \{r \epsilon A \mid Concepts(r) \cap Concepts(B) = \phi\},$$

where the concepts in a set of rules are defined as the union of the concepts in the individual rules in the set. The total deviation of the observed groups can then be defined as,

$$tot\_dev = \sum_{i=1}^{m} \mid dev(\mathcal{O}_i, \bigcup_{j \mid k_j = i} \mathcal{I}_j) \mid$$

The average deviation is thus

$$ave\_dev = \frac{tot\_dev}{n},$$

where $n$ is the total number of rules.

### Measuring Stability of Groups

A group formed by the primary rule clustering algorithm is stable if the replacement of its primary rule by any of the non-primary rules in the group causes no migration of rules to other groups when we rerun the clustering algorithm. Thus, we can measure stability just by counting migration of rules between groups, as each non-primary rule in each group is in turn allowed to assume the role of primary rule for that group. To avoid a combinatorial explosion, we vary the primary rule in only one group at a time. Thus, measuring the stability of a grouping of an $n$-rule rule base requires only $O(n^3)$ time, since the primary rule grouping algorithm requires $O(n^2)$ time. Assume the set of primary rules are:

$$\pi = \{p_1 \dots p_m\}$$

where $p_i = r_{k_i}$. Let G($p_i$) be the group computed from primary rule $p_i$. Then we define the stability of a rule base grouping based on a choice of primary rules $\pi$ and distance metric $d$ as:

$$Stability(\pi,\, d) = 1 - \frac{\sum_{i=1}^{m} \frac{1}{|G(p_i)|-1} \sum_{q \epsilon (G(p_i) \setminus p_i)} |\, (G(p_i) \setminus G(q)) \,|}{n - m}$$

Here "\" denotes set differences. That is,

$$A \setminus B = \{a \mid a \in A \; and \; a \notin B\}$$

## 4.2 Analysis of Results

In this subsection we describe the performance of the various distance metrics in grouping the three rule bases using the two clustering algorithms.

### Performance of Automatic Rule Clustering

Even if the user is already familiar with the key concepts in a rule base, automatic grouping can provide much insight into the interactions between those concepts. The most interesting aspect of this strategy is the way in which groups form around related sets of concepts. For example, we expected that the rules for battery, starter and engine problems in the car rule base would form independent groups. However, since the physical coupling between these subsystems is reflected in the rule base, these three groups combine under the automatic grouping strategy to form a single large group. Perhaps a more careful choice of distance metric could fix such problems, but we currently have no cure.

The effectiveness of the different distance metrics for grouping our three sample rule bases is shown in Table 1 through Table 3. As expected, the $d_{ant}$ distance metric worked well for the car and MMU rule base, resulting in average deviation of only 5% from the ideal grouping. Thus, the bulk of the domain knowledge was indeed carried by the antecedents of the rules. The $d_{df}$ distance metric did not do nearly as well for grouping these diagnostic systems.

Because of the chaining nature of rules in the animal rule base, the $d_{ant}$ distance metric did not capture the rule interdependencies well and resulted in 29% deviation from the ideal grouping. However, as expected, the $d_{df}$ distance metric captured this chaining nature very well and resulted in a perfect grouping.

The automatic strategy has done well with the distance metric $d_{all}$ in all three cases. This appears to be a more general metric which works well in the absence of more detailed information on the rule base. Since the deviation between the figures obtained for automatic and primary rule strategies is not very different, the automatic strategy can be proposed as a viable alternative to the primary rule strategy when no selection of primary rules is given.

| Distance Metric | Average deviation for automatic clustering | Average deviation for primary clustering | Stability of primary clustering |
|---|---|---|---|
| $d_{all}$ | 0.05 | 0.04 | 0.96 |
| $d_{df}$ | 0.20 | 0.84 | 0.29 |
| $d_{ant}$ | 0.05 | 0.00 | 0.95 |

Table 1: Clustering of MMU FDIR rule base

| Distance Metric | Average deviation for automatic clustering | Average deviation for primary clustering | Stability of primary clustering |
|---|---|---|---|
| $d_{all}$ | 0.05 | 0.05 | 0.73 |
| $d_{df}$ | 0.17 | 0.25 | 0.54 |
| $d_{ant}$ | 0.05 | 0.05 | 0.77 |

Table 2: Clustering of Car Diagnostic rule base

| Distance Metric | Average deviation for automatic clustering | Average deviation for primary clustering | Stability of primary clustering |
|---|---|---|---|
| $d_{all}$ | 0.00 | 0.00 | 0.73 |
| $d_{df}$ | 0.00 | 0.07 | 0.55 |
| $d_{ant}$ | 0.29 | 0.21 | 0.82 |

Table 3: Clustering of Animal rule base

### Performance of Primary Rule Clustering

The primary rule strategy does as good as the automatic strategy in most cases. For our experiments, we did not change the primary rules for the different metrics, because we wanted to judge the metrics under the same conditions. Given the right distance metric and the right primary rules, the primary rule clustering gives a perfect grouping for both the animal and MMU rule base. For the MMU rule base, the perfect grouping was obtained through the $d_{ant}$ metric, and for the animal rule base, through the $d_{all}$ metric. For the car rule base both $d_{all}$ and $d_{ant}$ work well. The data-flow approach does not succeed here for the reasons given in section 2.

Stability figures for almost all the rule bases are consistent with the deviation figures, except in the case of the animal problem. Here our conclusions are limited due to the small set of rules. We suspect that patterns which are present with both carnivore and ungulate rules, such as dark-spots and black-stripes, interfere with the grouping process. In the MMU and car rule base, not only do we get meaningful groups but their stability figures are also consistently good. This means that there is little coupling between the groups produced, as evidenced also by examination of the rule base. Thruster configuration rules, failure recovery rules, and tank/thruster rules can be seen to be structurally different. The data flow metric fails here, because there is no chaining of rules. Each rule's antecedent creates a situation and the consequent asserts a diagnostic message. There is no relationship, therefore, between the diagnostic performed by one rule, and the situation created by another rule. The primary rule strategy is effective and robust. The price to be paid for this is the need for selecting primary rules.

## 5 Conclusions

In this paper we have described semi-automatic grouping of rule-based systems, using two different approaches. The first was completely automatic, while the second clustered rules around primary rules selected by the user. We also defined three different distance metrics for our grouping algorithms, each capturing different kinds of information in the rules. We tested our ideas on three sample rule bases, and attempted to cluster them using both clustering algorithms, and trying all three metrics for each algorithm.

Our experiments have shown that effective grouping can be achieved with both the automatic and primary rule clustering strategies. The primary rule approach works very well, if a reasonable set of primary rules can be easily identified. In the absence of a reasonable primary rule set, the automatic approach could serve as a useful fall-back strategy.

The choice of metric is also important. In the absence of specific knowledge about the nature of the rule base, a safe approach is to use the distance metric taking all patterns in each rule into account. However, one can clearly do better by using a metric tuned to the rule base, as our results have demonstrated.

There are a number of minor problems with our approach to grouping. For example, attributes irrelevant to the grouping process tend to distort the clustering of groups. In general, the effectiveness of our approach is fundamentally limited by its reliance on syntactic pattern-matching and cannot take into account the semantics of the patterns. As noted by Michalski and Stepp [13], "a configuration of objects forms a class only if it can be closely circumscribed by a conjunctive concept involving relations on selected object attributes." Moreover, a pattern-matching approach cannot capture the "Gestalt properties" of rule clusters. In other words, the meaning of the whole is generally greater than the sum of its parts. A more sophisticated approach is required. Our measurement of deviation of computed groups from ideal groups contains ideas which could lead to grouping strategies reflecting Michalski and Stepp's insight; we hope to pursue this in the future.

However, even with these limitations, the techniques we have developed seem effective at grouping rule bases, not only as judged by our stability measure, but also when judged in terms of the way one would intuitively group these rule bases. Therefore, we feel that these techniques could be very useful both in the validation phase and during maintenance of rule bases.

## Acknowledgments

We would like to thank Paul Miner of NASA, and Shahid Bokhari and John V. Rosendale of ICASE for their helpful suggestions.

## References

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

[2] Knowledge Requirements for the Onboard Navigation Console Expert/Trainer System. Technical Report JSC-22657, NASA, Lyndon B. Johnson Space Center, Houston, TX., September 1988.

[3] CLIPS Reference Manual. Technical Report JSC-22948, Artificial Intelligence Center, NASA, Lyndon B. Johnson Space Center, Houston, TX., September 1988.

[4] B. Chandrasekharan. Generic tasks in knowledge-based reasoning: High-level building blocks for expert systems design. *IEEE Expert*, Fall 1986.

[5] W.J. Clancey. The advantages of abstract control knowledge in expert system design. In *Proceedings, National Conference on Artificial Intelligence*, pages 74–78, 1983.

[6] S. Lindell. Keyword cluster algorithm for expert system rule bases. Technical Report SD-TR-87-36, The Aerospace Corporation, El Segundo, CA., June 1987.

[7] K. Lindenmayer, S. Vick, and D. Rosenthal. Maintaining an expert system for the Hubble space telescope ground support. In *Proceedings, Goddard Conference on Space Applications of Artificial Intelligence and Robotics*, pages 1–13, May 1987.

[8] R. J. K. Jacob and J. N. Froscher. Developing a software engineering methodology for knowledge-based systems. Technical Report 9019, Naval Research Laboratory, Washington, D.C., December 1986.

[9] R. J. K. Jacob and J. N. Froscher. A software engineering methodology for rule-based systems. *IEEE Transactions on Knowledge and Data Engineering*, 1990, in press.

[10] D. G. Lawler and L. J. F. Williams. MMU FDIR Automation Task. Technical Report NAS9-17650, McDonnell Douglas Astronautics - Engineering Services, Houston, TX., February 1988.

[11] J. Giarratano and G. Riley. *Expert Systems Principles and Programming.* PWS-KENT Publishing Company, 1989.

[12] P. H. Winston. *Artificial Intelligence.* Addison Wesley Publishing Company, 1979.

[13] J. G. Carbonell R. S. Michalski and T. M. Mitchell. *Machine Learning - An artificial Intelligence Approach.* Tioga Publishing Company.

# Automatically structuring and translating CLIPS

James R. Geissman
Abacus Programming Corporation
Van Nuys, CA 91411

## Introduction

Expert systems become more valuable when they can be integrated into production environments. Code that is written for easy maintainability and that can be ported to different machines is code that can be easily integrated. This paper discusses two ports of CLIPS, implemented using the Metapack™ translation system developed by Abacus Programming Corporation.

One translation converts positional form CLIPS into "structured" or "template" CLIPS, a form that is more easily maintained. The other translation converts either form of CLIPS into OPS83, a semantically similar forward-chaining language that has somewhat higher performance resulting from its being compiled.

## Porting and Translating Expert Systems

Many expert system languages and shells implemenmt similar problem-solving models with equivalent semantics, and the forward-chaining production system is a good example. This model is implemented by CLIPS, OPS5, OPS83 and ART, to name a few widely available languages. Other, more elaborate systems such as Aion may offer a variety of models, but can also be translated because they include forward-chaining among their capabilities and have a text file knowledge representation that is used for transporting knowledge bases between machines.

## Translating CLIPS into OPS83

OPS83 is a forward-chaining production system language like CLIPS. Some users wish to translate their systems from CLIPS to take advantage of the increased speed available in OPS83. The input to the translator is (a) templates describing the fact types to be used and (b) CLIPS source code. The input code may be positional (with either defrelations or deftemplates) or in the structured form based on templates. The output is OPS83. Programs of the complexity of the CLIPS monkey-and-bananas example program can be completely translated. Some program elements such as user-defined functions and multiple-value fields are not translated, and comments in the output indicate where hand translation must take place to finish it off.

The translation operates in two passes, and requires that the input be syntactically correct (e.g., the code loads into CLIPS or passes CRSV). Most of the code is parsed in the first pass and converted into a more manageable form; some that cannot be converted into OPS83 in the present state of the translator is turned into comments and passed through as-is with warnings. (The translator can readily be extended when it encounters programs that contain large numbers of elements previously considered of low priority.) The second pass generates equivalent OPS83 for most CLIPS features and identifies areas where additional hand modifications may be required. For example, the following CLIPS left-hand-side conditions:

```
?fact1-id <- (fact1-type ?value ? ? ?color)
?fact2-id <- (fact2-type ?value ? ?color ?)
```

are converted into this OPS83 (assuming the field names are defined as shown):

```
&fact1_id  (fact1_type);
&fact2_id  (fact2_type
                     field1 = &fact1_id.field1;
                     field4 = &fact1_id.field3;);
```

OPS83 requires fact-ids more frequently than CLIPS, which uses bound variables local to the rules to connect patterns and only uses fact-ids to identify facts to be retracted or modified. (Compare the CLIPS and OPS83 in the above example—the variables ?value and ?color appear in both CLIPS facts, while the OPS83 makes the references by using &fact1_id.*fieldname*) Where the CLIPS facts to not have fact-id values, the translator automatically assigns the ids &1001, &1002, etc..

## Structuring Positional CLIPS

CLIPS facts can be written in two styles. The *positional* style is of the following form:

```
(fact-type field1-value field2-value field3-value)
```

where the field-value terms are variables or constants as required. Because the fields are positional, anonomous variables or wildcards are sometimes required to keep the fields aligned.

The *structured* or *template* style is of the following form. First, a template is defined:

```
(deftemplate fact-type
          (field field1-name (attribute value)...)
          ...
)
```

Then, field values within facts are identified by reference to the field names:

```
(fact-type (field-name field-value) (field-name field-value)...)
```

The structured form has some real advantages over the positional. First, the code is more readable, because the name of each attribute is given and does not have to be guessed from the variable names. Second, the code can be modified without getting fields out of alignment. (For example, juggling field order to improve performance can be done by rerranging the template without changing other code.) Third, the basic fact structures can be redefined by adding or deleting fields without having to change existing code (except to remove references to deleted fields).

The translation operates in one pass, and requires that the input be syntactically correct (e.g., the code loads into CLIPS or passes CRSV) including either deftempate or defrelation statements defining all the fact types. Most of the code except facts (left- or right-hand side) is passed through unchanged; templates are parsed and the field names are stored om a table, and defrelation statements are converted to deftemplates. Fields identified in defrelations are assigned the names field-001, field-002, etc., unique over the set of facts; the user can later change these with a text editor if something more recognizable is desired.

When a fact is encountered, the field names are retrieved from the table and added to the output, while wildcard (place-holder) variables are skipped. One required assumption is that the order of fields in the template is the same as their order in the facts. For example, the following left-hand-side condition:

```
?fact-id <- (fact-type ?value ? ? ?color)
```

is converted into this output:

```
?fact-id <- (fact-type
                       (field1 ?value)
                       (field4 ?color)
)
```

## The Metapack translation system

Metapack is a translator-writing package developed by Abacus Programming Corporation that can be used to translate any regular language or data, such as a computer program from one language or dialect to another, generic text to specific text, or data from one database or format to another. It can also be used for related purposes, such as filtering programs to check for standards, gathering statistical data about text or programs, etc.

The user writes a program containing syntax descriptions of input forms in Backus-Naur form (BNF), and rules that describe what output is to be generated or other actions performed when specific forms are encountered in the input. If the first rule does not match the input, Metapack automatically backs up and tries alternatives. The actions that it can perform include outputting either copies of the inputs or other data, collecting information in stacks, arrays and variables that can be of whatever data type is required. Metapack is thus similar to Prolog with the difference that reading input and matching it against the rules is inherent and does not have to be programmed explicitly.

## A6 Session:
## Space Shuttle and Real-time Applications

# INTERPRETATION OF SPACE SHUTTLE TELEMETRY

Donald R. Culp

Rockwell Space Operations Company
Lyndon B. Johnson Space Center
Houston, Texas U.S.A.

The real-time interpretation of Space Shuttle telemetry by flight controllers presents a formidable manual task. Currently, a flight controller scans binary and numerical data, searching or familiar normal patterns and system behavior. A suspect signature is analyzed by the flight controller in order to determine its origin. Mission replanning is undertaken for a degraded system. The Mission Operations Directorate (MOD) is creating a Real-Time Data System (RTDS) for systems monitoring. The Payload Deployment and Retrieval System (PDRS) Decision Support System will be a part of RTDS, devoted to Remote Manipulator System (RMS) automated monitoring.

This paper describes the PDRS Failure Analysis and Diagnosis problem, our planned use of an expert system employing the C Language Integrated Production System (CLIPS) language to solve it, and our progress to date. In addition, it will address the importance of the overall design within the context of automation in the RTDS architecture, the methodology used, and the validation approach taken. The Decision Support System will evolve as a modularized rule-based system programmed in the CLIPS expert system language. Each of five modules represents a subsystem of the RMS and includes the positioning and retention hardware (MPM/MRL), the Display & Control panel (D&C), the Arm Based Electronics (ABE), the End Effector (EE), and the Orbiter to RMS data interface unit (MCIU). The MPM/MRL subsystem was chosen as the target of a prototype development and its knowledge base is completed. Testing and validation of the software is conducted by experienced mission controllers. The initial design stages for the D&C panel module are beginning.

The automation of RTDS frees humans of the repetitive nature of telemetry interpretation by providing recognition of nominal and off-nominal signatures and subsequently offers already rationalized workarounds as potential real-time solutions to a real-time problem. The CLIPS language provides all necessary programming constructs to meet the early goals of the proposed solution for this diagnostic problem. The capabilities of present expert systems support this application. The PDRS expert system should prove valuable in areas beyond its initial requirements such as training, design, and the preservation of corporate knowledge.

## BACKGROUND

The present method of monitoring Space Shuttle telemetry differs little from the monitoring methods of earlier manned spacecraft operations. Data displays showing only the most basic and necessary processing of telemetry are constantly scanned by flight controllers. A flight controller's primary function is recognizing data patterns indicating nominal operations, pre-defined anomalous behavior, or unknown scenarios. This function is executed through the memorization and recall of explicit patterns and, without an improvement in the representation of information, any increase in workload reduces pattern recognition accuracy and efficiency.

The increased complexity of spacecraft systems and activities related to the Shuttle program has led to an increased workload for the flight controller without a corresponding improvement in the management of information. Currently this information, or telemetry, of Shuttle systems is displayed to the flight controller as a number of monochrome screens blanketed with binary or numerical data separated by abbreviations of text. The hieroglyphic nature of this method limits data interpretation to an experienced flight controller of a particular discipline, with 18 disciplines sharing the Shuttle monitoring task. Even trained flight controllers suffer from this environment, experiencing vision fatigue, mental fatigue, and boredom. The Real-Time Data System (RTDS) is an approach to remove or alleviate the majority of these negative aspects associated with the monitoring of Shuttle telemetry. The RTDS implementation of workstations with powerful graphics and advanced software will support the flight controller in the performance of specific monitoring tasks in addition to increasing the time available for procedures replanning, a task that is not easily automated and competes for the flight controller's attention at critical times.

The current environment of Remote Manipulator System (RMS) monitoring consists of two monochrome screens, two strip chart recorders, and several dozen event lights. A collection of printed materials is used including operational schematics, flight rules, physical system data and malfunction procedures. Recent certification of RMS RTDS software has begun a redefinition of the flight controller's tasks. Nevertheless, the RMS console arrangement still remains very similar to the previously described environment which existed during the earliest phases of manned flight.

## RTDS, DESSY AND PROTOTYPE ASSESSMENT

The RMS section's RTDS intiative consists of three separate projects. The Temperature Mode Monitor is an algorithmic software module intended to relieve RMS flight controllers from data scanning when the RMS is not in use,in what is called its temperature mode. The RMS Position Monitor displays a graphical representation of the RMS configuration during operations. Both of these applications are operated during flight. Finally, the Decision Support System (DESSY) will monitor RMS operations real-time in order to detect and analyze RMS failures.

In response to management encouragement for increased automation of systems monitoring in anticipation of Space Station requirements, the RMS discipline identified a specific need corresponding to the goals of RTDS. This need originates from a mandate to maintain at least four certified flight control teams of three flight controllers. Due to the limited number of Shuttle missions which carry an RMS, the

RMS 'specialty' section receives a lower training priority than disciplines which are required for each flight. Combined with the low amount of actual flight experience available to the teams, the RMS flight controller certification process requires more time and simulations than desirable. These simulations are expensive large-scale exercises in role-playing involving hundreds of people. The circumstances of this training and retraining issue creates a unique opportunity for an expert system which might both train and support the flight controller in the execution of low-level telemetry analysis and fault diagnosis.

The DESSY expert system project is primarily expected to provide real-time analysis and diagnostic capability in support of flight simulations and operations. This project is divided into two phases: phase I will use a rule-based approach, and phase II will augment the rule-based approach with model-based capabilities. The requirements analysis of phase I includes rule-system prototype development, validation, and testing; system architecture studies and scoping, user interface studies, initial use of the CONFIG tool for model-based support of rule-system prototyping activities, and use of Rule Checker, a software tool for verification of consistency and completeness of rule sets. The exploitation of capabilities for training and the validation of the Failure Modes and Effects Analysis (FMEA) and Malfunction Procedures (MALF) is still in the future.

DESSY is a modularized rule-based expert system consisting of five distinct developmental modules. Each module represents one of five defined subsystems within the RMS (see Figure 1) : Display & Control panel (D&C), Manipulator Positioning Mechanism/Manipulator Retention Latches (MPM/MRL), End Effector (EE), Arm-Based Electronics (ABE), and the Manipulator Controller Interface Unit (MCIU).

The MPM/MRL subsystem was selected for development of the initial prototype rule set. Certain characteristics of the subsystem appeal to its prototype role including the expert's level of expertise of the system, the level of instrumentation of the system, the inclusion of both mechanical and electrical devices, the component to failure ratio, the I/O requirement, independence from other RMS subsystems and a robustness that tolerates failures (see Figure 2). Other subsystems of the RMS share a common interface, an interface which in turn doubles as an RMS subsystem - the MCIU. Involvement of the MCIU is undesirable for prototyping purposes because of system complexity resulting from the nature of its interface.

The readily available option of using CLIPS 4.2 on a personal computer influenced early considerations of hardware and software requirements for the developmental stage. The portability of CLIPS code to an executable C file is of great importance since the expert system is to operate in a MASSCOMP computer environment and use an interface written in the C language. However, RTDS is developing software with the GENSYM expert system shell 'G2' and may seek to standardize the language of all MOD applications in the near future. A copy of G2 is now available to the DESSY developers. A future hardware upgrade is expected to provide the RMS console with a DECStation 3100 or similar workstation.

## MPM/MRL RULES

Downlinked Shuttle telemetry is the lowest level of system knowledge. This telemetry is composed of current and voltage values, switch scans, and microswitch indications. At the present time, all other knowledge of the system is deduced from this data. Future applications may include a user query function to identify the physical position of switches, talkbacks, and circuit breakers along with other 'astronaut' observed information. The importance of this user-input information fades with the understanding of its undependable availability. Astronauts are not asked for this information unless it is requested in direct support of failure diagnosis. The obvious advantage of user input would be in a training or off-line diagnostic effort.

A. Syntax

CLIPS' lack of restriction on knowledge representation is a two-edged sword. While allowing the developer to choose his own representation scheme, it also allows for misrepresentation of knowledge and creation of programs with poor structure. For beginning AI programmers, this liberty is confusing. Original attempts at representing the MPM/MRL knowledge used facts with as many fields as desired. This led to complex combinations of fields which differed in number from fact to fact. Our desire to use the Rule Checker tool encouraged consideration of an object-attribute-variable (O-A-V) fact representation. Ironically, it later became apparent that all knowledge in the subsystem is best represented by this classical form.

A review of the available knowledge suggested that the O-A-V fact representation is natural and is compatible with the results of analysis. This analysis is found in the Knowledge Acquisition section of the paper. The standardization of fact format is especially important in the early stages of rule development since the number or purpose of fields may affect the efficiency and compatibility of rules in an area such as search. The only restriction created by this typical triple field O-A-V representation is the lack of generality in object names. This restriction does not allow the possibility of classes of objects. Classes of objects can be easily defined in CLIPS by assigning more than one field to the object name.

B. Information

The MPM/MRL system has several discrete operational states thus allowing most rules to be associated within a context of state. This reduces the chance of an inappropriate rule being placed on the agenda as well as adding knowledge of the operating system to the rule set. A state can exist for each independent system. The MPM's, MRL's, and Ready-For-Latch's (RFL) each possess a state. Some possible states are deployed, in-transit, and stowed for the MPM's; latched, in-transit, and released for the MRL's; Ready-For-Latch and Not-Ready-For-Latch for the RFL's.

A status fact is used to represent the health of a component or system. Components are often defined with respect to their pedestal location. For example, the MRL located on the Mid pedestal is viewed as a single component which can have status values of nominal, single-motor-time, and failed, in addition to some less specific values.

Rules are written in order to identify every possible telemetry signature and assign a meaning to this state. Since rules are controlled based on the current state of the system, many variations in telemetry may pass undetected because of their inability to be placed on the agenda. This control philosophy is permissible when considering the mechanical nature of the MPM/MRL subsystem which limits unexpected eventualities but must be reconsidered in future DESSY software. Certain rules identify combinations of different states as valid but undesirable system activity.

These rules are usually based on violations of the Flight Rules. Flight Rules are restricitions against Shuttle activities that threaten crew or mission success. An example is an RMS operator's attempt to stow the arm without RFL indications.

There are no rules which make use of the FMEA documentation for failure analysis. Such rules would identify candidate failed components or reduce the size of ambiguity groups and will require inputs not presently made available to the rule set such as strip chart information. This information will be included in later versions of the rule set.

C. Organization

After the completion of a significant number of rules a pattern evolved which served as a template for the majority of rules in the rule set. This template included the following conditions : 1) a state fact, 2) a timing fact, and 3) telemetry facts. The following actions were common in rules based on this template : 1) state assertions, 2) status assertions, and 3) output. An example rule is shown in Figure 3.

A rule hierarchy takes advantage of the physical and operational realities in the system. This is illustrated in Figure 4. Rules at each level are restricted to a reasoning process which activates a rule at the same or lower level. The final state is the result of a data-driven forward chaining process through a restricted search space.

The MPM/MRL rule set consists of approximately 90 rules : 70 MRL rules and 20 MPM rules. A 60% reduction in the number of rules could be achieved by considering classes of objects. Nevertheless, this amount of repetition is not likely to be seen in rule sets of other RMS subsystems, so a new representation of knowledge would offer only limited gains in rule set efficiency. No accurate extrapolation can be made between the number of rules in the MPM/MRL rule set and an expected number of rules in other RMS rule sets.

The architecture of the DESSY software is shown in Figure 5. The LOADER module loads all programs necessary for the expert system's operation. The WORLD program initializes the knowledge by identifying the system's initial or default state. The RULE program contains the diagnostic rules of the system or subsystem involved. The TEST-IN program loads one second of data after all rules placed on the agenda have been fired. The TEST-DATA program supplies the test data or simulated telemetry. The USER-INPUT program allows the user to augment or alter the knowledge base after the initialization routine or before the next activation of the TEST-IN program.

## KNOWLEDGE ACQUISITION, METHODOLOGY, & TOOLS

The integration of various sources of knowledge and their relationship to the knowledge base is illustrated in Figure 6. The central importance of the expert is made clear by recognizing his participation in the development of system publications (e.g. MALF, FMEA, etc.) in addition to his current role in the expert system. Advice on system architecture and methodology is provided by the Lockheed Engineering, Management, & Support Company's Artificial Intelligence Section. Support for rule set development is being provided by the Intelligent Systems Branch of the Engineering Directorate. This support includes the use of two important software packages : Rule Checker and CONFIG.

## A. Knowledge Acquisition & Methodolgy

An important part of any prototype expert system is the acquisition of knowledge. Identification and availability of the human experts is the first step in this process. This requires assigning key employees to a long-term high payoff project - a difficult sacrifice for any organization. Unfortunately, this activity fell victim to a total misunderstanding of the knowledge gathering methodology. In an ideal world, all flight controllers are to be equally knowledgeable of the entire RMS system thus allowing the knowledge engineer and expert to be the same person if the knowledge engineer is to be a user. However, successful implementations of AI have been based on reality and in the real world levels of knowledge vary greatly from one person to the next when the domain is sufficiently specific. In addition, this type of singular activity cannot be expected to produce an abundance of strategic heuristics - heuristics being the heart of an expert system. Therefore, the first error was the lack of a clear distinction, or any distinction, between the interviewer and interviewee. Secondly, the project was inappropriately viewed as an opportunity to create a system expert. This resulted in a situation where an employee new to the RMS and the field of AI is assigned to be knowledge engineer and expert. Extremely optimistic timelines were a natural consequence of this.

A second step in the knowledge acquisition process is the location of all known published information concerning the system of interest. This included the RMS Mission Histories, Failure Modes and Effects Analysis, Malfunction Procedures, Shuttle Operational Data Book, Console Handbook, system schematics, and documented efforts in the development of limit sensing algorithms. Reorganizing the collected information into different formats often aided in the task of discovering relationships between two pieces of knowledge. For example, the FMEA's were diagrammed into a matrix that plotted failed components versus the system effect. This type of information quickly established which failures could be identified with certainty. State-transition diagrams (see Figure 7) defined the operational limitations of the system. These diagrams were deduced from physical system data, operational modes, and flight rules.

Testing of the rules in the PC environment has been accomplished by creating data files which represent actual telemetry. The total number of tested cases is 24. These cases are representative of a broad range of possible failures including transients, motor failures and microswitch failures. These data files are constructed so that individual files can be combined in different orders to represent a great number of failure scenarios.

## B. Tools

Rule Checker performs a formal analysis of consistency and completeness for a fault management rule set. This analysis is used for rule set development and for validation of diagnostic software. A pair of rules is labeled ambiguous or redundant; the former if both rules fire but have different actions and the latter if both rules fire and have the same actions. Ambiguity is permissible if both rules should fire when given a particular signature. In one case 45 comparisons were made of 10 rules resulting in 19 ambiguous cases and 4 redundant cases. Redundant cases result from both parts of an 'OR' statement proving true. The majority of ambiguous cases may be resolved through the addition of explicit state information regarding the RFL's and this work is now in progress. The RFL's had previously been treated as a state value for the MRL's even though the RFL's always possess a defined state. This treatment of the RFL's was acceptable based on the state-transition diagram and it successfully operated during rule validation simulations but failed the logical tests of Rule Checker. This is an example of how certain knowledge can be considered less important by the expert because of its contextual insignificance. Without proper guidance, this user development approach will result in a poorly designed knowledge base.

CONFIG is a modeling and simulation tool prototype for analyzing the normal and faulty qualitative behaviors of engineered systems. Qualitative component models are defined in terms of normal and faulty modes and processes, which are defined by invocation statements and effect statements with time delays. System models are constructed graphically by using instances of components and relations from object-oriented hierarchical model libraries. System failure syndromes are simulated using a modified form of discrete event simulation. CONFIG models of the MPM/MRL are completed and their capabilities for supporting analysis of system fault detection and diagnosis have been demonstrated. The present models will be updated to achieve consistency with rule objects and attributes that were used for fault diagnosis.

## FUTURE WORK

Design assistance in the development of human-computer interface concepts is provided by the MITRE Corporation. Development of phase I RMS expert system interface requirements is the current focus of this work. Early display proposals from the RMS user community suggest a total of five window displays for the expert system - one display for each rule set module. These displays would be part of a larger network of displays belonging to other RMS RTDS applications. Since the flight controller must continually be aware of the entire system's performance, the display sharing and hierarchy issue is of particular importance to the user. Although this issue has received considerable attention, no complete design concept has been evaluated.

Disregarding the lack of a detailed interface design, much progress has been made in the understanding of the design problem and the development of prototype concepts. An illustrative early screen design for the MPM/MRL expert system is shown in Figure 8. Compared to current display capabilities (see Figures 9), one immediately appreciates the promises of RTDS. No longer is it necessary to match crowded numbers to mental patterns. The clear and concise display of information is made more meaningful for the less experienced or nonexperienced through the addition of graphics, color, and limited animation.

Long term goals include completion of phase II goals, integration of the model-based software with the rule-based software and investigation of the G2 expert system shell. G2 offers significant advantages over CLIPS in display building abilities, interfaces to simulation data and real-time flight data, and real-time syntax checking.

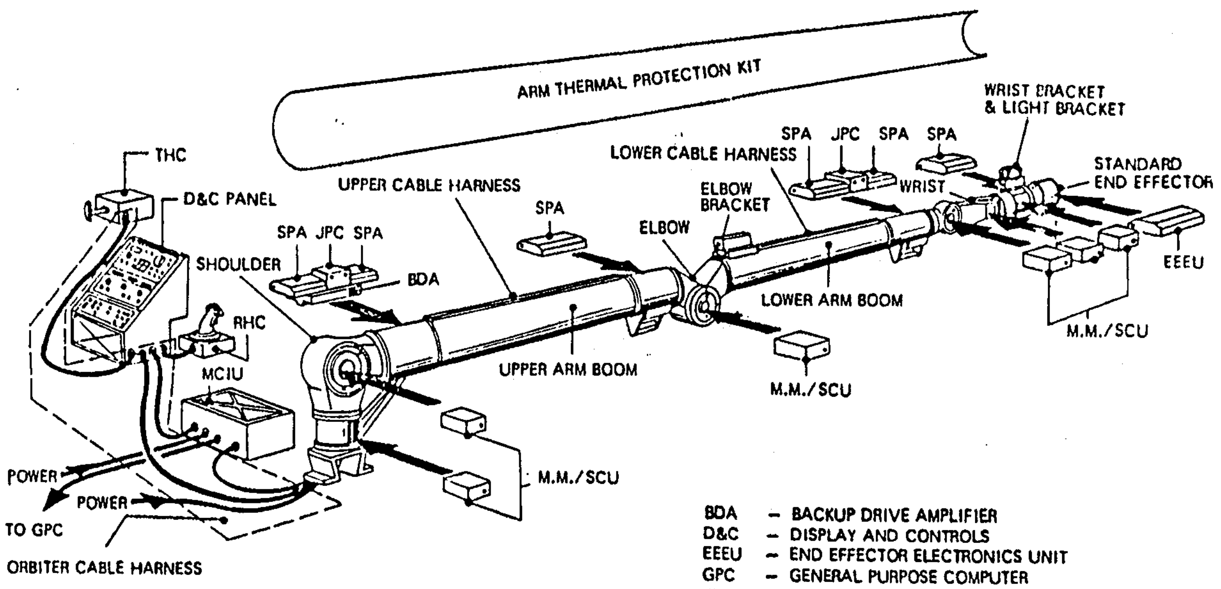
Only the earliest steps have been taken toward certification of project products. Final certification for console support will await the completion of the DESSY project. Preliminary steps toward certification include the use of Rule Checker, development of CONFIG models, and the testing of MPM/MRL rules with test case data. Approximately 20 test cases representing both nominal and likely off-nominal telemetry signatures have been tested with the rules performing as expected. Final certification requires software to be transferred to MASSCOMP computers and serve in support of simulations.

## REFERENCES

[Cragun, et al., 1987] Cragun, Brian J., and Steudel, Harold J., Decision-Table-Based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems. Int. J. Man-Machine Studies, 1987.

[Malin, et al., 1990] Malin, J., Basham, B., and Harris, R., Use of Qualitative Models in Discrete Event Simulation to Analyze Malfunctions in processing Systems. In Artificial Intelligence in Process Engineering (M. Mavrovouniotis, ed.), Academic Press, 1990.

[Muratore, et al., 1988] Muratore, J.F., Real Time Expert System Prototype for Shuttle Mission Control. Proc. 2nd Annual Workshop on Space Operations Automation and Robotics, NASA Conference Publication, 1988.



| | | |
|---|---|---|
| BDA | – | BACKUP DRIVE AMPLIFIER |
| D&C | – | DISPLAY AND CONTROLS |
| EEEU | – | END EFFECTOR ELECTRONICS UNIT |
| GPC | – | GENERAL PURPOSE COMPUTER |
| JPC | – | JOINT POWER CONDITIONER |
| MCIU | – | MANIPULATOR CONTROLLER INTERFACE UNIT |
| MM/SCU | – | MOTOR MODULE/SIGNAL CONDITIONING UNIT |
| RHC | – | ROTATIONAL HAND CONTROLLER |
| SPA | – | SERVO POWER AMPLIFIER |
| THC | – | TRANSLATIONAL HAND CONTROLLER |

Figure 1 - RMS components.

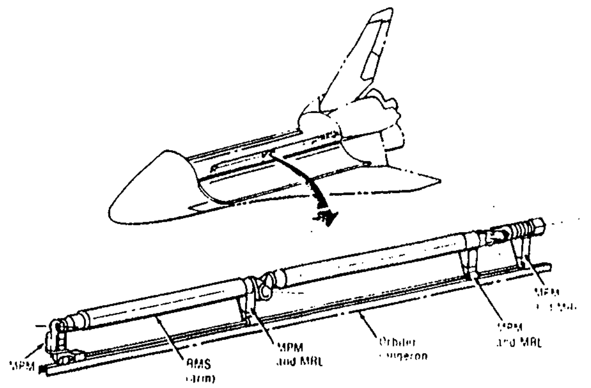


Figure 1a - MPM/MRL Locations

| MPM MRL JETT | EE | MCIU | D&C | HTR | ABE EEEU | |
|---|---|---|---|---|---|---|
| 71 | 39 | 88 | 145 | 9 | 114 | NUMBER OF FAILURES |
| 28 | N/A | 8 | 27 | 56 | 23 | NUMBER OF MAJOR FUNCTIONAL COMPONENTS |
| 10(18) | N/A | 13 | 8(32) | 2 | 2 | INPUTS (INCLUDES SWITCHES) |
| 75 | 6 | 80 | 43 | 13 | 66 | OUTPUTS |
| 0 | N/A | 3 | 1 | 1 | 1 | NUMBER OF OTHER INTERFACES |

Figure 2 - Subsystem Assessment for Prototype

```
(defrule  PDRS1129-mrl-forward-latch-commanded  "R-F-L"
     ;;context - applicable only when mrl is r-f-l
          ?state <- (mrl-forward state r-f-l)
     ;;current time
          (current-values pseudo-time ?time)
     ;;used to prevent multiple firings while the mrl state is
      unchanged
          (not(mrl-port-forward cmd latch)
     ;;command indicators
           (mrl-port-forward-mca2-latch  status  0)
           (mrl-port-forward-mca4-latch  status  0)
     ;;mark old facts to be updated
          ?time  <-  (current-values  time-forward-latch-cmd-init  ?)
         ?status  <-  (mrl-port-forward  status  ?)
 =>
     ;;printout conclusion of the rule
       (printout t t "MRL Forward Latch Commanded at " ?time)
     ;;update timer and indicate command
         (retract ?time ?state ?status)
          (assert  (current-values  time-forward-latch-cmd-init  ?time)
                   (mrl-port-forward state latch-cmd)
                   (mrl-port-forward status nominal)))
```

Figure 3 - Example Rule

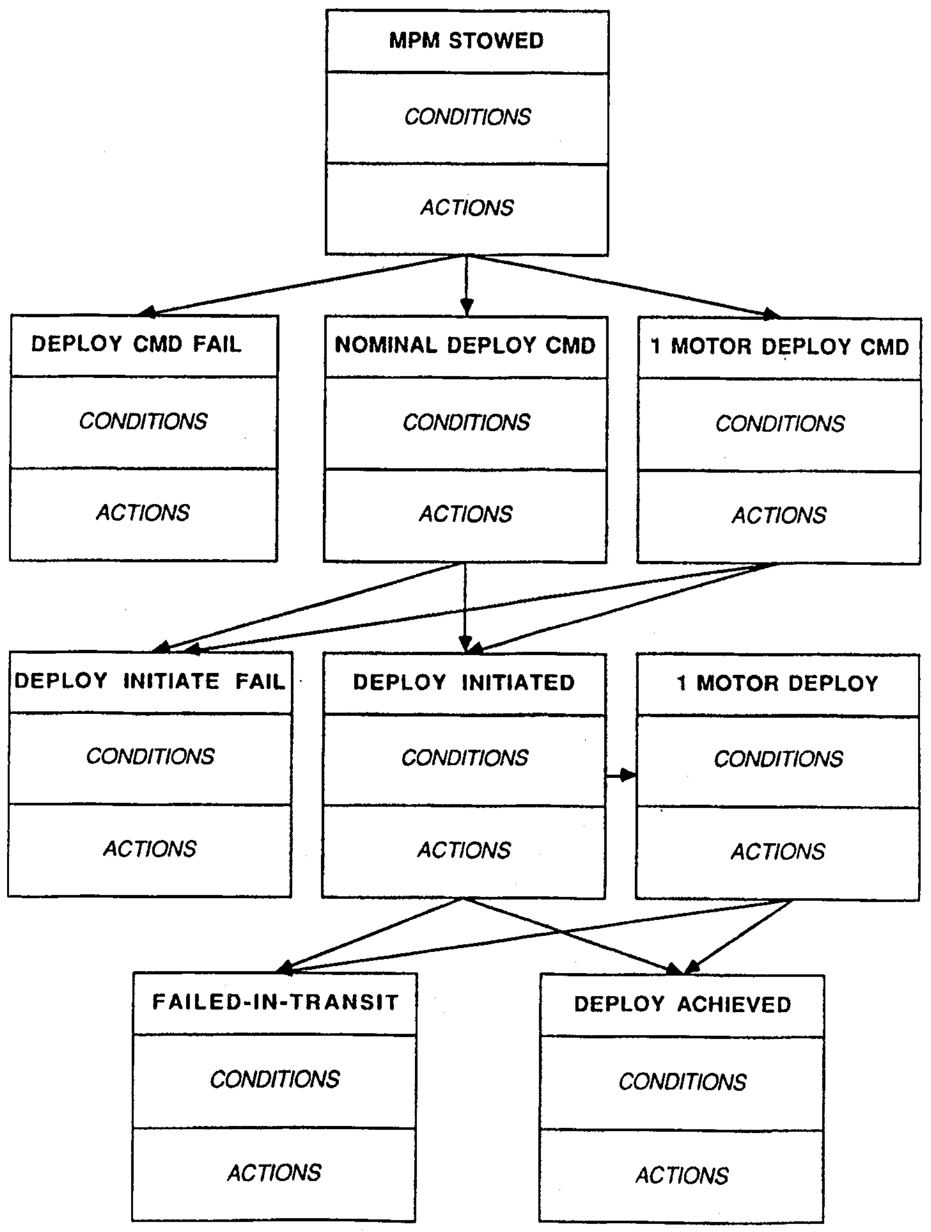


Figure 4 - Rule Hierarchy

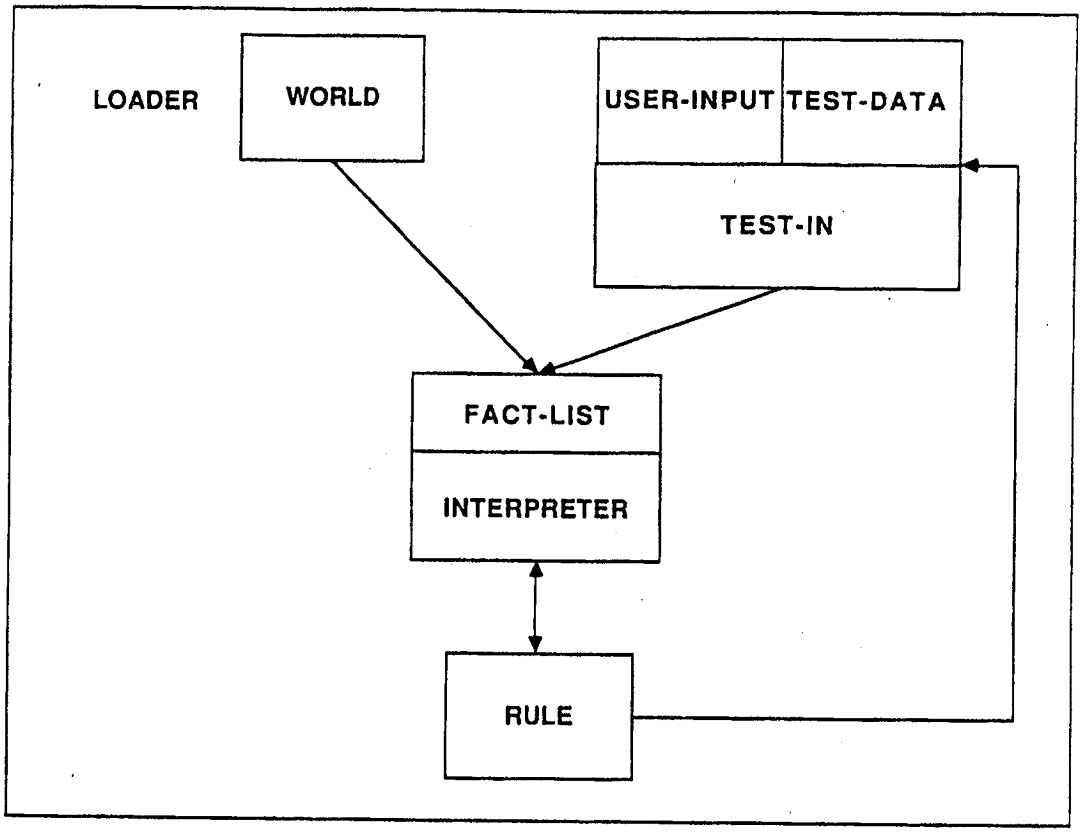


Figure 5 - DESSY Architecture for PC-based Rule Set

## PDRS/DES MODELING INFORMATION FLOW

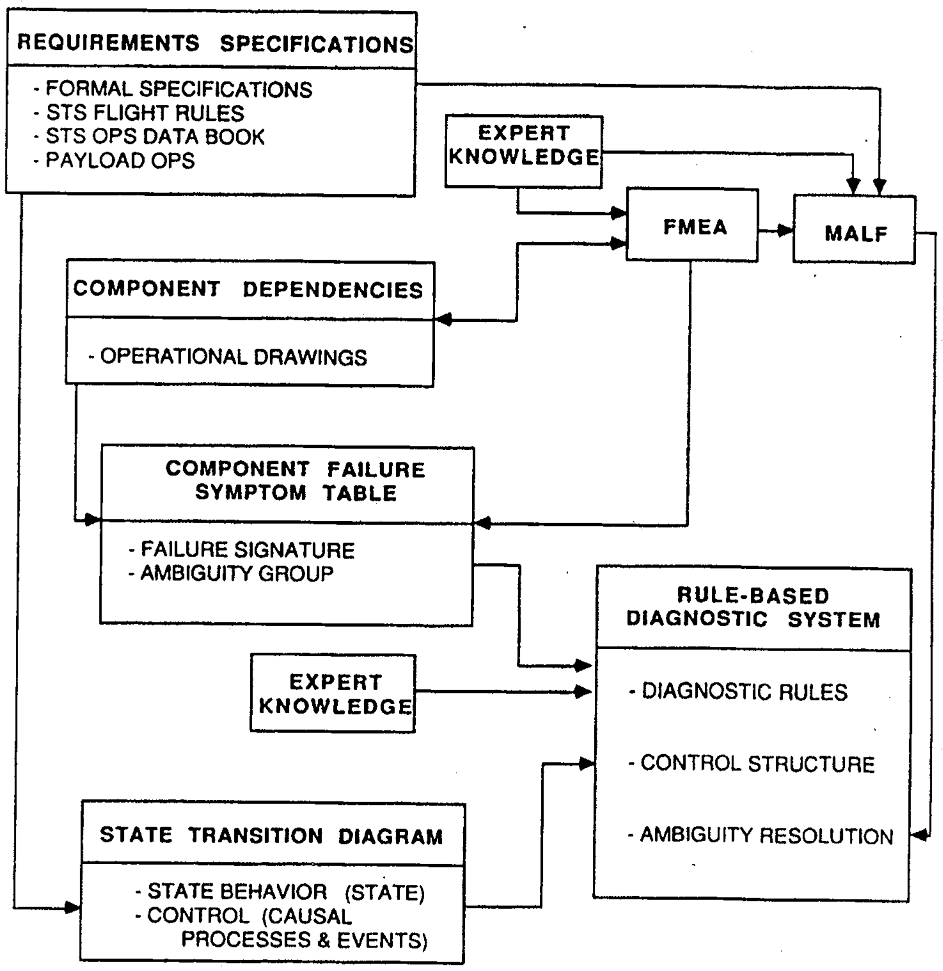


Figure 6 - Methodology of Knowledge Acquisition

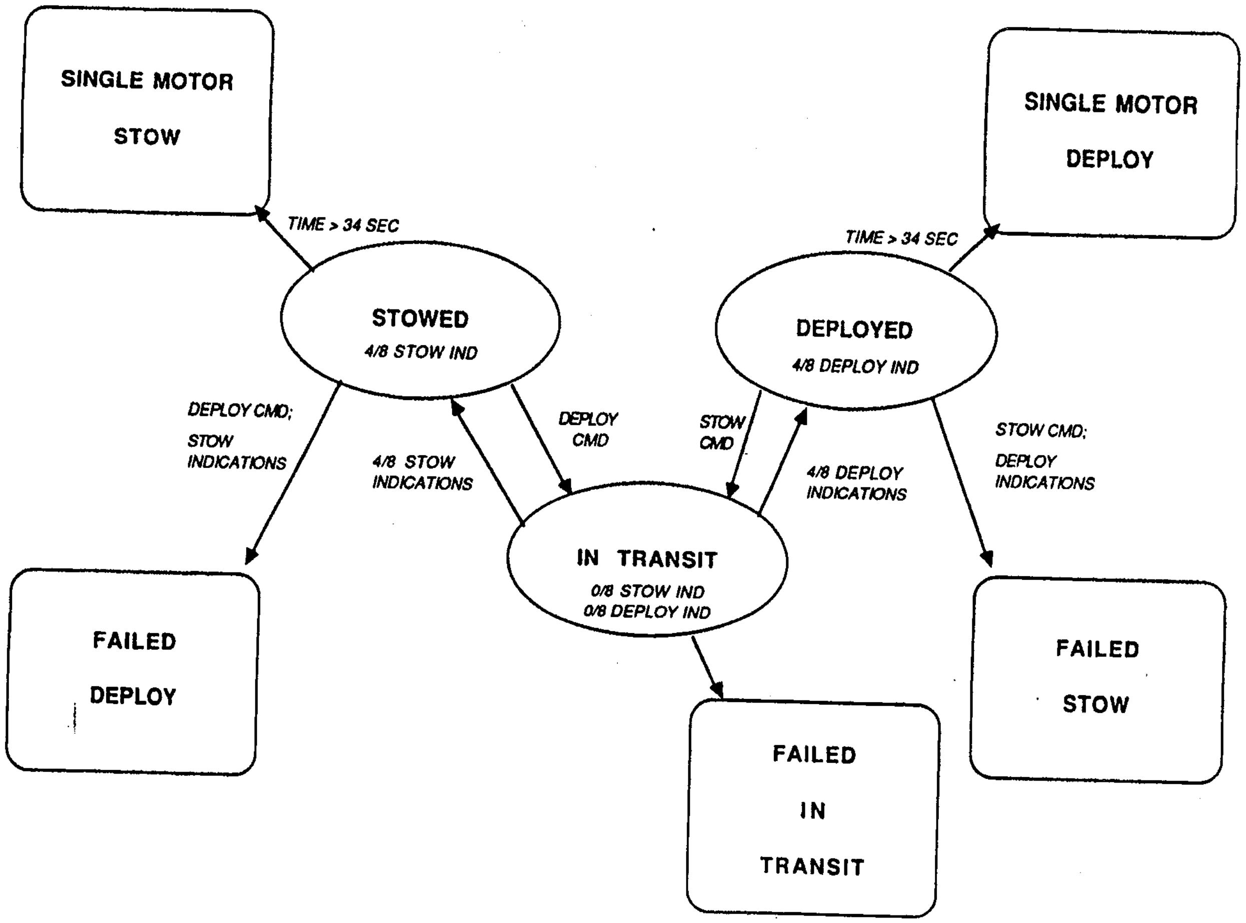


Figure 7 - State Transition Diagram

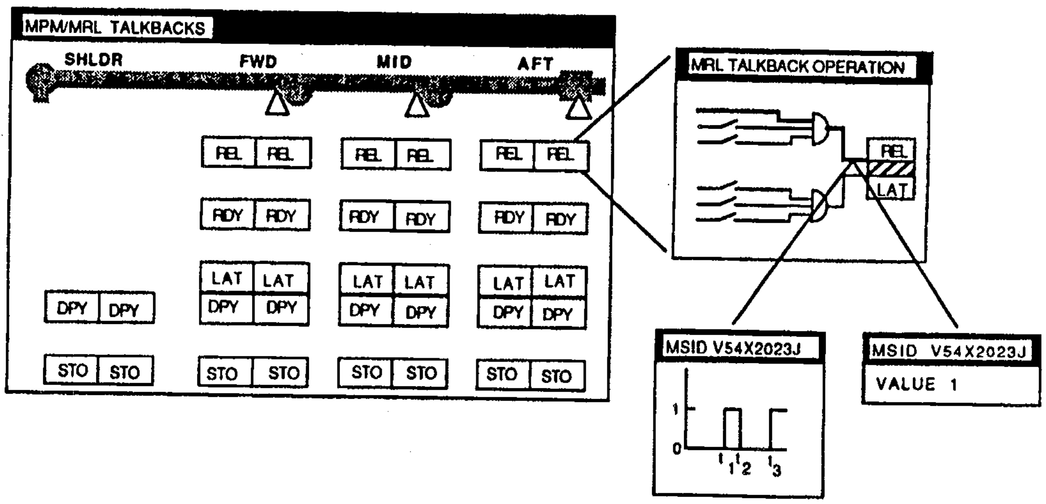
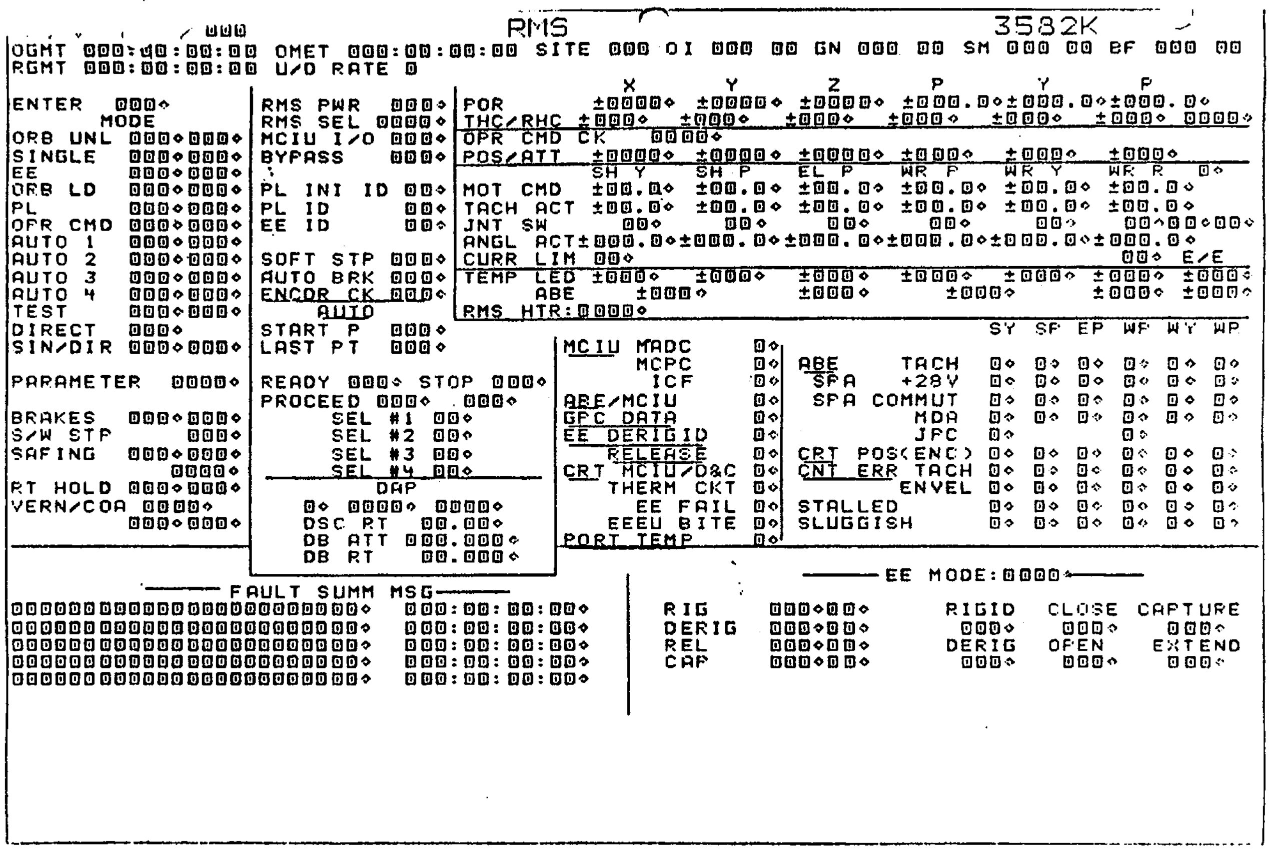
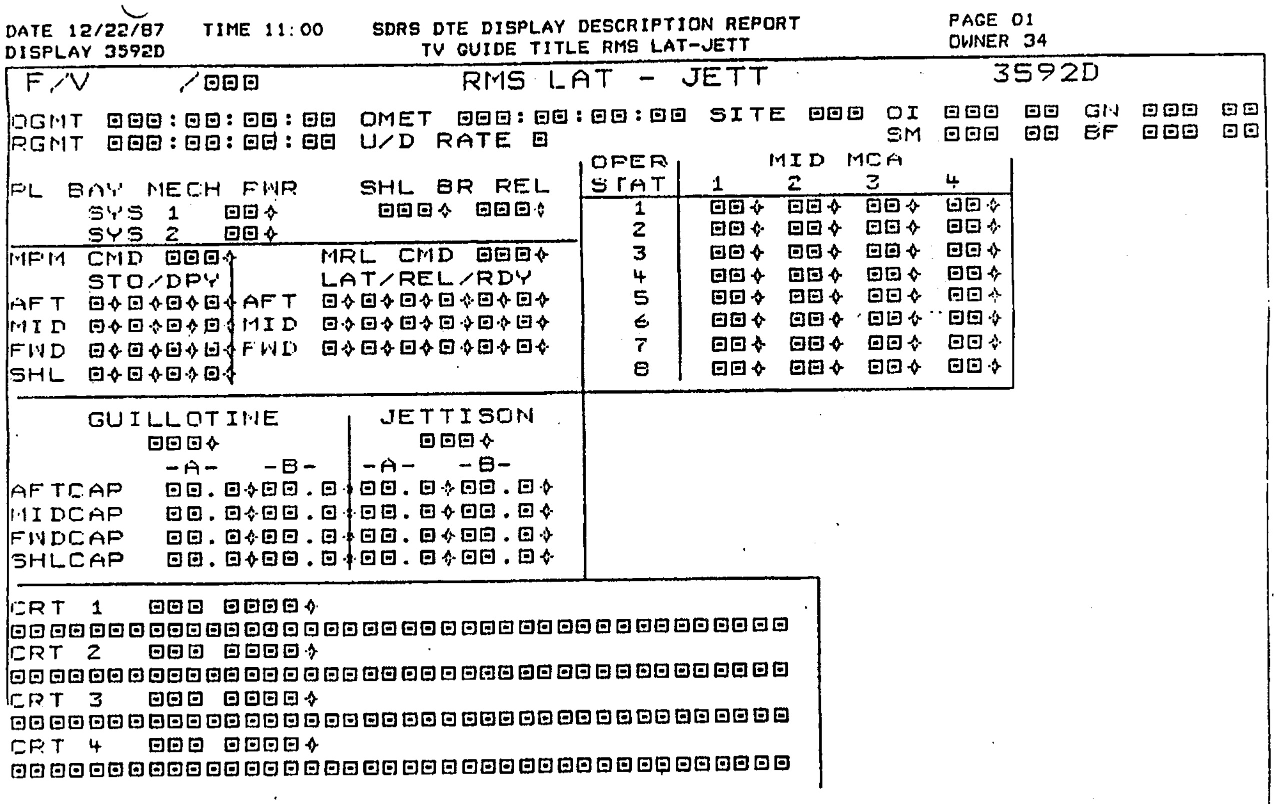


Figure 8 - Display Concepts for Prototype



```
       / 000                    RMS                                              3582K
OGMT 000:00:00:00  OMET 000:00:00:00  SITE 000 OI 000 00 GN 000 00 SM 000 00 BF 000 00
RGMT 000:00:00:00  U/D RATE 0
                                                     X          Y          Z          P          Y          P
ENTER    000◊        | RMS PWR    000◊ |POR        ±0000◊   ±0000◊   ±0000◊   ±000.0◊±000.0◊±000.0◊
       MODE          | RMS SEL   0000◊ |THC/RHC ±000◊    ±000◊    ±000◊    ±000◊    ±000◊    ±000◊  0000◊
ORB UNL  000◊000◊    | MCIU I/O   000◊ |OPR CMD CK     0000◊
SINGLE   000◊000◊    | BYPASS     000◊ |POS/ATT    ±0000◊   ±0000◊   ±0000◊   ±000◊    ±000◊    ±000◊
EE       000◊000◊    |                 |           SH Y      SH P      EL P      WR P      WR Y      WR R     0◊
ORB LD   000◊000◊    | PL INI  ID  00◊ |MOT CMD    ±00.0◊   ±00.0◊   ±00.0◊   ±00.0◊   ±00.0◊   ±00.0◊
PL       000◊000◊    | PL ID       00◊ |TACH ACT   ±00.0◊   ±00.0◊   ±00.0◊   ±00.0◊   ±00.0◊   ±00.0◊
OPR CMD  000◊000◊    | EE ID       00◊ |JNT SW        00◊      00◊      00◊      00◊      00◊      00^00◊00◊
AUTO 1   000◊000◊    |                 |ANGL ACT±000.0◊±000.0◊±000.0◊±000.0◊±000.0◊±000.0◊
AUTO 2   000◊000◊    | SOFT STP   000◊ |CURR LIM 00◊                                                 00◊  E/E
AUTO 3   000◊000◊    | AUTO BRK   000◊ |TEMP LED ±000◊    ±000◊    ±000◊    ±000◊    ±000◊    ±000◊  ±000◊
AUTO 4   000◊000◊    | ENCOR CK   000◊ |       ABE      ±000◊            ±000◊          ±000◊          ±000◊  ±000◊
TEST     000◊000◊    |       AUTO      |RMS HTR:0000◊
DIRECT   000◊        | START P    000◊                                                  SY   SP   EP   WP   WY   WP
SIN/DIR  000◊000◊    | LAST PT    000◊         MCIU MADC      0◊|
                     |                                  MCPC      0◊|ABE      TACH     0◊   0◊   0◊   0◊   0◊   0◊
PARAMETER    0000◊   | READY 000◊ STOP 000◊           ICF      0◊| SPA     +28V     0◊   0◊   0◊   0◊   0◊   0◊
                     | PROCEED 000◊  000◊  ABE/MCIU       0◊| SPA COMMUT        0◊   0◊   0◊   0◊   0◊   0◊
BRAKES   000◊000◊    |      SEL #1  00◊    GPC DATA       0◊|              MDA  0◊   0◊   0◊   0◊   0◊   0◊
S/W STP      000◊    |      SEL #2  00◊    EE DERIGID     0◊|              JPC  0◊             0◊
SAFING   000◊000◊    |      SEL #3  00◊        RELEASE    0◊|CRT POS(ENC)      0◊   0◊   0◊   0◊   0◊   0◊
            0000◊    |      SEL #4  00◊    CRT MCIU/D&C   0◊|CNT ERR TACH      0◊   0◊   0◊   0◊   0◊   0◊
RT HOLD  000◊000◊    |         DAP             THERM CKT  0◊|           ENVEL  0◊   0◊   0◊   0◊   0◊   0◊
VERN/COA 0000◊       |   0◊ 0000◊ 0000◊          EE FAIL  0◊|STALLED           0◊   0◊   0◊   0◊   0◊   0◊
         000◊000◊    |   DSC RT   00.00◊       EEEU BITE  0◊|SLUGGISH          0◊   0◊   0◊   0◊   0◊   0◊
                     |   DB ATT 000.000◊   PORT TEMP      0◊|
                     |   DB RT   00.000◊
                                                               ———— EE MODE:0000◊————
———— FAULT SUMM MSG————
0000000000000000000000000◊   000:00:00:00◊  |    RIG     000◊00◊       RIGID    CLOSE    CAPTURE
0000000000000000000000000◊   000:00:00:00◊  |    DERIG   000◊00◊        000◊     000◊      000◊
0000000000000000000000000◊   000:00:00:00◊  |    REL     000◊00◊       DERIG    OPEN     EXTEND
0000000000000000000000000◊   000:00:00:00◊  |    CAP     000◊00◊        000◊     000◊      000◊
0000000000000000000000000◊   000:00:00:00◊  |
```

Figure 9a - RMS 3582 Display



```
DATE 12/22/87    TIME 11:00    SDRS DTE DISPLAY DESCRIPTION REPORT          PAGE 01
DISPLAY 3592D                     TV GUIDE TITLE RMS LAT-JETT                 OWNER 34

 F/V      /000              RMS LAT - JETT                     3592D
OGMT 000:00:00:00  OMET 000:00:00:00  SITE 000  OI 000  00  GN 000  00
RGMT 000:00:00:00  U/D RATE 0                        SM 000  00  BF 000  00
                                          |OPER |      MID  MCA
PL BAY MECH PWR          SHL BR REL       |STAT |  1     2     3     4
      SYS 1    00◊        000◊  000◊      |  1  | 00◊   00◊   00◊   00◊
      SYS 2    00◊                        |  2  | 00◊   00◊   00◊   00◊
MPM  CMD 000◊        MRL CMD 000◊         |  3  | 00◊   00◊   00◊   00◊
     STO/DPY   |     LAT/REL/RDY          |  4  | 00◊   00◊   00◊   00◊
AFT  0◊0◊0◊0◊|AFT  0◊0◊0◊0◊0◊0◊           |  5  | 00◊   00◊   00◊   00◊
MID  0◊0◊0◊0◊|MID  0◊0◊0◊0◊0◊0◊           |  6  | 00◊   00◊   00◊   00◊
FWD  0◊0◊0◊0◊|FWD  0◊0◊0◊0◊0◊0◊           |  7  | 00◊   00◊   00◊   00◊
SHL  0◊0◊0◊0◊|                            |  8  | 00◊   00◊   00◊   00◊

     GUILLOTINE        |  JETTISON
        000◊           |    000◊
     -A-      -B-      | -A-     -B-
AFTCAP  00.0◊00.0◊     |00.0◊00.0◊
MIDCAP  00.0◊00.0◊     |00.0◊00.0◊
FWDCAP  00.0◊00.0◊     |00.0◊00.0◊
SHLCAP  00.0◊00.0◊     |00.0◊00.0◊

CRT 1   000  0000◊
000000000000000000000000000000000000000000000000000000000000
CRT 2   000  0000◊
000000000000000000000000000000000000000000000000000000000000
CRT 3   000  0000◊
000000000000000000000000000000000000000000000000000000000000
CRT 4   000  0000◊
000000000000000000000000000000000000000000000000000000000000
```

Figure 9b - RMS 3592 Display (MPM/MRL Display)

# The INCO Expert System Project: CLIPS in Shuttle Mission Control

Arthur N. Rasmussen
MITRE Corporation
John F. Muratore
Troy A. Heindel
NASA Lyndon B. Johnson Space Center
Houston, Texas

## INTRODUCTION

The INCO Expert System Project (IESP) was undertaken in 1987 by the Mission Operations Directorate (MOD) at NASA's Johnson Space Center (JSC) to explore the use of advanced automation in the mission operations arena [1]. One of MOD's primary responsibilities in the space shuttle program is the manning of the flight control positions in the Mission Control Center (MCC) at JSC. The MCC is the central hub for all ground support activities in support of Space Shuttle missions. Its responsibility extends from the moment of lift-off from the launch pad though the completion of landing. The flight controllers in the MCC are tasked with monitoring the health and status of all on-board systems and payloads, detecting and remedying errors and failures, and modifying flight activity plans accordingly.

The MCC is organized as shown in Figure 1. The flight controllers are members of a team headed by the Flight Director who has the ultimate responsibility for the safety and success of the flight. Directly supporting the Flight Director are the senior controllers in charge of each of the technical areas or disciplines; together they occupy the Flight Control Room (FCR) or "front room". The front room controllers cooperate to recommend actions to the Flight Director who is the final authority for all decisions. In support of most front room controllers are "back room" controllers who perform detailed analyses of problems and make recommendations for appropriate actions to their front room team leaders. The back room controllers are located in Multipurpose Support Rooms (MPSRs) which surround the FCR.

Through training and experience, a flight controller normally progresses from junior positions in the back room to become a front room controller; front room controllers advance to become flight directors based on their abilities and experience. A flight controller must earn certification to be considered competent to support shuttle missions. The certification process includes both study of shuttle systems and participation in mission simulations. Typical training times include one to two years before certification for a back room position and five or more years to reach the front room.

The high degree of required training combined with the normal rate of attrition as flight controllers move on to other responsibilities makes it difficult to maintain a trained cadre of certified flight controllers. Exacerbating this is the increasing flight rate needed to support NASA's planned science and Space Station *Freedom* construction missions. An additional problem is the age gap in experienced personnel caused by the period of relative inactivity that occurred between the end of the Apollo program and the beginning of shuttle flights.

A great deal of knowledge is being lost due to these factors. MOD has been investigating ways to retain this knowledge. One method being tried is the use of advanced automation approaches including artificial intelligence. The purpose of IESP is to determine the applicability of such approaches to real-time mission operations with the goal of improving flight decisions and thereby improving flight safety and mission success probabilities. An important additional goal is to determine the feasibility of using advanced automation to reduce the mission support manpower needed for long duration missions such as Space Station *Freedom* and a manned mission to Mars.

## APPROACH

To explore this area, the Integrated Communications Officer (INCO) flight control positions were selected by this project as its first area of investigation. This flight control discipline was selected because of ready access to expert controllers, because its systems are well instrumented and primarily digital in nature, and because of the existence of a number of self-contained monitoring tasks which are critical to mission success but not life critical.

An important factor in allowing the project to proceed and one which contributed greatly to its overall success is the requirement that any system developed under the project have no potential for negative impact on current MCC operations. This forced the development of a system that parallels the existing MCC data systems including the acquisition of real-time telemetry data. The existing MCC systems for acquiring telemetry data are based on processing the telemetry data through customized telemetry hardware and feeding it into a large mainframe; all flight control consoles are "dumb" terminals attached to the mainframe and have no processing capability. This approach fit well with past technologies but has numerous drawbacks in the face of emerging technologies such as artificial intelligence and distributed systems.

A four layer architecture has been conceived to permit operating the IESP system in parallel with the existing MCC systems and thereby ensure no negative impact on flight safety, shown in Figure 2. The first layer decommutates the data from the shuttle's 192,000 bit-per-second downlink stream. This is accomplished using a commercial telemetry processor. The resulting data are transferred into a host computer (a Unix-based engineering workstation) either by direct memory access or network connection.

The data are then fed to the second layer which performs algorithmic functions upon them. The operations performed by this layer are quite numerous but relatively simple, consisting mainly of conversion from analog-to-digital counts and calibration to engineering units. The algorithms at this level contain no discipline-specific knowledge.

More complex algorithmic operations are performed by layer three; it is this layer in which discipline-specific knowledge is introduced. A tool named CODE (Computation Development Environment) was developed which gives a user the ability to specify algorithms of moderate complexity in a high-level form; the tool uses the specifications to produce application programs that implement the algorithms in the 'C' programming language. The tool performs much of the work of integrating the program with the two lower architectural layers as well as with the user interface, making it substantially easier to create and modify the algorithms' implementations.

The fourth layer takes the information produced by the third layer and performs both algorithmic and heuristic operations on it. These operations are based on deeper knowledge than that embodied in the third layer and have been found to be generally well suited to the use of a rule-based system.

The four layer architecture is proving its viability in several ways. Drawing clean and distinct interfaces between the layers has allowed them to develop and evolve with minimal impact on each other; it has also facilitated design and implementation by independent teams of people without excessive burdens of communication and organization. Each layer has been certified independently and each has its own simulation capability so that progress on one upper layer is not held back waiting for completion of another.

The layered architecture plays a central role in the ability of the expert system to operate in real time. Each layer reduces the volume of data and increases its information density. At the lower layers, filtering such as noise removal is performed, simple redundancy is eliminated, subcomponents are united to create a single value, and decoding and calibration are performed. Upper layers add algorithmic knowledge and remove most of the remaining redundancy. The result is highly condensed data entering the fourth layer. By factoring these operations into the lower layers, the design applies the rule base to the task for which it is most suited: interpreting the data using deeper knowledge. Performing operations in an upper layer that can be reasonably done at a lower layer unnecessarily burdens the upper layer and (especially at the fourth layer) can create such inefficiency that the ability to respond in real-time is lost.

## CHOOSING THE REPRESENTATIVE PROBLEM

To probe the ability of using advanced automation in the flight control environment, an initial area of application was chosen within the INCO domain. The INCO flight control team is composed of a front room position and three back room support positions. This team is responsible for monitoring all of the shuttle's communications systems. The scope of the team's activities is clearly much too broad to attempt automating it in a single application; since the project's purpose is one of demonstration, it became obvious that a subsystem within the INCO realm should be chosen.

The system selected as the first candidate for automated monitoring was the payload communication subsystem. This system consists of hardware that maintains a bidirectional data link carrying status data from the payloads to the shuttle and commands from the shuttle to the payloads. This system was chosen because it presents a monitoring task of moderate difficulty for a flight controller. It is an important system that is actively used during most shuttle missions; it is clearly critical to mission success but is not considered life critical.

The payload communication system has presented problems in past missions, both simulated and actual, not only because of failures but also because its configuration states are numerous and many manual state transitions are performed during a typical mission. An error in execution of a transition can lead to loss of communication and possible loss of control of a payload, so such errors must be quickly analyzed and corrected. For this reason expert INCO flight controllers felt it would provide a good challenge for an automated system; additionally the system, if successful, would be immediately useful.

The payload communications system is quite well instrumented and nearly all status data is available on the ground from the shuttle telemetry stream. This contributed to its selection as the candidate system. However it has several subsections for which the desired level of instrumentation is not available. This is a significant factor because concerns such as cost, weight, complexity, and bandwidth limitations always compromise the level of on-board instrumentation, and unavailable data must be inferred from the data that is available. A similar problem exists when an on-board sensor fails. A demonstration of automated monitoring must show its ability to deal with this problem to be considered robust.

## CHOOSING THE KNOWLEDGE-BASED TOOL

CLIPS was chosen as the rule-based interpreter to be used at the fourth architectural level. CLIPS was selected because it met a number of criteria set forth at the project's outset. Its forward-chaining basis is appropriate to the task of monitoring and analyzing a stream of data. Initial tests seemed to show CLIPS's speed of execution was not outstanding but was adequate to produce an acceptable level of overall system performance. An experienced flight controller takes an average of less than 5 seconds and no longer than 15 to 20 seconds to analyze a problem in the payload communication system; this was taken as the desired level of performance for the automated system, and the initial testing showed a CLIPS-based system could achieve this.

Another factor in the choice is the ease with which CLIPS can be embedded within a larger system. The role taken by other rule-based systems seems much too dominant for smooth integration into the project's architecture. It is strongly felt the rule interpreter must fit into the architecture dictated by the problem; however, the more elaborate the shells of candidate systems, the more likely it appears that the architecture would be forced to fit the mold of the shell rather than the needs of the problem.

Other factors in the choice of CLIPS include the local availability of a high level of support and the ready access to the source code of CLIPS. The former allows keeping pace with a very tight implementation schedule, while the latter allows minor changes to CLIPS such as instrumenting desired portions to determine information such as rule firing activities.

## IMPLEMENTATION APPROACHES

The initial approach taken embodied the knowledge entirely as CLIPS facts. The rules written to support these contained almost no domain knowledge but rather interpreted portions of the CLIPS facts as meta-rules. This resulted in an inference engine which was customized to the needs of the problem. The approach proved extremely flexible and very useful for the initial development of the knowledge base. It also provided an inherent ability to trace backwards through an inference chain, an ability enjoyed by backward chaining engines but normally lacking with forward chaining. Although primarily intended to support an 'explain' facility for the user, it also proved very useful for knowledge verification. Unfortunately the additional layer of interpretation became too inefficient to be used for real-time operations and was replaced by a second rule set which embodied the knowledge in a more direct, conventional manner.

The second and current approach has been produced by manually translating the meta-rules expressed as facts in the first approach directly into the CLIPS rules. The portion of the fact base that maintained the state of the payload subsystem remained basically unchanged. This greatly improved performance but still left the system several times slower than required to provide adequate real-time response.

Additional adjustments to the rules have achieved an acceptable level of performance. The largest gain has come from keeping the number of partial matches as low as possible. A primary technique for doing so is splitting the task of updating the fact base into rules that "predict" which facts need updating (by examining existing facts) and rules that actually perform the update.

The rules of both approaches are constructed in a phase-based manner; during each phase, a "phase" fact naming the phase is asserted into the fact base. The fact enables only those rules relevant to the phase, thereby inhibiting all other rules. This allows the rules to be more easily implemented and tested because of the repeatability of the rules' response to incoming data. The approach also creates a more modular and stable environment for development and evolution of the rule base.

## STRUCTURING THE RULE BASE

The rule base is organized as a small number of one-time startup initialization phases followed by a repeated cycle of phases called the "major cycle". The major cycle repeats indefinitely and is composed of three main steps: acquiring real-time data, analyzing the data and posting the data for display by the user interface.

The acquisition of real-time data is performed by the rule base in two phases. The first phase polls the third architectural layer and receives the next sampling of data while the second phase removes data that has become out of date. Only data that has actually changed is admitted to the fact base; data whose values have not changed do not cause any alteration of the fact base. This greatly improves overall efficiency since the rules are not called into play unless there is a need.

From early in the design of the rule base, the admission of data into the fact base only during a single phase has been considered essential for two reasons. First it preserves the time homogeneity of the data. The data is sampled at discrete intervals, and data sampled in the same interval must be considered in the context of the environment of that interval. Examining a datum out of the context of its sample interval can lead to inappropriate conclusions since many events are time-correlated.

The second reason for limiting the assertion of new data into the fact base to a specific phase is that asynchronous injection of data can interrupt the line of inference currently being constructed. Because the data from a single interval are interrelated, they must be analyzed together; injecting newer values before the process is complete is inappropriate and could lead to incorrect conclusions.

The combination of asserting only changed values into the fact base and of allowing new data to enter the fact base only once per major cycle results in a rather large variation in the time needed to complete a major cycle. When there are many data that change, the rules require a longer period to analyze the changes; fewer changes mean the major cycle completes more rapidly.

The result is some data are missed when the major cycle fails to complete the analysis of one interval's data before the next interval's data is available for sampling. Since this means that some data is ignored, it was initially cause for concern. However, in practice it has not been a problem: the data being monitored rarely changes so drastically that a major cycle cannot be completed. In addition, there are almost no events that occur whose duration is so short that they would not be detected.

To ensure that short duration events are not missed because of the rule base's inability to examine data from every sampling interval, two techniques have been explored. The first is the ability to interrupt the basic phase processing should the need arise to handle an usual or high priority event. This is implemented using "asynchronous phases" which can be invoked either between any two phases in the major cycle or between any two rule firings. This allows the rule base to respond to the event and then return to complete the interrupted processing.

The second technique for dealing with delayed response from the rule base is embodied in the interface between the rule base and the lower layers of the architecture. In a manner similar to saving lower level hardware interrupts while a higher interrupt is processed, the interface can poll the lower layers and hold significant events in abeyance until the rule base is able to deal with them.

These techniques exemplify the overall system design philosophy of managing activities at the lowest possible level in the architecture. They also mimic the way in which a flight controller deals with asynchronous events. If the event is of a higher priority than the one being currently engaged, the current activity is set aside in favor of the event needing attention. This presents the same problems for the rule base that it does for the flight controller, the most troublesome of which is the need to switch contexts quickly and smoothly while retaining the ability to resume previously suspended tasks.

The IESP system's technique of using asynchronous phases provides the ability to clear the agenda of pending rules so that the new event can be addressed, but there is no similar facility for the fact base. The ability to focus on a specific subset of the fact base or to create a new level of the base would be useful for handling this problem; CLIPS currently has no such facility.

A set of rules used for debugging has been developed. They provide access to both the internal operation of the rule base and to the CLIPS control and user interface functions. These rules are constructed in a modular fashion; they can be excluded with no effects on the operation of the system other than to disable their function. Most of the rules operate at a salience higher than any other rules in the rule base so that when they are triggered there is an immediate response. This minimizes the time between triggering a debugging rule and its firing, which in turn means the system can be examined at any desired point in its operation.

## USER INTERFACE

The console displays currently used in the MCC are based on black-and-white video technology and possess limited graphics capabilities; an example of the content of these displays is shown in Figure 3. To aid the flight controller in visualizing the payload communication system, the display shown in Figure 4 was developed. Both figures show essentially the same information; the IESP display makes use of the high-resolution color graphics display capability of the project's workstation to present an animated system schematic and associated status tables.

The IESP display uses a simple color scheme to keep visual distractions to a minimum. The color green is used to indicate a component or status is "good" while red indicates a failure. Yellow is used to indicate that there is a possible problem but that no clear failure has occurred. Although yellow may signify an unconfirmed malfunction, it most commonly occurs when an element of the communication system is configured incorrectly for the current mission context. Since the components of the shuttle are inherently very reliable, this occurs far more frequently than actual failure.

As the system has developed, it has been demonstrated on a regular basis to experienced flight controllers. The area that has received the most intensive scrutiny is the user interface, as expected. Feedback from the controllers has been used as much as possible to tailor the interface to their needs, resulting in four major redesigns. This feedback, while requiring considerable and persistent effort to obtain, has had two extremely positive effects. First and most obvious, it has resulted in a much superior design. Second, it has engendered in the flight controllers a strong sense of ownership of the system and responsibility for its effectiveness. This has played a strong role in the high degree of user acceptance the system is receiving.

As the system has developed, two features have been incorporated into the user interface that have never before been used in Mission Control. The first is an explanatory facility that is in addition to the normal 'help' facility. By choosing an on-screen graphical object, the flight controller can request that the system display an explanation of the reasoning leading to the current status of the object. This provides a means for experienced operators to review the system's conclusions and for operators in training to learn about both the operation of the payload subsystem and the expert system. The facility is supported by a set of rules and underlying procedural code that access both the fact base and prepared script files.

The second feature, dubbed 'telemetry pop-up', allows a flight controller to request data from lower layers of the architecture such as raw telemetry values associated with a conclusion of the rule base. This is proving valuable not only for verifying correct operation of the rule base, but also as a means of building confidence in experienced flight controllers (who in the past used only the raw telemetry as the basis for their conclusions). The feature provides visibility into all layers of the architecture and has been very warmly received; it is proving to be a key element in the acceptance of the system by its users.

The IESP system separates the user interface task from the inferencing task by implementing them as separate processes. The inferencing process runs as a "background" task so that it can actively monitor data while the user's attention may be focused elsewhere. The user activates the user interface process in order to monitor the current status of the inferencing process or to control its operation. The two processes communicate using interprocess communication facilities, using predominantly shared memory for greatest speed.

This design not only allows continuous operation of the inferencing process, but also allows for several user interface processes to be monitoring the inferencing simultaneously. Each user interface process maintains its own context for user 'help', explanations and display configuration. By using a distributed display facility such as that provided by X-windows, the expert system can be monitored and controlled from any workstation in the MCC if desired.

A separate messaging service is used to allow a background task such as the inferencing process to communicate with the user when the user interface process is not active. This service is always active and maintains a list of recent messages in a window tile at all times. The structure of the design is shown in Figure 5.

The inferencing process has the capability of logging all facts as they are asserted into the fact base which provides a log of all data acquired from the real-time data interface. The facility includes the ability to replay data from the log. This feature is used to great advantage in the development of the rule base since it allows recording a scenario and replaying it many times. As the rule base evolves, a suite of files containing a variety of test situations are replayed to observe the effects of the evolution. The facility has also shown itself to be extremely useful for both training and demonstrations.

## SUMMARY

The IESP system incorporating the rule base was first used in simulations of the STS-26 return-to-flight mission and was used during the actual mission to monitor the deployment of the mission's payload, the Tracking and Data Relay Satellite (TDRS); it is the first knowledge-based system ever used in the MCC during manned space flight operations.

It has repeatedly demonstrated its ability to aid the experienced flight controller in coming to a correct conclusion more quickly and accurately than the system used previously. In addition the wide range of visibility into all layers of the architecture has been used to quickly detect a problem and trace it to its root in the telemetry stream, a technique that is not possible with the previous system. This visibility is also a key ingredient in the high level of user acceptance enjoyed by the system.

As work on the system continues, expectations are that it will allow flight controllers of lesser experience to function at the level of the more seasoned controllers. The 'help' and explanatory features can help the controllers learn more quickly and refresh their knowledge more readily. Current plans also include a small reduction in flight control team size based on the success of the project; the hope is that continued efforts in advanced automation and knowledge capture can reduce the potentially staggering operations costs of long-term programs like Space Station *Freedom*.

The system has been reviewed and praised by NASA senior management as exciting and innovative. The success of the system has lead to the incorporation of many of the principles it has demonstrated into the requirements for the Mission Control Center Upgrade project (MCCU) as well as the requirements for the Space Station Control Center (SSCC). It also has led to a follow-on project, the Real-Time Data System project (RTDS), that is continuing to explore new approaches to advanced automation in support of manned space flight. RTDS is currently being expanded to encompass a majority of the flight control disciplines and includes advanced automation using both conventional programming and knowledge-based techniques. In addition, the techniques and much of the IESP system itself are being incorporated into flight research expert system projects at both NASA Ames-Dryden and Edwards Air Force Base [2].

Planned work includes a distributed expert system that will interact with both conventional and expert systems supporting individual flight control disciplines. The distributed system will integrate the conclusions of the individual areas to detect problems affecting multiple disciplines and advise individual areas of the overall status of shuttle systems. Other plans include intelligent reconfiguration of the payload communication expert system to adapt to new payloads and an expert system to verify the programming of equipment used in layer one to process telemetry data.

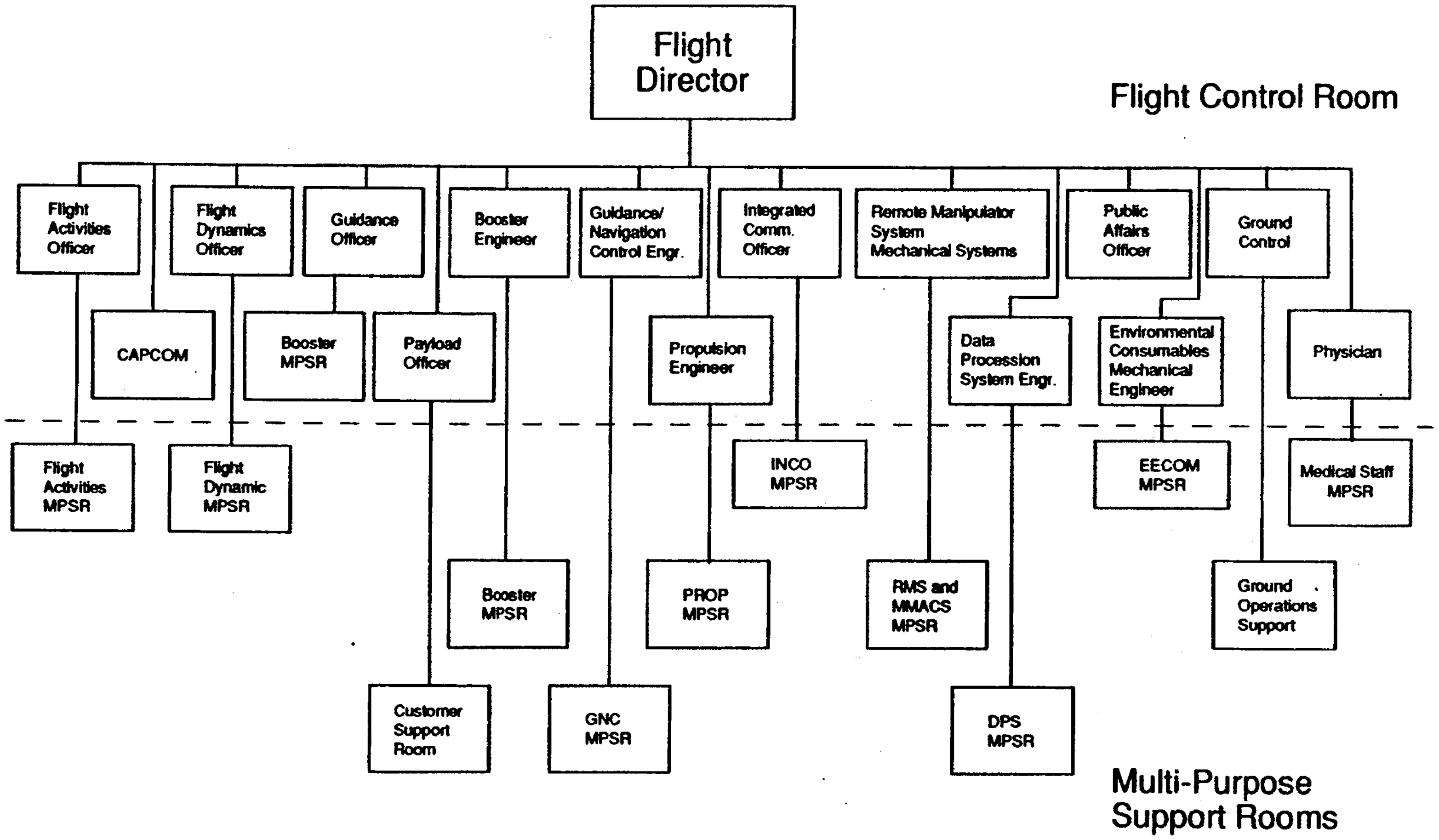


Figure 1. Mission Control Center

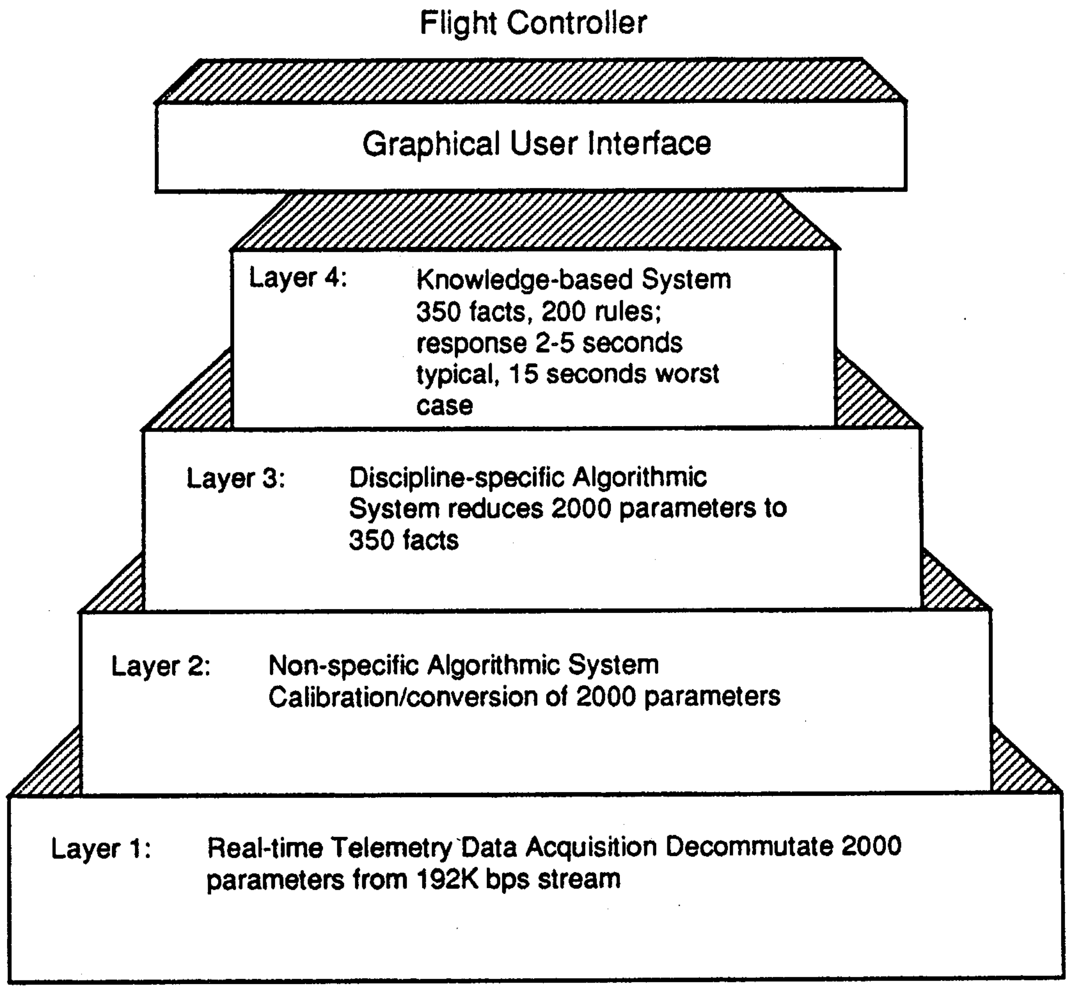
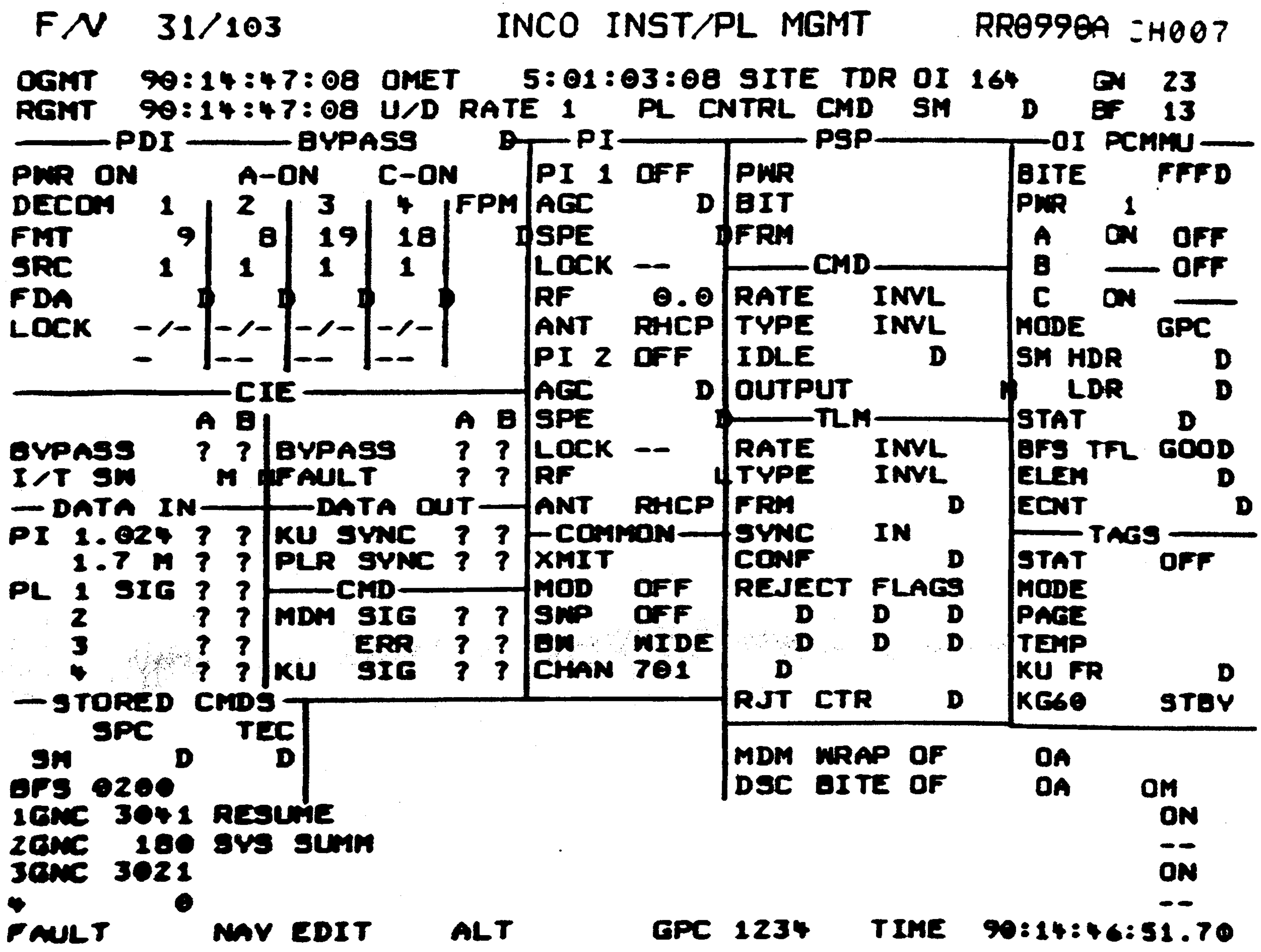

Flight Controller
Graphical User Interface
Layer 4: Knowledge-based System 350 facts, 200 rules; response 2-5 seconds typical, 15 seconds worst case
Layer 3: Discipline-specific Algorithmic System reduces 2000 parameters to 350 facts
Layer 2: Non-specific Algorithmic System Calibration/conversion of 2000 parameters
Layer 1: Real-time Telemetry Data Acquisition Decommutate 2000 parameters from 192K bps stream

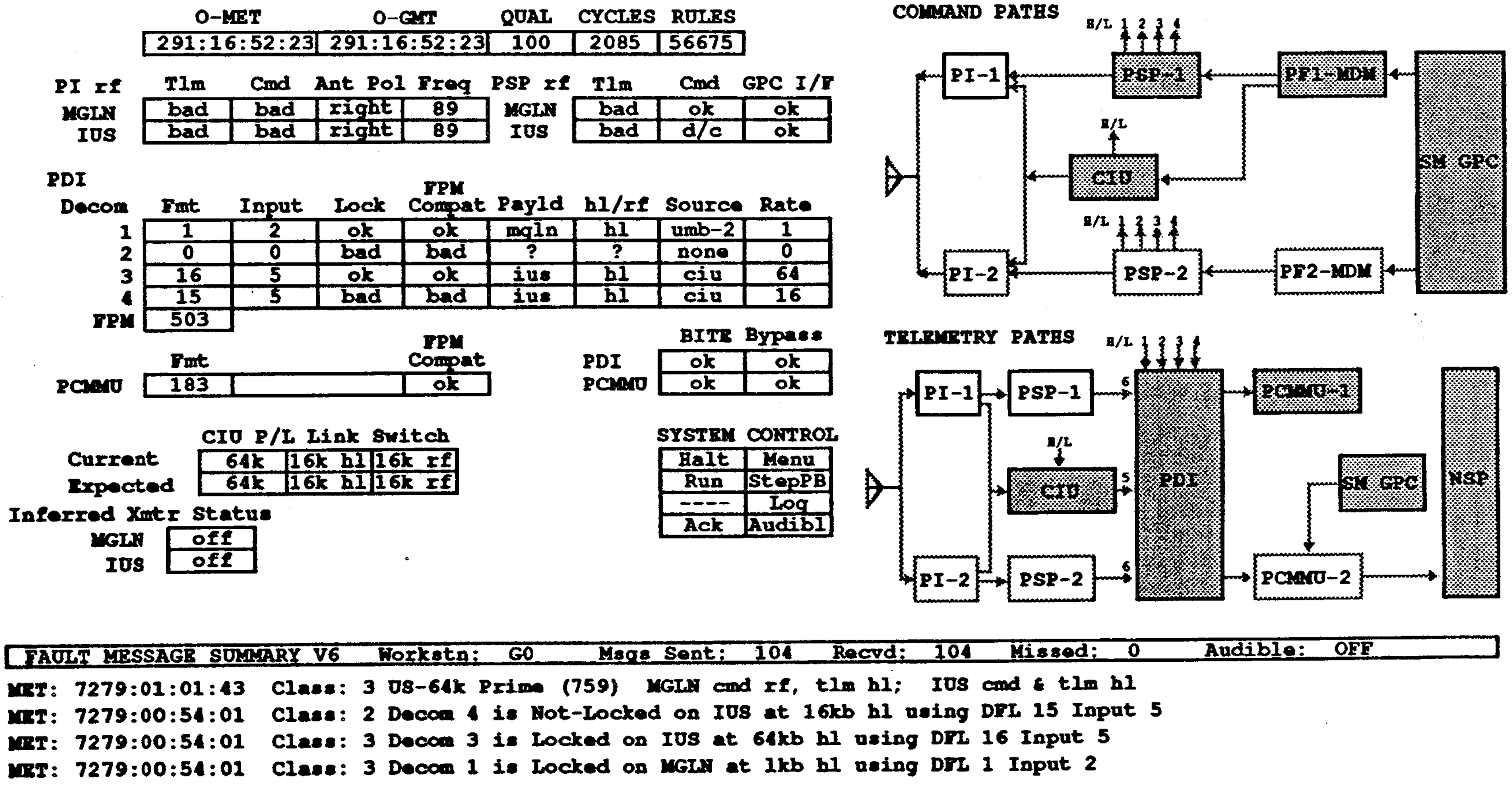
Figure 2. IESP Layered Architecture



```
F/V  31/103              INCO INST/PL MGMT         RR0990A :H007

OGMT   90:14:47:08 OMET    5:01:03:08 SITE TDR OI 164     GN  23
RGMT   90:14:47:08 U/D RATE 1    PL CNTRL CMD  SM    D    BF  13
 ------PDI ------BYPASS     D----PI--------PSP----------OI PCMMU----
PWR ON       A-ON    C-ON   |PI 1 OFF |PWR           |BITE     FFFD
DECOM  1  | 2  | 3  | 4 |FPM|AGC     D|BIT           |PWR   1
FMT     9 |  8 | 19 | 18 | DSPE     DFRM            | A   ON  OFF
SRC    1  | 1  | 1  | 1 |   |LOCK --  |-----CMD------| B   ---- OFF
FDA       D    D    D    D  |RF    0.0|RATE    INVL  | C   ON  ----
LOCK  -/- |-/- |-/- |-/- |  |ANT  RHCP|TYPE    INVL  |MODE      GPC
       -  |--  |--  |--  |  |PI 2 OFF |IDLE       D  |SM HDR      D
-----------CIE------------- |AGC     D|OUTPUT        M   LDR      D
          A B|         A B |SPE      D-----TLM-------|STAT      D
BYPASS    ? ?|BYPASS   ? ? |LOCK --   |RATE    INVL  |BFS TFL  GOOD
I/T SW     M MFAULT    ? ? |RF       LTYPE    INVL  |ELEM        D
-DATA IN-----|---DATA OUT--|ANT  RHCP|FRM        D  |ECNT         D
PI 1.024  ? ?|KU SYNC  ? ? |-COMMON--|SYNC    IN    |-----TAGS-------
   1.7 M  ? ?|PLR SYNC ? ? |XMIT     |CONF       D  |STAT     OFF
PL 1 SIG  ? ?|-----CMD-----|MOD  OFF |REJECT FLAGS  |MODE
   2      ? ?|MDM SIG  ? ? |SWP  OFF |   D    D   D |PAGE
   3      ? ?|     ERR ? ? |BW   WIDE|   D    D   D |TEMP
   4      ? ?|KU  SIG  ? ? |CHAN 701 |  D           |KU FR        D
-STORED CMDS ---------------------------|RJT CTR     D |KG60      STBY
   SPC     TEC |                         |-----------------------------
 SM      D    D|                         |MDM WRAP OF      OA
BFS 0200       |                         |DSC BITE OF      OA     OM
1GNC 3041 RESUME                                                  ON
2GNC  180 SYS SUMM                                                --
3GNC 3021                                                         ON
4        0                                                        --
FAULT     NAV EDIT    ALT        GPC 1234   TIME   90:14:46:51.70
```

Figure 3. Mission Control Console Display



| O-MET | O-GMT | QUAL | CYCLES | RULES |
|---|---|---|---|---|
| 291:16:52:23 | 291:16:52:23 | 100 | 2085 | 56675 |

| PI rf | Tlm | Cmd | Ant Pol | Freq |
|---|---|---|---|---|
| MGLN | bad | bad | right | 89 |
| IUS | bad | bad | right | 89 |

| PSP rf | Tlm | Cmd | GPC I/F |
|---|---|---|---|
| MGLN | bad | ok | ok |
| IUS | bad | d/c | ok |

PDI

| Decom | Fmt | Input | Lock | FPM Compat | Payld | hl/rf | Source | Rate |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | ok | ok | mgln | hl | umb-2 | 1 |
| 2 | 0 | 0 | bad | bad | ? | ? | none | 0 |
| 3 | 16 | 5 | ok | ok | ius | hl | ciu | 64 |
| 4 | 15 | 5 | bad | bad | ius | hl | ciu | 16 |
| FPM | 503 | | | | | | | |

| | Fmt | | FPM Compat |
|---|---|---|---|
| PCMMU | 183 | | ok |

| | BITE | Bypass |
|---|---|---|
| PDI | ok | ok |
| PCMMU | ok | ok |

CIU P/L Link Switch

| | | | |
|---|---|---|---|
| Current | 64k | 16k hl | 16k rf |
| Expected | 64k | 16k hl | 16k rf |

Inferred Xmtr Status

| | |
|---|---|
| MGLN | off |
| IUS | off |

SYSTEM CONTROL

| | |
|---|---|
| Halt | Menu |
| Run | StepPB |
| ---- | Log |
| Ack | Audibl |

```
FAULT MESSAGE SUMMARY V6   Workstn:  G0     Msgs Sent:  104    Recvd:  104    Missed:  0      Audible:  OFF
MET: 7279:01:01:43  Class: 3 US-64k Prime (759)  MGLN cmd rf, tlm hl;  IUS cmd & tlm hl
MET: 7279:00:54:01  Class: 2 Decom 4 is Not-Locked on IUS at 16kb hl using DFL 15 Input 5
MET: 7279:00:54:01  Class: 3 Decom 3 is Locked on IUS at 64kb hl using DFL 16 Input 5
MET: 7279:00:54:01  Class: 3 Decom 1 is Locked on MGLN at 1kb hl using DFL 1 Input 2
```

Figure 4. IESP Graphical Display

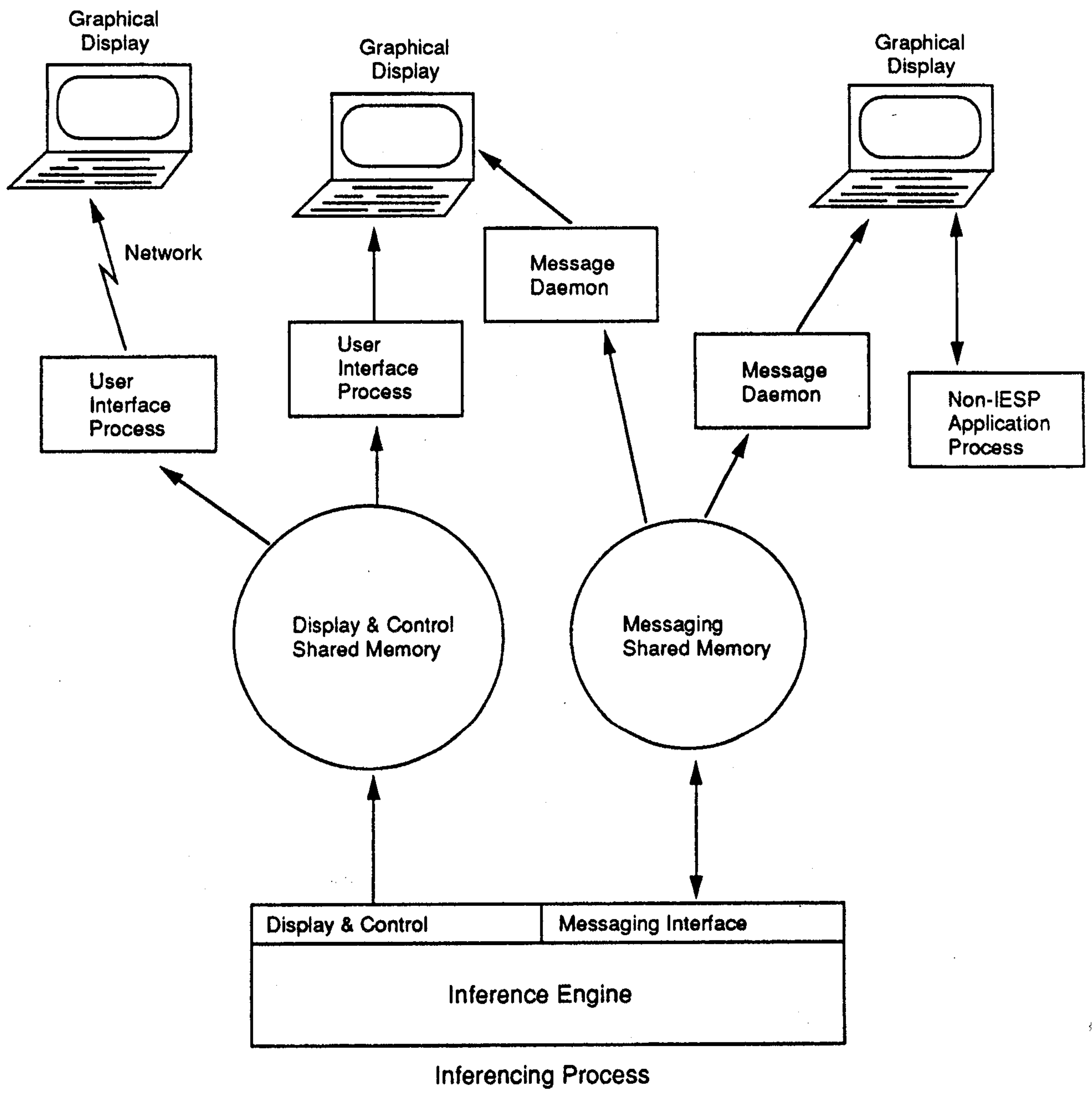


Figure 5. Process Structure

## REFERENCES

1. John F. Muratore, Troy A. Heindel, Terri B. Murphy, Arthur N. Rasmussen and Robert Z. McFarland, Space Shuttle Telemetry Monitoring by Expert Systems in Mission Control. In *Innovative Applications of Artificial Intelligence*, H. Schorr and A. Rappaport, Ed., AAAI Press, 1989, pp. 3-14.

2. D. A. Mackall, M. D. Pieckett, L. J. Schilling and C. A. Wagner, *The NASA Integrated Test Facility and Its Impact on Flight Research*, NASA Technical Memorandum 100418, Ames-Dryden Flight Research Facility, 1988, 14 pages.

## SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| IESP | INCO Expert System Project |
| INCO | Integrated Communications Officer |
| JSC | Johnson Space Center, Houston, Texas |
| MCC | Mission Control Center |
| MCCU | Mission Control Center Upgrade project |
| MOD | Mission Operations Directorate |
| RTDS | Real-Time Data Systems project |
| SSCC | Space Station Control Center |
| TDRS | Tracking and Data Relay Satellite |

## TRADEMARKS

Unix is a trademark of the AT&T Corporation.

# A comparison of CLIPS- and LISP- based approaches to the development of a real-time expert system.

R. Frainier*, N. Groleau**, R. Bhatnagar**, C. Lam***, M. Compton*, S. Colombano*, S. Lai**, P. Szolovits +, M. Manahan ++, I. Statler +++, L. Young**.

* Artificial Intelligence Research Branch, NASA-Ames Research Center, NAS Moffett Field, CA. 94035

** Man-Vehicle Laboratory, Massachusetts Institute of Technology, Cambridge, MA. 02139

*** Knowledge Systems Laboratory, Stanford University, Palo Alto, CA. 94034

+ Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA. 02139

++ Human-Computer Interaction Laboratory, NASA-Johnson Space Center, Houston, TX. 77058

+++ Aerospace Human Factors Research Branch, NASA-Ames Research Center, NAS Moffett Field, CA. 94035

## Abstract

*This paper describes an ongoing expert system development effort started in 1988 which is evaluating both CLIPS- and LISP- based approaches. The expert system is being developed to a project schedule and is planned for flight on Space Shuttle Mission SLS-2 in 1992. The expert system will help astronauts do the best possible science for a vestibular physiology experiment already scheduled for that mission [1,2]. The system gathers and reduces data from the experiment, flags "interesting" results, and proposes changes in the experiment both to exploit the in-flight observations and to stay within the time allowed by Mission Control for the experiment. These tasks must all be performed in real time. Two Apple Macintosh computers are used. The CLIPS- and LISP- based environments are layered above the Macintosh computer Operating System.*

*The "CLIPS-based" environment includes CLIPS and HyperCard. The LISP-based environment includes Common LISP, Parmenides (a frame system), and FRuleKit (a rule system). Important evaluation factors include ease of programming, performance against real-time requirements, usability by an astronaut, robustness, and ease of maintenance. Current results on the factors of ease of programming, performance against real-time requirements, and ease of maintenance are discussed.*

# A comparison of CLIPS- and LISP- based approaches to the development of a real-time expert system.

R. Frainier*, N. Groleau**, R. Bhatnagar**, C. Lam***, M. Compton*,
S. Colombano*, S. Lai**, P. Szolovits +, M. Manahan ++, I. Statler +++, L. Young**.

* Artificial Intelligence Research Branch, NASA-Ames Research Center, NAS Moffett Field, CA. 94035

** Man-Vehicle Laboratory, Massachusetts Institute of Technology, Cambridge, MA. 02139

*** Knowledge Systems Laboratory, Stanford University, Palo Alto, CA. 94034

+ Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA. 02139

++ Human-Computer Interaction Laboratory, NASA-Johnson Space Center, Houston, TX. 77058

+++ Aerospace Human Factors Research Branch, NASA-Ames Research Center, NAS Moffett Field, CA. 94035

## Abstract

*This paper describes an ongoing expert system development effort started in 1988 which is evaluating both CLIPS- and LISP- based approaches. The expert system is being developed to a project schedule and is planned for flight on Space Shuttle Mission SLS-2 in 1992. The expert system will help astronauts do the best possible science for a vestibular physiology experiment already scheduled for that mission [1,2]. The system gathers and reduces data from the experiment, flags "interesting" results, and proposes changes in the experiment both to exploit the in-flight observations and to stay within the time allowed by Mission Control for the experiment. These tasks must all be performed in real time. Two Apple Macintosh computers are used. The CLIPS- and LISP- based environments are layered above the Macintosh computer Operating System.*

*The "CLIPS-based" environment includes CLIPS and HyperCard. The LISP-based environment includes Common LISP, Parmenides (a frame system), and FRuleKit (a rule system). Important evaluation factors include ease of programming, performance against real-time requirements, usability by an astronaut, robustness, and ease of maintenance. Current results on the factors of ease of programming, performance against real-time requirements, and ease of maintenance are discussed.*

## 1. Introduction

When trying to develop an expert system for a real-world, real-time application, the following factors are important: ease of programming, performance, usability, robustness, and ease of maintenance. There are several alternative programming environments that might be used to build such an expert system. In this paper, we will examine our experience with two sets of tools applied to the same programming project, the Principal-

Investigator-in-a-Box (PI-in-a-Box) expert system. One of the tool-sets illustrates a LISP-based approach, and the other illustrates a rule-based approach. We introduce the project, describe the current status of the expert system with respect to each tool set, and compare their relative strengths and weaknesses.

Why use a LISP-based approach? LISP is a powerful, flexible language, and it has been widely used to build expert systems. However, some might argue that any system using LISP will be too slow, will not meet real-time performance criteria due to uncontrollable dynamic storage reclamation episodes (garbage-collection), and cannot be maintained due to a scarcity of LISP programmers. This paper offers some evidence contradicting these claims.

Why use a rule-based approach? Some might claim that experts use declarative knowledge. Therefore, a declarative representation of knowledge, and deductive inference based on that declarative representation of knowledge, is best. These claims are appealing, and this paper offers some evidence supporting the usefulness of such an approach.

Why not also consider C? We have not attempted a prototype directly in C. Prototyping in C is too time-consuming, and we do not believe we could have made good progress programming our expert system in C.

## 2. The PI-in-a-Box expert system

The PI-in-a-Box expert system is described in [3]. The system is designed to help scientist-astronauts do the best possible scientific experimentation in space given fairly severe time constraints and the need to work in areas outside their main specialty. The lack of time affects both pre-flight training in, and in-flight operation of, the experiment. The system will help the astronaut become a better scientific collaborator with the ground-based Principal Investigator who has designed the experiment by sharing with the astronaut observations about the quality and importance of the data as it is being generated in-flight. The system has several modules. These modules together will allow the diagnosis of data-collection problems, hypothesis monitoring and formulation (limited to an analysis of "interesting" data in the initial system), and determination and scheduling of the experiment's steps.

The Data Acquisition Module (DAM) and Data Quality Monitor (DQM) acquire data from the experiment, extract parameters from the data and interpret them, and also analyze the data to determine its quality with respect to the experimental apparatus. The DQM activates the Diagnostic and Troubleshooting Module when necessary.

The Diagnostic and Troubleshooting Module (DTM) helps the astronaut isolate and recover from data-collection problems during the experiment. It suggests various tests to isolate equipment faults. It also presents assessments of problem severity and possible recovery strategies with respect to the remaining experiment session time.

The Interesting Data Filter (IDF) module monitors subject-specific data from the experiment passed to it by the DAM and approved by the DQM. The IDF analyzes the data to determine its quality with respect to the human subject under evaluation and to decide if any pre-flight hypotheses are being violated. An unreliable subject is termed "erratic" and receives lower priority for the rest of the current experiment session as well as the next experiment session. A violated hypothesis causes the system to suggest confirmation as a prelude to (possibly) suggesting an alternate hypothesis to test.

The Protocol Manager (PM) module ensures that the best possible experimental protocol is always available for use. It also displays information for, and accepts information from, the user-astronaut. Corresponding to these two major functions, the PM has two logical components: the scheduling component called the Protocol Suggester (PS) and the human interface component called the Session Manager (SM).

The PS reacts by creating a new experimental protocol when:
* there is a predicted shortage of time, indicating a possible need to cut steps.
* there is a predicted excess of time, indicating an opportunity to add steps.
* an experimental subject provides interesting data, indicating a possible need to substitute steps that will collect more information about the interesting data.
* an experimental subject is sick or otherwise unable to participate, indicating a need to cut that subject's steps to substitute another subject and experiment.
* the user-astronaut desires to do so.

The SM displays the current state of the experiment including progress against the protocol, elapsed times, and the history of other sessions occurring earlier in the mission. The SM also displays procedural step-by-step checklists of experimental steps to be performed by both the subject and the user-astronaut. The SM updates the current protocol, elapsed times, and the history of other sessions occurring earlier in the mission in response to user-astronaut editing. The SM also offers a scratch-pad to allow the user-astronaut to record her/his thoughts. The user-astronaut can currently perform the following actions using the SM:
* Display the status of the current session. This includes a list of completed steps, the current step, and all pending steps. It also includes temporal information about the session and the current step.
* Display alternative protocols (better, maximal, and original).
* Display the history of other sessions occurring earlier in the mission.
This history is a list of all completed steps, including the subject and experimental condition used for each step.
* Display checklists for a given experiment step.
* Edit any protocol (usually on a line-by-line basis) and all temporal information known, and used, by the system.
* Replace the current protocol with any of the other available protocols (better, maximal, or original).
* Order a new set of protocols for consideration (by calling the PS).

The Protocol Manager module is more completely described in [4] and [5].

Two further modules are planned for future versions of the expert system, an Experiment Suggester and a (mission) Scheduler. The Experiment Suggester (ES) will work in conjunction with the IDF. Given a new hypothesis from the IDF, the ES will generate a set of tests that can be used to investigate the new hypothesis. The (mission) Scheduler will look at all of the sessions currently scheduled for all of the experiments in the mission (*e.g.* seven-day shuttle flight), and then suggest better schedules.

## 3. Implementation

As currently designed, the system will run on two Macintosh computers connected by a serial port. These machines will be referred to as the "AI-Computer" and "Data-Computer". The AI-Computer hosts a database, astronaut interface (or SM), DTM, PS, and IDF modules. The Data-Computer hosts the experiment interface, as well as the DAM and DQM modules. The implementation of these two modules on the Data-Computer is essentially complete (and will not be discussed further, except to point out that they are compatible with either CLIPS- or LISP- based modules on the AI-Computer). The Experiment Suggester and (mission) Scheduler will be hosted by the AI-Computer, but do not yet exist in either the CLIPS- or LISP- based implementations.

## 3.1 The AI-Computer

One version of the system is being developed on a Macintosh IIcx with 5MB RAM using CLIPS and HyperCard. HyperCard is used primarily as a tool to prototype the astronaut interface: the CLIPS interface alone is not rich enough for our needs. This version also uses the MacroMaker and QuicKeys INITs to facilitate transferring control between CLIPS and HyperCard. The system (including MultiFinder) currently resides in just over 4MB of RAM. The project documentation text, source code, and image files currently occupy about 6MB of hard disk storage.

A parallel version of the system is being developed on a Macintosh IIcx with 5MB RAM using Macintosh Allegro Common LISP (MACL), Parmenides (a frame-based knowledge representation tool from Carnegie-Mellon University (CMU)) and FRuleKit (a rule-based reasoning tool from CMU that complements Parmenides). The system (including MultiFinder) currently resides in just under 3MB of RAM. The project documentation text, code source and image files currently occupy about 3-1/2MB of hard disk storage.

Both versions fit comfortably on the Macintosh IIcx computers used for system development. We do not anticipate needing more than 8MB of RAM in the final system with either version.

The hardware/software for each system maps to the system architecture as seen in figure 1. Notice that an extra module, the "executive", is present in the CLIPS/HyperCard implementation. This module is used to control the call-out/call-back between HyperCard and CLIPS.

## 3.2 CLIPS-based Implementation

In the CLIPS-based implementation, the astronaut interface (or SM), has been prototyped in a single heterogeneous HyperCard stack. The system uses only one stack, and the interface includes most of the stack's four-to-five dozen cards (about a dozen of the cards are used as a database, and are discussed later in this section).

The interface contains about a dozen cards that are used as checklists - one card per type of experiment step. There are several cards that are used to hold explanations furnished from CLIPS reasoning: the appropriate explanation is displayed to the user-astronaut on request. Currently, 16 cards are used to display system "help" text for the user. There is a user scratch-pad card and a "main menu" card.

There is one card that serves as the main interface for both user and developer, and is viewed during most of the session. This card contains 34 text fields and 16 buttons which are used to display and control temporal information, display and control the experimental protocol, and to initiate certain procedures. It will be significantly restructured specifically for astronaut use before Space Shuttle Mission SLS-2 in 1992.

In the current implementation, the executive essentially consists of the HyperCard "idle" message. It uses idle to:
* read the serial port for data from the Data-Computer and update the data base;

* pass (temporary) control to the CLIPS-resident DTM, IDF, or PS modules (using the MacroMaker and QuicKeys INITs, as mentioned above.); and
* perform time keeping for the PM module.

The data base is divided between HyperCard and CLIPS. The HyperCard-resident portion consists of about a dozen cards. Half of these are used to store data from the experiment : one card is used for each step of the experiment that generates data. Although they are not present yet, a few cards will soon be used by the IDF module to store summarized data used in testing hypotheses. There is also one card used for each experiment session to summarize the results of that session (including the partial results of the current experiment session). Two cards are used to store data used "globally" by several of the modules. The CLIPS-resident portion consists of about 120 base facts. About one-third of these duplicate information from the HyperCard-resident data base. About one-half of the facts represent domain knowledge. The remaining one-sixth are control facts, global variables, pathnames, etc. During the typical operation of the PS, there are about 200 facts in the data base at any one time (out of 120 base facts and 400 generated facts).

The DTM and IDF modules each presently consist of about 60 CLIPS rules. The IDF rules are all contained in one file, and will have to be broken up as the file size has reached the 32K CLIPS size limit. The PS module consists of about 200 rules in seven files.

## 3.3 LISP-based Implementation

The LISP-based implementation is not nearly as complete as the CLIPS-based implementation. There is no interface (aside from the MACL "Listener" window and pull-down menu items). Further, the DTM and IDF modules have not yet been started. The data base and PS module have both been 50% completed. A much simpler executive is used in the LISP-based implementation as FRuleKit is well-integrated with Parmenides, which in turn is well-integrated with MACL. The data base and the PS module are described below.

The data base is a combination of Parmenides frames (object/property/ value notation) and a few LISP global variables. Two major frame taxonomies account for the majority of the frames used: the "protocols" taxonomy contains the current protocol, all previously-completed protocols, and the better, maximal, and original protocols; the "steps" taxonomy contains information describing each experiment step. The data base is integrated in that LISP procedures can operate on frames, and can be used within rules. Also, rules can match on frame-stored information. Thus, no duplication of data is necessary.

The PS module consists of 10 LISP procedures and 13 FRuleKit rules. It is expected to scale-up linearly to about 20 procedures and 26 rules when completed.

## 4. Ease of Programming

What is meant by "ease of programming"? A tool or language facilitates programming if it is easy to learn and use, includes help for debugging, supports a "rapid prototyping" programming style, allows good control over the concepts (data types) of the language, and can be used to do the things the programmer needs the program to do. The CLIPS-based implementation primarily uses two distinct programming tools: CLIPS and HyperCard. The LISP-based implementation primarily uses three integrated programming tools: MACL, Parmenides, and FRuleKit.

## 4.1 CLIPS Programming

CLIPS is fairly easy to learn. It is also fairly easy to use. It is best at pattern matching and deduction based on pattern matching. In PI-in-a-Box (an event-driven system), it is not as good at procedural control. CLIPS wants to keep control until the agenda is empty and tends to fire any rule in the data base any time it can. However, at any given time, we want to have only "relevant" rules considered as candidates for firing. This partitioning has several benefits: It (potentially) limits the amount of time committed to a particular chain of deductive inference; it makes pattern matching more efficient; it makes it easier to verify and validate the results of a chain of deductive inference; and it makes debugging easier. We use the standard CLIPS programming idioms for this purpose: carefully thought-out rule saliences and control facts. The chief disadvantages of these idioms are: the need to code rules whose only function is control; as more of the system is implemented (and functionality is extended), groups of saliences may need to be reset; and, as the number of control facts needed grows, a rule's Left-Hand-Side (LHS) begins to get cluttered. Also, the PI-in-a-Box team members had specific module assignments. When these modules were first integrated in CLIPS there were control fact collisions, and some rules had to be rewritten. Another procedural control problem is iteration. Rules refract to keep the same instantiation from firing repeatedly. However, sometimes iteration is desired. This generally means continually retracting and reasserting an iteration control fact (another standard idiom) which adds clutter to the rule's LHS and Right-Hand-Side (RHS).

There is fairly good support for debugging in CLIPS. On our Macintosh machines, the relatively small screen area (13-inch monitor) can sometimes make viewing the debugging information difficult. As mentioned earlier, the system has about 320 rules and 200 current facts in working memory. A typical run of the PS module includes 300 rule firings. This means that it can be difficult to see what is happening without a fair amount of scrolling. Compounding this is the long length of many of the facts in PI-in-a-Box: the PS module was written in version 4.2 of CLIPS (before deftemplate).

CLIPS is a good rapid prototyping tool. It is easy to write new rules and to load them quickly into working memory. It is also easy to manipulate facts in the data base. All deductions, computations and manipulations necessary to support the DTM, IDF, and PS modules can be accomplished with CLIPS.

## 4.2 HyperCard Programming

For a programmer already familiar with the Macintosh, HyperCard is very easy to learn. It is also easy to use. It has a fairly good control mechanism and most of the things that the PI-in-a-Box system needs HyperCard to do can in fact be done. However, it does not have good support for debugging. [We recently acquired an add-on tool for debugging HyperCard, but have not used it enough yet to report on it.] There is no "single-stepper" or even a graceful way to "break" a computation. If a script with looping forms is manipulating text fields and does not exit due to a programming error, it is sometimes impossible to quit HyperCard without corrupting the stack. In pathological cases the entire stack must be discarded. HyperCard has a tendency to save the state of the stack at the slightest provocation. This is good when you are using it to permanently store data, but not always helpful during development, test, and debugging. Because of the frequent auto-save, and lack of graceful breakpoints, a copy of the current stack should be used for HyperCard prototyping (with subsequent backcopying of debugged code). Support for the creation and editing of HyperTalk scripts is fair. The biggest problem is the script window: only one can be open at a time. Another problem concerns stacks: only one can be open at a time. Thus, copying debugged code from a test stack to a stack that represents the application can be tedious. One possible solution is to have two HyperCard applications running under MultiFinder, then one stack can be open in each RAM-resident application.

The above limitations aside, HyperCard is a good rapid prototyping tool. An interface can be prototyped in several days' time. It supports several Graphical User Interface (GUI) input styles: text in fields, message box, pop-up menus, check-lists (button hilite), radio buttons, and "vanilla" buttons. The HyperTalk language gives the programmer good control over on-screen graphics. However, there is a problem with dialog box positioning in HyperCard. No positioning control exists at the HyperTalk level. Dialogs pop up in the upper center of the card. One possible solution is to reconfigure the interface such that the least useful information occupies the "real-estate" dialogs pop-up over. Due to the limited card size on the screen, and the desirability of the upper center of the card for displaying useful information to the user (After all, why does the dialog go there by design?), this solution is not feasible. A better solution is to go to the Mac toolbox and reset the dialog pop-up position.

HyperCard is serving as a data base as well as a GUI, but it is less useful in this role. One limitation is the interaction between message passing and the addressability of information stored in text fields during rapid prototyping. Elaboration of this point is beyond the scope of this paper.

While HyperCard's automatic save feature is a disadvantage when prototyping, it is an advantage as a data base resiliency mechanism at runtime. If the program or machine "crashes", the previous state of the system is largely intact.

The degree of integration of the tools impacts the degree of integration of the database. We have already mentioned that some facts are duplicated in CLIPS. Another impact is the care that must be taken when programming the CLIPS-HyperCard interface. While not especially difficult, the lack of integration between the two tools requires attention to the syntax of the forms passed between the two tools.

## 4.3 LISP Programming

Common LISP is easy to learn (it is widely taught as an introductory programming language) and use. Parmenides and FRuleKit are easy to learn and use if already familiar with rule-based and frame-based programming, though the documentation is barely satisfactory. The programmer has good control over the tools and the PI-in-a-Box system's needs are covered. Still, Parmenides is incomplete as a language for Object Oriented Programming (OOP): not all of the expected supporting functions exist. This is not a critical limitation, as the source files are available, and the language is easy to extend by writing LISP-based procedures. Writing rules is more difficult in FRuleKit than in CLIPS. There are more constraints on the LHS and RHS clauses, and they are not all "reasonable". [Usually, "unreasonable" constraints imply either an incomplete implementation or an optimization.] Still, this is not a major difficulty as there will be far fewer FRuleKit rules than CLIPS rules (Two dozen instead of 200 in the PS module).

There is adequate support for debugging in LISP. While the debugging environment is not as rich as a LISP Machine's [6], it has the standard trace, step, break, macroexpand, backtrace, inspect, and eval facilities. As noted above with CLIPS, the relatively small screen area of our machines can sometimes make viewing the debugging information a chore.

The LISP-based tool set is a good rapid prototyping environment. There is fairly good support for OOP during prototyping, although Parmenides is missing a graph-based inspection/editing facility for the frame taxonomy, and there are some limitations on the order in which objects can be introduced in the hierarchy. Also, it is quite likely that the interface will not be as easy to prototype with MACL as with HyperCard.

In this section we have discussed programming in CLIPS, HyperCard, and LISP. The differences noted are not always comparable, but we conclude that both systems (CLIPS/HyperCard and LISP) can be used to quickly program a prototype.

## 5. Performance

PI-in-a-Box is a real-time expert system. It must receive, analyze, and act on the data that is generated by the experiment quickly enough to be of use to the user-astronaut. Each data-producing step in the vestibular physiology experiment consists of six 30-second trials. The system has 40 seconds between the end of the fifth trial of one run and the start of the first trial of the next run to perform several actions. It must check the data for correctness and agreement with the current hypotheses, and (if necessary) suggest new protocols. The team has concluded that the time needed to suggest new protocols should be no more than 30 seconds.

In the current CLIPS/HyperCard version, it takes about 70 seconds to suggest new protocols. Once the need to suggest new protocols is established, HyperCard writes information from its fields to a file on disk and passes control to CLIPS. This alone takes just over 20 seconds. CLIPS then resets its data base, reads the current information into working memory, reasons over it, and writes out newly-suggested protocols with associated explanations to another file on disk. Typically, about 300 rules are fired and 400 facts are asserted. This takes about 20 seconds. HyperCard resumes control and reads the newly-suggested protocols with their supporting explanations into text fields. This takes just under 30 seconds. The current implementation takes too much time!

One way to reduce the needed time is to recode the CLIPS rules to eliminate some of the rule firings and to exchange data between CLIPS and HyperCard using RAM instead of disk files. The PI-in-a-Box team feels that this might shave about 10 seconds off of the current 70-second cycle time. Another idea is to code some of the routines directly in either C or Pascal to try to save more time. It is unclear if the current tools can meet run-time performance requirements. It is clear that tool integration (chiefly communication between the tools) is THE critical issue.

The current LISP-based version is not nearly as complete as the CLIPS-based version. Only about half of the work needed to suggest new protocols is currently represented in the LISP-based version's code. Still, even though a fully-comparable system does not exist, a partial comparison can be made. In the current LISP-based version, running interpreted LISP code, it takes about 4 seconds to suggest new protocols nine-out-of-ten times. The tenth time, it takes about 8 seconds (four seconds of which represents a "garbage collection"). If these times scale-up linearly with the size of the system, a LISP-based system two-and-one-half times larger would usually take 10 seconds to suggest new protocols, but would sometimes take 20 seconds to suggest new protocols. These times should be somewhat improved by using compiled, in place of interpreted, code. Thus this system can probably meet the performance requirements of PI-in-a Box.

The LISP-based version offers better performance than the CLIPS-based version by a wide margin. The primary reason is that the CLIPS-based version is hurt by the lack of integration with HyperCard. Another reason is that the LISP-based version is helped by its ability to use procedures for control, and both a frame taxonomy AND deductive rules for inference. There is no need to fire a rule to write a line of text to the screen. There is no need to fire a rule to conclude that object "oscilloscope- check.5" is a step to be scheduled. Rules are only used when needed for deductive inference. As future work on CLIPS extends its representation abilities, and as work on integrating CLIPS and HyperCard lets them exchange information efficiently, the current differences in performance will be narrowed.

## 6. Maintenance

Maintenance is related to ease of programming (discussed above) in that tools which are used to build, test, and debug a program are also used to maintain and extend it. Maintenance by the end-user is not practical with either of the two sets of tools currently being used on PI-in-a-Box. Maintenance by a general-purpose computer support person is feasible with the CLIPS-based implementation. Potential drawbacks are the lack of debugging support in HyperCard, the restriction of only one script editing window open at a time, and the limited ability to control and extend the basic data types (*e.g.* dialog box positioning and use). Maintenance of the CLIPS-based implementation by a programmer who is an expert in the use of CLIPS and HyperCard, or by the expert system's authors, is also feasible. The "potential drawbacks" just mentioned are somewhat less important here. Maintenance by a general-purpose computer support person familiar with OOP is also feasible with the LISP-implementation. If this person is not familiar with the concepts of OOP, she will have greater difficulty. Finally, maintenance of the LISP-based implementation by a programmer who is an expert in the use of LISP and frame-based and rule-based tools is feasible. The flexibility of LISP is particularly valuable when extending an expert system.

There are other factors, not directly related to ease of programming, to consider as well. These factors are more of a concern to the program maintainer than the program author (though hopefully the author considers them as well out of concern for the maintainer). They include:
* the source of the tool, including the availability of technical support (especially when extending the system) and ability to obtain new versions that match new versions of the hardware's operating system.
* the total number of tools used, because the ability to obtain new versions that match new versions of the hardware's operating system applies to each tool in the system.
* the ease of conversion from the development software configuration to the runtime software configuration.

PI-in-a-Box has not yet begun the process of "tuning" the rules and procedures or converting the system to a runtime configuration, so there is little to report here on that topic. However, some comments can be made about the source of the tools currently used. Apple seems to be committed enough to HyperCard for it to be available on upcoming operating system releases. Also, a number of spin-off products to extend HyperCard and HyperTalk are starting to appear. Likewise, NASA seems committed to maintaining and extending CLIPS (*e.g.*, deftemplate, COOL, user group meetings.). The LISP-based tools cannot make the same claims. The most recent version of MACL is only available through Apple Programmer's and Developer's Association (APDA). Parmenides and FRuleKit are "research quality" (as opposed to "commercial quality") tools, and are only available through CMU. Still, Parmenides and FRuleKit have been available for four years, and have been updated regularly. Further, with the LISP-based source files, it is possible to port those tools (except perhaps the interface) to another LISP environment.

### 6.1 PM Module Maintenance

The PM module of the PI-in-a-Box system was prototyped by one programmer through May, 1989, and has been maintained by another from October, 1989 to the present time. The maintainer was very familiar with the Common LISP HOL, OOP programming paradigms, and rule-based programming paradigms. The maintainer was also familiar with the C HOL and with the Macintosh computer. The maintainer had not used HyperCard, CLIPS, MACL, Parmenides, or FRuleKit. The CLIPS/ HyperCard version's documentation for the PM module was fairly thorough [5]. Maintenance to date has included bug-fixes, porting to new versions of the tools and operating system, and functionality extensions. In addition to observations already made in the ease of programming section, above, the following can be said. Maintenance in HyperCard can be tedious, as each field, button, and card has fragments of code. It is difficult to browse the source code and follow the thread of procedural flow. This is especially true since only one script window can be open at a time. Maintenance in CLIPS is easier. Multiple windows can be open to help the programmer follow the chain of inference. One difficulty concerns intermediate data lists. A fair number of the rules are concerned with manipulating these intermediate lists. Pattern-matching differences are sometimes subtle, and it requires careful thought to discern all of the possible cases being considered.

In general, maintenance is easiest in CLIPS, and most difficult in HyperCard. There are potential maintenance problems with respect to the LISP-Parmenides-FRuleKit implementation, as the separate tools may not follow operating system upgrades as quickly as desired.

## 7. Conclusions

The PI-in-a-Box team has found both CLIPS and HyperCard good for rapid prototyping. Also, the PI-in-a-Box team has successfully maintained the CLIPS/ HyperCard-based PM module. However, the performance of the system has been disappointing.

The PI-in-a-Box team has also found the combination of LISP, Parmenides and FRuleKit good for rapid prototyping. No direct evidence was presented for or against the question of maintainability, but the question had the issue of programmer availability at its roots. About half of the present team's members have significant LISP programming experience. Maintenance will not be a problem in the near future. Finally, the LISP-based system is faster than the CLIPS/HyperCard system, primarily due to the factor of tool integration (inter-tool communication). Thus, the LISP-based system will probably be fast enough to meet the real-time performance requirements of the vestibular physiology experiment, while the CLIPS/HyperCard system will not. This result highlights the need for tightly-integrated programming environments.

## Acknowledgements

The project was made possible by NASA grant NCC 2-570 and RTOP 506-47-11 for "Crew Station Design", respectively from the AI Research Branch and the Human Factors Division at NASA-Ames. The Stanford University Knowledge Systems Laboratory, the Carnegie-Mellon University School of Computer Science, and Apple Corporation also provided generous support.

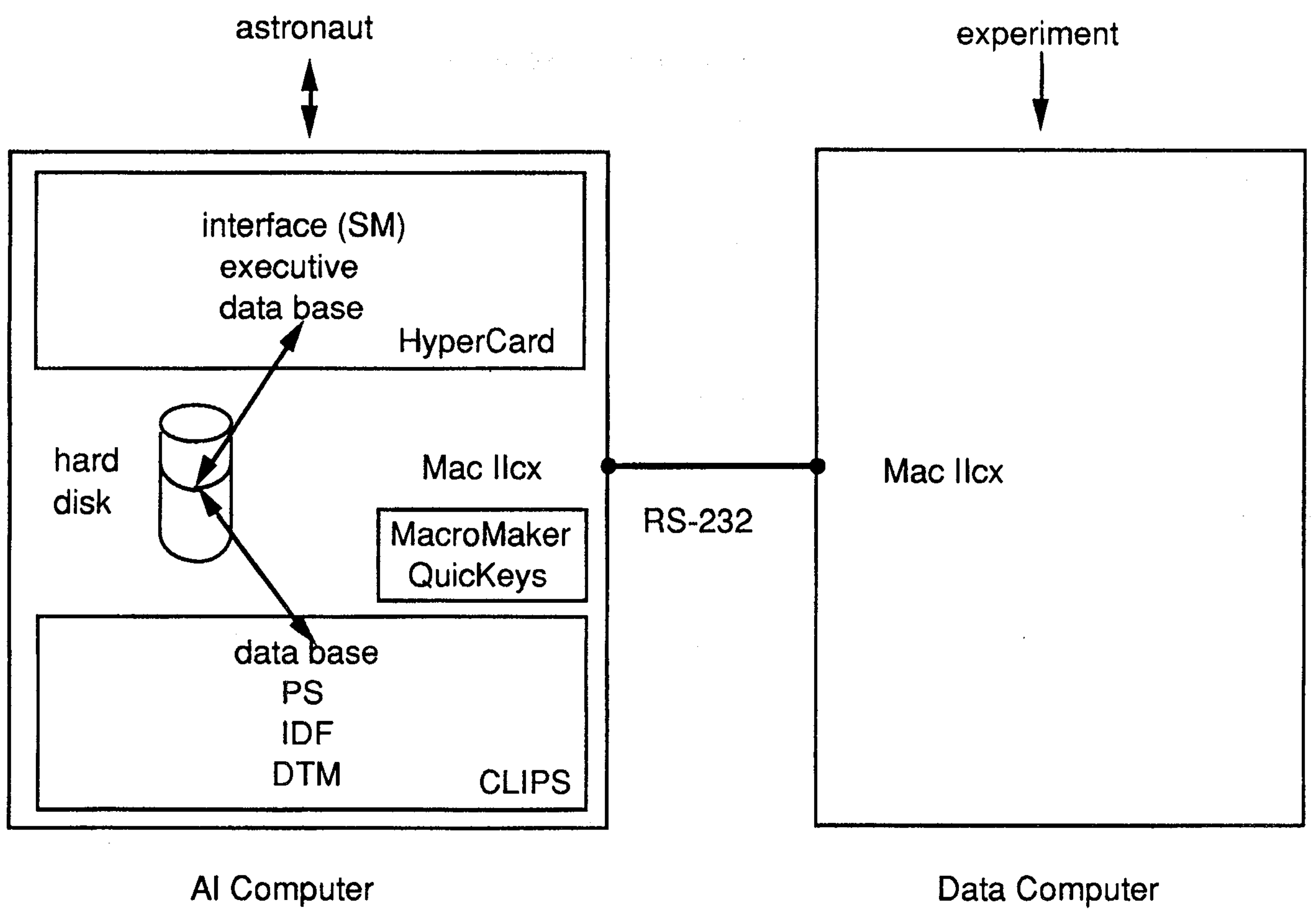


Figure 1a. CLIPS-based System Architecture

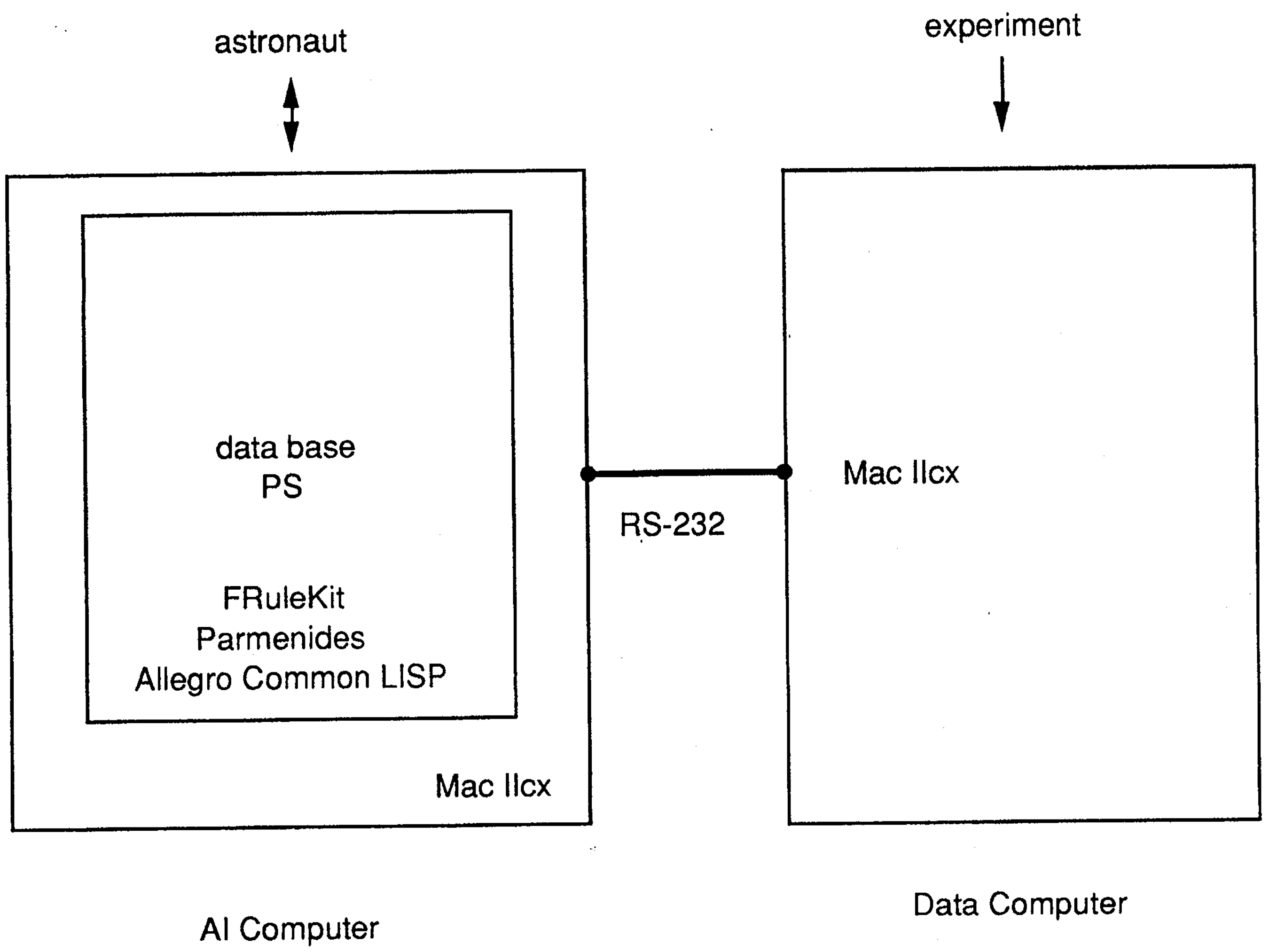


Figure 1b. LISP-based System Architecture

## References

[1] Young, L.R.; Oman, C.M., Watt, D.G.D., Money, K.E., Lichtenberg, B.K., Kenyon, R.V., Arrott, A.P. "M.I.T./Canadian Vestibular Experiments on the Spacelab-1 Mission: 1. Sensory Adaptation to Weightlessness and Readaptation to One-G: an Overview," *Experimental Brain Research 64: 291-298,* 1986a.

[2] Young, L.R.; Shelhamer, M., Modestino, S.A. "M.I.T./Canadian Vestibular Experiments on the Spacelab-1 Mission: 2. Visual Vestibular Interaction in Weightlessness," *Experimental Brain Research 64: 299-307,* 1986b.

[3] Young, L.R.; Colombano, S.P.; Haymann-Haber, G.; Groleau, N.; Szolovits, P.; Rosenthal, D. "An Expert System to Advise Astronauts During Experiments", *Proceedings to the 40th Congress of the International Astronautical Federation,* October 1989.

[4] Haymann-Haber, G.; Colombano, S.P.; Groleau, N.; Rosenthal, D.; Szolovits, P.; Young, L.R. "An Expert System to Advise Astronauts During Experiments: The Protocol Manager Module," *Proceedings to the Conference on Space Automation and Robotics,* July 1989.

[5] Haymann-Haber, G. "PI-in-a-Box: The Design of an Expert Protocol Manager," *Knowledge Systems Laboratory Report 89-55*, Stanford University, 1989.

[6] Moon, D.; Stallman, R.; Weinreb, D. "LISP Machine Manual," Massachusetts Institute of Technology, 1983

# KNOWLEDGE BASE RULE PARTITIONING DESIGN FOR CLIPS

Joseph D. Mainardi and Dr. G.P. Szatkowski

GENERAL DYNAMICS
*Space Systems Division*

5001 Kearny Villa Road
San Diego, CA 92138
(619) 496-7093

## ABSTRACT

This describes a knowledge base (KB) partitioning approach to solve the problem of real-time performance using the CLIPS AI shell when containing large numbers of rules and facts. This work is funded under the joint USAF/NASA Advanced Launch System (ALS) Program as applied research in expert systems to perform vehicle checkout for real-time controller and diagnostic monitoring tasks.

The Expert System advanced development project (ADP-2302) main objective is to provide robust systems responding to new data frames of 0.1 to 1.0 second intervals. The intelligent system control must be performed within the specified real-time window, in order to meet the demands of the given application. Partitioning the KB reduces the complexity of the inferencing Rete net at any given time. This reduced complexity improves performance but without undo impacts during load and unload cycles. The second objective is to produce highly reliable intelligent systems. This requires simple and automated approaches to the KB verification & validation task. Partitioning the KB reduces rule interaction complexity overall. Reduced interaction simplifies the V&V testing necessary by focusing attention only on individual areas of interest.

Many systems require a robustness that involves a large number of rules, most of which are mutually exclusive under different phases or conditions. The *ideal* solution is to control the knowledge base by loading rules that directly apply for that condition, while stripping out all rules and facts that are not used during that cycle. The *practical* approach is to cluster rules and facts into associated "blocks". A simple approach has been designed to control the addition and deletion of "blocks" of rules and facts, while allowing real-time operations to run freely. Timing tests for real-time performance for specific machines under R/T operating systems have not been completed but are planned as part of the analysis process to validate the design.

## BACKGROUND

The Air Force and NASA have recognized that our nation's current suite of launch vehicle systems has a number of problems making them inadequate for the projected needs after the late-1990's. High costs of above $2,000/lb of payload delivery, a low reliability, poor resiliency (standdowns of many months for current expendables), and limited launch rate capacity are reasons behind the joint USAF/NASA effort for an operational ALS and Shuttle 'C'. These will serve the commercial and DoD mission models beginning in 1998. In order to meet the goals of $300/lb and launch rates as high as 50 missions annually, these systems and their associated ground operations segment must be made as autonomous as possible, while at the same time improving reliability and safety. Under the ALS Program, a study was initiated to explore the use of knowledge-based system (KBS) techniques for the purpose of automating the decision processes of these vehicles and all phases of the ground operations segment by assessing the feasibility, benefits, and risks involved.

An expert decision aid is a software approach to solving particular problems that are constantly changing over time and are complex or adaptive in behavior, the opposite of an analytical problem that is basically deterministic. Examples of these types of problems are: the re-scheduling of a vehicle checkout due to a damaged cable; or, determining if a system is indeed faulty given conflicting sensor readings. These heuristic problems require a depth of knowledge and experience (art rather than science) to form solutions quickly. Expert systems embody that collection of knowledge and experience in modular pieces that are rules and facts that describe the proper thought process for a given set of circumstances arrived at by any path. It is this modular independence that makes expert systems attractive. The incremental improvement of knowledge and experience can be built and tested readily without re-testing the rest of the software system, unlike conventional software that is difficult to maintain in a day to day changing environment.

## INTRODUCTION

This work was funded under a joint effort on the part of NASA and USAF for the Advanced Launch System (ALS) program, as applied research in expert systems. The implementation of robust, real-time expert systems is considered to be an important technological addition to the design and development of the ALS vehicle. In a document released by the Aerospace Industries Association of America (AIAA) in April 1989, Artificial Intelligence technology was identified as one of the eight key technologies for the 1990's. This selection was founded on the basis of greatest potential payoff, broadest application base, and highest leverage. With the success of non-real-time expert systems already established, it was important to produce real-time expert systems that would exhibit the benefits of the technology within the scope of the ALS program.

In attempting to select a candidate expert system development tool to implement robust, real-time production systems, speed was the most obvious factor considered. A leading candidate was the "C Language Integrated Production System" (CLIPS), developed at the Artificial Intelligence Section (AIS) at NASA/Johnson Space Center. Based on the fact that CLIPS is written in C (considered one of the faster programming languages), and its lean implementation, it was determined that it held the highest potential of all tools reviewed, for use in real-time expert systems applications. Another factor was that of software flexibility. CLIPS was chosen because of the nature of the tool. It is public domain software so therefore the source code can be modified. Its C implementation permitted easy imbedding into the application system and easy integration with most graphics libraries and other languages (for ex. LISP). And finally, it provided a hardware independent inference vehicle, allowing development on lap-tops and run-time multi-system integration on high performance workstations and multi-processor systems. Overall, the belief was that CLIPS would be the best available tool to realize robust, real-time expert system implementations within the ALS program.

One of the major problems facing robust, real-time expert systems is that there have not been enough successful systems to make a significant impact on the commercial or military community. This was the same type of problem that initially hindered the success of non-real-time expert systems. An earlier attempt at creating a modification of CLIPS for use in real-time applications, the Portable Inference Engine (PIE) by T. Le and P. Homeier of the Aerospace Corporation, has not yet been widely accepted by the CLIPS user community.

The approach that was chosen for the ALS program was to create a rule partitioning approach that would not require modification of the actual CLIPS code, but would instead be application-specific CLIPS code that could be used to reduce the knowledge base (the number of active rules and facts). This is effectively knowledge-base "control blocking". The examination of rules that do not fire during an expert system application's cycle is the unfortunate overhead that expert systems typically carry. There a only a few expert system development tools that implement the concept of knowledge base rule blocking, where production rules are organized into logical arrangements to facilitate better control of the execution of those rules. An example of this approach is found in IBM's Expert System Environment (ESE). The challenge was to create a similar rule partitioning approach in CLIPS. This would then permit the development of robust, real-time controller applications for use in the ALS program.

## APPROACH

The approach taken to achieve real-time response using CLIPS was to control the number of production rules that would appear in a real-time window (0.1 to 1.0 seconds) during any cycle, without any significant degradation in performance - i.e. continual real-time operations. The method that was selected was one that would remove blocks of production rules and associated facts that were not being used during a real-time cycle and would add any blocks of production rules and facts that would be required during that same cycle. This is a procedure that is automated in some expert system development tools, but this automation has overhead that does not permit expert systems to perform in real-time. As an example, ESE uses "Function Control Blocks" to control rule sets. This is an effective method of controlling production rule examination, yet it does not permit real-time applications to be implemented.

In developing the CLIPS approach, overhead costs were unavoidable. The first cost was that each production rule would carry the overhead of an associated fact, in order to facilitate production rule removal. Each production rule is assigned a fact whose value is the rule name, which is then referenced by an **excise** statement. It was determined that this was the simplest and most efficient approach. The next cost was that production rule removal requires a CLIPS rule that will remove both the production rule and the associated fact. By using an **excise** statement to remove the rule and a **retract** statement to remove the fact (a pointer to the rule), all traces of the production rule are removed from the real-time cycle. It was determined that this was, again, the simplest and most efficient approach.

Another factor contributing to the overhead is that each production rule partition should (conceptually) be stored in an external file; I/O calls to external files are a major contributor to system performance degradation. A partial solution to this problem is to group together blocks of production rules that have common characteristics, and store the grouped rules in fewer external files. This will greatly reduce the number of I/O calls required in a real-time cycle, and therefore speed up the execution of the expert system application. To further reduce the overhead of excessive I/O calls, production rule partitions should be organized in an efficient manner. This can be done using a variety of different methods. Using mutual exclusivity of production rule partitions is very important in reducing I/O calls. By using mutually exclusive production rule partitions, all extraneous test level rules (see below) can be avoided. When organizing test level rules, attempt to organize production rule partitions in a manner that will not attempt to load duplicate production rules. This task can be performed by analyzing all conditions that are used to load each production rule, categorizing those rules, and putting all of the similar production rules in categorized external files. For rule removal, the simple rule is to remove only the production rules that are not needed for the upcoming real-time cycle. In other words, don't leave a production rule in the rule base if it's not needed during that cycle; it can be loaded at the next appropriate cycle.

Finally, each test level check, for both production rule block removal and loading, is in itself a rule. These test level rules can be reduced in number by finding conditions that are common within the application's testing conditions and incorporating them into fewer test level rules. When used across an entire expert system application, this type of control will greatly reduce the number of test level rules. Unfortunately, the test level rules cannot be removed. Therefore, if a test level rule will not remove at least three production rules, it is probably not efficient to execute that test level rule.

The test level rules must be executed at the beginning of each cycle. Therefore, they should be assigned the highest salience possible. In addition, these test level rules should be ranked, with the basis of the ranking taking on the characteristics of the application. For example, a system that has mutually exclusive blocks of production rules should have the mutual exclusivity test level rules examined first, and then examine the explicit test level rules. Test level rules remove subsets of production rules loaded during the previous cycle, and add production rules needed for the current real-time cycle. The test level rules can also turn on and off any facts generated by test level rules. This will further control production rule loading and removal. These facts can also be used to preserve rules that must stay resident in the production rule-base for multiple cycles. Typically, a test level rule with a salience of 10000 will assert facts that will have an effect on test level rules with a salience of 9999 and below. The rules with the salience of 9999 will then assert facts that will have an effect on test level rules with a salience of 9998 and below. This procedure will continue until all test level rules have been examined. The production rules that remain in the knowledge base for that cycle will then execute.

Figure 1 gives an example of loading and removing a "control block" (a collection of rules and facts).

## CLIPS Knowledge Base Partitioning Scheme

**Example: actual CLIPS code, loaded into a file (remove_rulea.clp)**

```
(deffacts x (x 0))
(deffacts rulea_facts (rulea a0) (rulea a2))
(defrule a0 (x 0) => (fprintout t "Answer is 0" crlf))
(defrule a1 (x 1) => (fprintout t "Answer is 1" crlf))
(defrule a2 (x 2) => (fprintout t "Answer is 2" crlf))
(defrule a3 (x 3) => (fprintout t "Answer is 3" crlf))
(defrule a4 (x 4) => (fprintout t "Answer is 4" crlf))
(defrule remove_rulea (?kptr <- (rulea ?kname) =>
  (fprintout t "Going to remove rule " ?kname crlf) (excise ?kname) (retract ?kptr))
```

**Procedure:**

```
>(load "remove_rulea.clp")
>(reset)
>(facts)
f-0   (initial-fact)
f-1   (x 0)
f-2   (rulea a0)
f-3   (rulea a2)
>(rules)
a0
a1
a2
a3
a4
remove_rulea
```

[continued next column]

```
>(run)
Going to remove rule a2
Going to remove rule a0
2 rules fired
>(facts)
f-0   (initial-fact)
f-1   (x 0)
>(rules)
a1
a3
a4
remove_rulea
```

Figure 1 —CLIPS Knowledge Base Partitioning Scheme

## CONCLUSIONS

There is significant interest in developing and implementing robust, real-time expert systems applications for the ALS Program. Real-time expert systems show great promise in reducing costs for ALS vehicle launches. With small rule sets of twenty to fifty rules and a real-time operating system, the performance of the rule partitioning approach was well within the limits of the one second real-time window. Future timing tests performed for large rule sets under a real-time operating system, will demonstrate the ability of CLIPS to perform in real-time.

There are additional benefits of the knowledge base rule partitioning approach. Using the approach outlined above, the application of generic verification and validation techniques for expert systems are more likely to succeed than with traditional expert systems. By efficiently grouping the production rules, modular testing and documenting of modifications and enhancements are easier to perform than non-modularized expert systems applications. In addition, each partitioned production rule set can be independently verified and validated. This is a significant advantage over expert systems with non-partitioned production rules, and can ultimately lead to lower expert system maintenance and enhancement costs. When compared to non-partitioned rule bases, modularized production rule bases are easier to document, and subsequently the documentation is better. Better documentation normally results in lower maintenance and enhancement costs. There is strong evidence that the ability to verify and validate expert systems will be a major factor in their expanded use in the military and commercial community.

## B6 Session:
## Medical, Biological, and Agricultural Applications

# GENEX: A KNOWLEDGE-BASED EXPERT ASSISTANT FOR GENBANK DATA ANALYSIS

Sajeev Batra
*Department of Computer Science*
*Bradley University*
*Peoria, IL 61625*

and

Mark A. Macinnes
*Life Sciences Division*
*Los Alamos National Laboratory*
*Los Alamos, NM 87545*

Abstract

We describe a knowledge-based expert assistant, GENEX (Gene Explorer), that simplifies some analysis of Genbank data. GENEX is written in CLIPS (C Language Integrated Production System), an expert system tool, developed at the NASA Johnson Space Center. The main purpose of the system is to look for gene start site annotations, unusual DNA sequence composition, and regulatory protein patterns.

## 1 Introduction

Automated expert assistance simplifies the analysis of any large database, such as Genbank. Genbank, a molecular biology database is a library containing genetic entries for various species. Each entry contains useful information about a gene, as well as the DNA sequence for that gene. The typical length of the DNA sequence of a gene is about 20,000 nucleotide bases. Genbank entries are compiled by the Los Alamos National Laboratory. A version of Genbank, which we used, is installed on the FLOVAX network in the Life Sciences Division at Los Alamos.

For our analysis of Genbank data, we developed a knowledge-based expert assistant, named GENEX. Like Genbank, GENEX is on the FLOVAX network. In addition, a DNA sequence analysis software package, developed by the Genetics Computer Group (GCG) at the University of Wisconsin at Madison, is installed on FLOVAX. GENEX supervises the execution of a set of GCG programs.

This paper will focus on what the expert system does, as well as its layout. GENEX is partitioned into three knowledge bases (KB's) in a hierarchal structure. It looks for gene start site annotations, unusual DNA composition, and regulatory protein patterns. Like GENEX, the rest of this paper is divided into three sections.

## 2 The Fetch phase

In this phase GENEX supervises the GCG program Fetch to pull selected genetic entries from Genbank. Then GENEX scans the entries to find certain keywords to help it look for mRNA start site annotations. Once the start sites are found they are stored in the fact list, where they will be used later by the other knowledge bases. Many inconsistencies with start site annotations were found here. Therefore, the annotation searching is very general.

## 3 The Composition phase

In the Composition phase, we are trying to locate so called genomic islands in the DNA sequence. Genomic islands, defined as regions of considerably biased dinucleotide content, have ranges from 200 to 1000 bases. There are four different bases in a DNA sequence: A, C, G, and T. A dinucleotide is made up of two bases, for example: AG, GT, CC, etc. Therefore, one can see that there are sixteen possible dinucleotide combinations.

The second knowledge base uses a modified version of the GCG program composition, which examines DNA dinucleotide frequencies in the selected gene entries. The dinucleotide statistics are taken on three sections of the promoter region in the DNA sequence: upstream, downstream, and total. For example, in Table 1, the mRNA start coordinate is found to be at position 415. Therefore, the downstream coordinates are from 415 to 830. Likewise, the upstream coordinates are from 0 to 415. And the total promoter site region coordinates are from 0 to 830, or simply the upstream region and the downstream region. A maximum of 500 bases are examined by GENEX in the upstream and downstream region.

Table 1 shows an example of what the Composition phase yields for the ERCC1 gene, a genbank entry. In Table 1, the dinucleotides are on the left hand side are. Beside them are their frequency rankings for the total, upstream, and downstream region. These frequencies are taken from the

**Table 1: Composition KB Output for ERCC1 gene**

```
ERCC1.dat
mRNA Start Site at 415
Total Coord.        0      830
Down Coord.       415      830
Up Coord.           0      415


Dicnucleotide     Total     Up    Down        Delta-R
         TC          9       8     13            -5
         CG          8      12      6            +6
         CT          6       1      8            -7
Top 3    GG          1       3      1            +2
Top 3    GA          2       4      2            +2
Top 3    AG          3       2      4            -2
Top 3    GC          4       5      3            +2
Top 3    CT          6       1      8            -7

-----
```

three regions and are usually variable, sometimes having dramatic shifts. These dramatic shifts are what make the DNA composition interesting. A certain dinucleotide will show up on the table if the absolute value of the Delta-R value is 5 or greater. Delta-R is defined as the difference of the frequency ranking in the upstream region and the frequency ranking in the downstream region. Another way a dinucleotide can show up in the table is if the frequency ranking in any region is in the top three.

## 4 The Sp 1 Phase

In this phase, GENEX looks for Sp 1 DNA binding sites, which can be identified by patterns of about ten bases in length. The transcription factor protein Sp 1 binds to certain C-rich and G-rich DNA motifs, presumably located around gene start sites. Note that the gene start sites of each gene are located in the fetch phase.

The third knowledge base recognizes certain patterns, identifying Sp 1 sites, in the DNA sequence. The GCG program Find is used in this phase. A specified pattern with many ambiguities is scanned for in each gene. The amount of matches varies in each gene. Once found, the patterns are categorized into four groups according to their putative Sp 1 binding affinities: High, medium, low, and none. In Table 2 the Sp 1 consensus sequence, labeled as the "perfect" pattern, is given at the top. For each position, the found pattern differs from the "perfect" pattern, a penalty is given. For example, suppose one of the found patterns is: GGGGCGTGGG. The idea is to see how much this pattern differs from the consensus sequence (or "perfect" pattern). Clearly, the symbols in positions 7 and 10 are different. The symbol in position 7 is a T; therefore, a penalty of -1 is given. The symbol in position 10 is a F; therefore, a penalty of -2 is given. Then, since penalties are cumulative, the total penalty is found to be -3. The penalty of -3 corresponds to a Medium affinity on the chart. Similarly, many patterns are evaluated.

**Table 2: Table of Sp 1 Consensus Binding Sites**

| Consensus (or "perfect" pattern) | | | | | | | | | | Single substitution | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| G | G | G | G | C | G | G | G | G | C | Penalty | Affinity |
| A | A | | | | | T | A | A | T | -1 | |
| T | T | | | | | | T | T | | -1 | High |
| C | | | | | | | | | A | -2 | |
| | | | | | | | | | G | -2 | High |
| | | | | A | | | | | | -3 | Medium |
| | | | | T | | | | | | -5 | Low |
| | | C | C | | | C | C | C | | -7 | None |
| | | | | | | | | | | <-7 | None |

## 5 Conclusions

The interesting output from the Composition and Sp 1 phase are directed to a database management system and to a statistics data analysis program. GENEX is being run on hundreds of Genbank entries and is continuously updated as more inconsistencies are uncovered in the entries. Furthermore, we have run GENEX on many randomized DNA. This was done to determine the occurrence of similar composition biases in the random sequences. Our objective for locating Sp 1 sites is to assess their statistical correlation with promoter regions in DNA entries.

## References

Devereux, J., et al. A comprehensive set of sequence analysis programs for the VAX. Nucleic Acids Research. 12:387-395 (1984).

Evans, T., et al. A promoter of the rat insulin-like growth factor II gene consists of minimal control elements. J. Molecular Biology. 199:61-81 (1988).

Giarrantano, J.C. and G. Riley. Expert Systems: Principles and Programming, 1989.

Mitchell, P.J. and R. Tigan. Transcriptional regulation in mammalian cells by sequence-specific DNA binding proteins. Science 245:371-378 (1989).

Wells, R.D., et. al. The Chemistry and Biology of unusual DNA structures adopted by oligopurine-oligopyrimidine sequences. FASEB J. 2:2939-2949 (1988).

# Knowledge Assisted Diagnosis Of Mood Disorders Using DSM-III

Robert H. Fritz
University of Houston Clear Lake
Link Flight Simulation

**Abstract.**

As part of an Expert Systems class at the University of Houston Clear Lake, a system has been developed using CLIPS to allow a clinical psychologist or psychiatrist to diagnose mood disturbances by providing answers to questions corresponding to branches of a DSM-III criteria tree. Experienced clinicians may assert indications of the client's behavior in order to circumvent multiple levels of the tree, thus speeding diagnosis. An explanation facility was developed for validation of the diagnosis. It also allows for "what if" scenarios by allowing the clinician to move backwards from the diagnosis to any decision branch and alter the answer previously provided. The system was implemented with a limited vocabulary of symptoms associated primarily with depressive disorders. However, the design supports the addition of vocabulary modules and knowledge bases for other types of disorders. The system currently has applicability in an instructional setting. With the addition of a more complete vocabulary, it could have applicability in a clinical setting. The overall design will support any application where determinations are made via a decision tree.

## Introduction.

This paper represents the culmination of a semester's work in expert systems at the University of Houston-Clear Lake. This paper discusses the author's final project for this course, Moody, a system written in CLIPS to allow a psychologist or an associated technician to perform a rudimentary diagnosis of mood disorders in adults. The first part of this paper uses a comparison of the psychological and medical diagnostic processes to illuminate the niche for a system like Moody. Next, there is a discussion of the attempts of various researchers to computerize the diagnostic process. Finally, Moody will be discussed in terms of the approach to its implementation, its output, and the resulting conclusions.

## Problem Background.

The psychological diagnostic process is less similar to the processes used by the other medical professions than simple observation might reveal. The first phase of the diagnosis of a psychological disorder is the patient interview. The interview is used by the clinician to ascertain not only the client's chief complaint, but also to determine the client's *real* identity: the household in which they were nurtured (or not nurtured); the types of people they call friends; the type of job they hold; traumatic experiences past or present. In short, the clinician needs to understand the external factors influencing the client's perspective of the world. In contrast, since most medical problems treated by the general practitioner are internal disorders manifesting themselves in external symptoms (e.g. pain), the interviewing process centers around any significant past medical history relevant to the current problem. The general practitioner is concerned more with the physiological nature of the patient. Keeping in mind the hypochondriacal illnesses treated by the general practitioner, and the forms of depression recurring in families for genetic reasons, the primary difference between the two professions is the outcome of the interviewing process. The general practitioner's interviewing process is better suited to diagnosing illnesses due to natural causes while the interviewing process of the psychologist is better suited to diagnosing illnesses due to "nurtural" causes (i.e. nature versus nurture).

Following the interview, both professions progress to the assessment phase which once again shows their dissimilarities. The purpose of the assessment phase is to measure quantitatively the effects of the symptoms in order to both refine the diagnosis and affect the treatment. The general practitioner has a large number of techniques with which to probe (both literally and figuratively) the human body. Accordingly, there is a vast difference between the information gleaned from an EEG and a biopsy. In general, less intrusive testing requires greater judgement to interpret the results. However, the use of intrusive procedures allows the re-evaluation of the judgmental data, allowing the subjective procedures to be less than rigorous. The psychological assessment process is based primarily on non-physically intrusive testing. In order to ensure different clinicians reach the same conclusions, very rigorous protocols for making assessments were developed. Herein lies the problem: rigorous procedures tend to be tedious. However, tedious problems are generally adaptable for computer use.

Among the psychologist's arsenal of assessment tools are the 16 Personality Factor Questionnaire, the Minnesota Multiphasic Personality Inventory (MMPI), and the Diagnostic and Statistical Manual of Mental Disorders, 3rd Edition (DSM-III). The DSM-III is a particularly rigorous solution based on a decision tree structure (see Figure 1). The clinician must answer a detailed question at every branch of the tree. Diagnoses are made at the tree's leaves. However, this process represents a tedious exercise to the experienced clinician. The purpose of Moody was to adapt the (1) DSM-III decision tree for use with a system that would allow the expert to provide knowledge about the client in a more free format. The system would associate this information with the appropriate branches of the DSM-III decision tree, alleviating the need to ask the questions at particular levels. This method of using knowledge to assist the protocol would lead to a shortened time to reach a diagnosis thereby eliminating some of the tedium.

Before Moody was implemented, research was done to ascertain if and how any other similar systems had been implemented. The next section of this paper highlights this research.

## Differential Diagnosis of Mood Disturbances

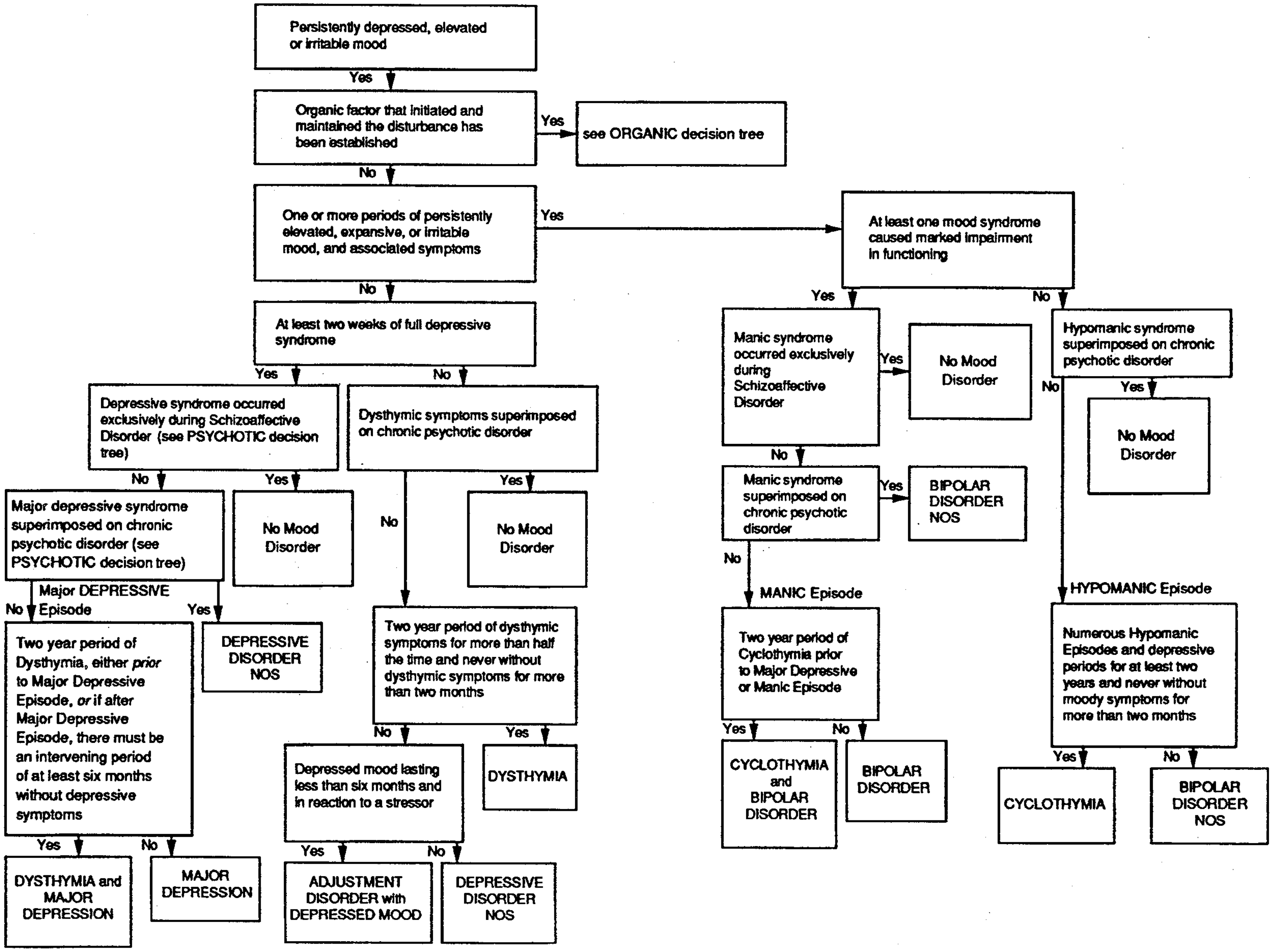

Persistently depressed, elevated or irritable mood
Yes
Organic factor that initiated and maintained the disturbance has been established
Yes
see ORGANIC decision tree
No
One or more periods of persistently elevated, expansive, or irritable mood, and associated symptoms
Yes
At least one mood syndrome caused marked impairment in functioning
No
At least two weeks of full depressive syndrome
Yes
No
Depressive syndrome occurred exclusively during Schizoaffective Disorder (see PSYCHOTIC decision tree)
Dysthymic symptoms superimposed on chronic psychotic disorder
No
Yes
Major depressive syndrome superimposed on chronic psychotic disorder (see PSYCHOTIC decision tree)
No Mood Disorder
No
Yes
No Mood Disorder
Major DEPRESSIVE Episode
No
Yes
Two year period of Dysthymia, either prior to Major Depressive Episode, or if after Major Depressive Episode, there must be an intervening period of at least six months without depressive symptoms
DEPRESSIVE DISORDER NOS
Two year period of dysthymic symptoms for more than half the time and never without dysthymic symptoms for more than two months
Yes
No
DYSTHYMIA and MAJOR DEPRESSION
MAJOR DEPRESSION
No
Yes
Depressed mood lasting less than six months and in reaction to a stressor
DYSTHYMIA
Yes
No
ADJUSTMENT DISORDER with DEPRESSED MOOD
DEPRESSIVE DISORDER NOS
Yes
No
Manic syndrome occurred exclusively during Schizoaffective Disorder
Yes
No Mood Disorder
No
Manic syndrome superimposed on chronic psychotic disorder
Yes
BIPOLAR DISORDER NOS
No
MANIC Episode
Two year period of Cyclothymia prior to Major Depressive or Manic Episode
Yes
No
CYCLOTHYMIA and BIPOLAR DISORDER
BIPOLAR DISORDER
Hypomanic syndrome superimposed on chronic psychotic disorder
No
Yes
No Mood Disorder
HYPOMANIC Episode
Numerous Hypomanic Episodes and depressive periods for at least two years and never without moody symptoms for more than two months
Yes
No
CYCLOTHYMIA
BIPOLAR DISORDER NOS

**Figure 1**

**Literature Review.**

In (2), Murphy and Pardeck discuss the various applications of computer technology to psychology. They concentrate on the area of the computer as a therapeutic tool. They warn that the computer as a provider of clinical services tends to dehumanize the client.

The first portion of the article extols the virtues of the computer as a mechanism for standardization. Using standard processes for modeling the data allows the data to be treated as an "inert object." The standardization thereby controls uncertainty by eliminating human bias from judgments. The authors site studies indicating computer based standardized inventory testing is cost-effective. In addition, the formalization of the process of collecting clients' histories has enabled the clinician to gather more comprehensive information.

The area of debate within clinical assessment is that although clinicians tend to be influenced by their biases and expectations, and computer-based evaluations solve this problem, computers are seen to give the client impersonal treatment. However, the authors site a study showing not only did 88% of the subjects claim the automated method was no more complex than one with a clinician, but also found it easier to confide sensitive problems to the machine. It appears the clients innately understand their clinician, as a human-being, will tend to judge behavior.

Interestingly, the areas of computer-assisted therapy and computerized therapy are marked by vastly different approaches. Computer assisted therapy has been practiced using role playing games such as Dungeons and Dragons as a "simulation" of life. As the client plays the game, the therapist acts as a mediator in extrapolating the client's responses to the game to real-life situations. In contrast, computerized therapy has used programs such as Weizenbaum's ELIZA with some "satisfying" results.

The authors' main criticism concerning the use of computers is actually a by-product of its chief advantage: standardization. The authors contend the encoding of information for a computer results in an overall loss of context for the information. Specifically, ***affect***, the way in which something is spoken, is lost. This loss can make it difficult to decipher the *meaning* of client's behavior and thereby make the client more difficult to diagnose and treat. However, the problem is not unusual. Any time information is categorized, there is always the dilemma of what to do with information that does not quite fit. There was an attempt to alleviate part of this problem within Moody.

In concluding, the authors indicate the need to understand the ramifications of a computer's use in the clinical setting. They reinforce the idea of the computer as a device not just supplementing, but mediating treatment. The abstracting qualities of the machine tend to divorce behavior from its meaning. In addition, the computer is unable to assume the role of the client to gain this understanding. Finally, the point is well taken that a system so general as to be appropriate for everyone, actually helps no one.

Christian Heath's (3) approach was interesting in that while most articles discussing computer aided diagnosis focus on the software's ease of use and accuracy, this article focuses on the diagnostic system's effect on the discourse between the doctor and the patient in a clinical environment. Researchers videotaped doctors utilizing a system to diagnose professional actors masquerading as patients. The idea was to observe any vocal and nonvocal behaviors induced by the machine.

The researchers' observations support the intuitively obvious occurrences of every day conversation. For instance, eye-contact is used in every day living to infer the focus of attention (e.g. parent speaking to child: "Look at **me** when I talk to you!!") The researchers found a patient would wait for a doctor to look up from the machine before answering a question. If the doctor looked back down at the machine while listening to the patient, the patient's speaking pattern changed to include devices for attracting the doctor's attention. These perturbations lowered the amount of information being provided to the doctor. In addition, it was shown the doctor missed information while looking at the machine. Taped instances showed a doctor looking up from the machine to ask a question to which the patient had just provided an answer. The author's viewpoint is amusing: "... using the computer whilst [sic] the patient is speaking is quite simply an uneconomic use of the doctor's time." (2, p. 336) What about the patient's time? The author sites a study indicating this problem is not just indicative of a computer. If the doctor writes while talking to the patient, similar phenomena are observed.

The last portion of the article dealt with the problem of noise during the session. There was no mention of what type of computer they were using, but it apparently produced a noise at regular intervals. It is no surprise the noise initially distracted both the doctor and the patient and made discourse difficult. However, it appears from the tapes both doctor and patient determined the rhythm of the noise and coordinated their conversation around it. Once again, the computer mediated treatment.

In the context of Moody, it is clear that if the machine is in the clinical setting with the patient, its operation should be as unobtrusive as possible to allow maximum communication between the clinician and the client. Any "noises" should be eliminated.

Harold Erdman (4) discusses how the deployment of an expert system designed for non-experts effected the quality of treatment. The study involves an expert system created by Dr. Erdman for the treatment of depression, the most common psychiatric problem seen by primary care physicians. The author sites studies indicating " ... primary care physicians' diagnosis and treatment of psychiatric problems leaves a great deal to be desired." (1, p. S138)

Dr. Erdman emphasizes the main benefit of his system is its ability to explain the reasoning behind a decision. He sites studies indicating explanations can increase the willingness to accept the computer's advice. Dr. Erdman places great value on a system's ability to occasionally change the mind and behavior of the clinician. And he goes on to state "If computer and human are to make better decisions together than either can alone, the clinician must sometimes follow the computer's advice when the computer disagrees with the clinician and the clinician must sometimes reject the computer's advice." (1, p. S139)

There were two decision models and three modes of evaluation of Dr. Erdman's system. The first analysis examined the rate of agreement between the physician and the system. The second analysis attempted to determine whether physician judgments improved as a result of using the program. The final analysis examined the physicians' positive attitudes toward the system both with and without the explanation facility. The two groups exposed to the system were physicians and psychiatrists. Both groups were shown to agree with the system to a statistically significant extent. As would be expected, physicians' attitudes towards the system were positive when explanations were provided. Psychiatrists' attitudes were unmoved by explanations (although they sometimes changed their minds to agree with the recommendation).

As part of the final analysis, Dr. Erdman compared the effectiveness of his two decision models, a decision analytic model and a protocol. Although neither model performed significantly better than the other, Dr. Erdman expressed a preference for the decision analytic model since its modularity made it easier to develop and modify.

Other interesting results were cited. Dr. Erdman indicates although the computer model performed at the level of the average psychiatrist, psychiatrists' performance improved, supporting the two-heads-are-better-than-one theory. In terms of explanations, Dr. Erdman indicates that physicians, less expert than the system, may have been "inappropriately influenced" by the system's explanations, even when the designated treatment was incorrect. In addition, in cases where the system disagreed with the expert and a weak explanation was provided, the expert's opinion of the system declined.

Dr. Erdman concludes by indicating even an imperfect aid may improve patient care. Explanations may also improve the system's usefulness, but his study was not able to statistically measure the extent to which explanations altered the physician's judgment. This ability would allow explanation content and therefore its effects to be controlled.

## Overview of Moody.

The premise for creating Moody is that although the DSM-III is a very rigorous and accurate clinical assessment protocol, it is tedious because of its decision tree nature. The DSM-III forces the clinician to organize thoughts and materials into a template suiting itself, not the clinician.

Moody uses two knowledge bases. The first knowledge base represents the DSM-III decision tree for performing differential diagnosis of mood disorders. Since the DSM-III allows for both the assignment of a category and a numerical severity indicator to a diagnosis, this system may be said to contain "shallow knowledge," since severity is not processed at this time. The second knowledge base represents a natural language interface capable of processing terminology used by the clinician during assessment. Since Moody was implemented primarily as a proof of concept and used only one expert as a reference, the language base represents only a shallow subset of the technical terminology available.

Without the language base, Moody operates by leading the clinician query by query down the decision tree to a diagnosis. Once a diagnosis has been reached, the explanation facility allows the clinician to backup and re-examine any node in the tree, altering the response to any query and thereby the diagnosis rendered by the system. By using the language base, at any point during the query process the clinician can specify client information using the appropriate jargon. If criteria-satisfying information is provided, a corresponding number of query nodes will be circumvented. Shortening the decision path will shorten the time to diagnosis and thereby alleviate some of the tedium.

## Detailed Description of Moody.

Moody was implemented in three phases. The first phase consisted of encoding the DSM-III decision tree from Figure 1, each node representing a single CLIPS fact. Using a strict computer science definition, the Moody decision tree is not a *tree* at all, but a *lattice*, since some terminal (leaf) nodes can be reached through several paths. This structure results in a node with multiple parents. In order to support an explanation facility, the lattice was turned into a tree by replicating the leaf nodes and giving them a unique identity. The adapted decision tree and the associated rule numbers are shown in Figure 2. The hatched areas identify the replicated nodes with their unique rule numbers.

For CLIPS representation purposes, nodes were divided into two categories: decision (non-leaf) nodes and diagnosis (leaf) nodes. The template for a decision fact is:

```
query <rule_f??>  <yes-rule> <no-rule> decision <question-1>
                                              ...<question-N>
```

where <rule_f??> is the current rule number (from Figure 2) and <yes-rule> and <no-rule> represent the rule numbers of the query's "yes" and "no" responses. Data item <question-X> represents the ASCII query used to prompt the clinician. For example, the following is the actual CLIPS representation for the decision tree root:

```
(query rule_f2           rule_f2a rule_f3 decision
     "Was it an organic factor that initiated"
     "and maintained the disturbance?"

)
```

When presented with this query, if the clinician answers "Yes," the next node to be displayed will be rule_f2a. A "No" response causes node rule_f3 to be displayed.

## Differential Diagnosis of Mood Disturbances

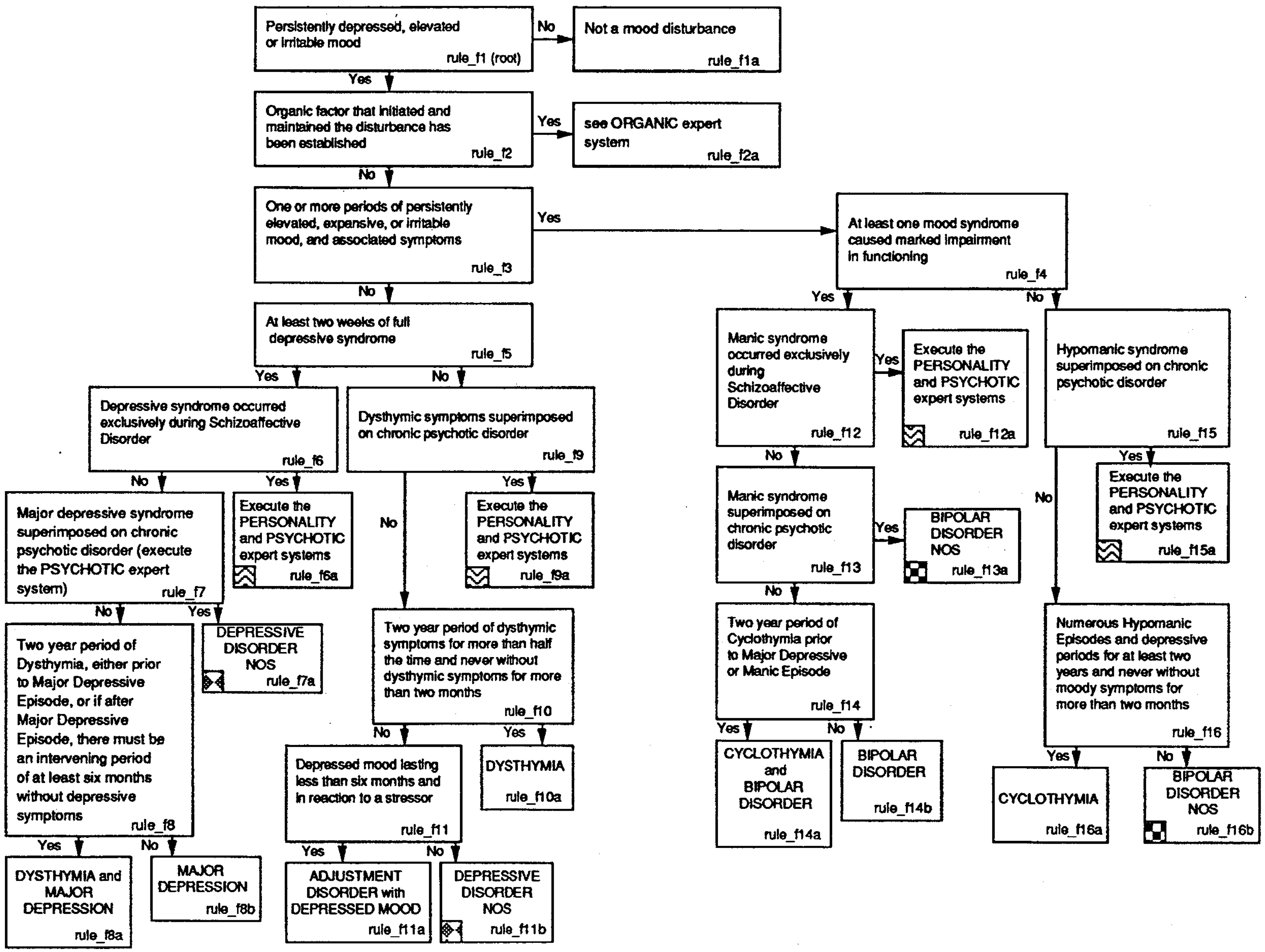

Persistently depressed, elevated or irritable mood rule_f1 (root)
No
Not a mood disturbance rule_f1a
Yes
Organic factor that initiated and maintained the disturbance has been established rule_f2
Yes
see ORGANIC expert system rule_f2a
No
One or more periods of persistently elevated, expansive, or irritable mood, and associated symptoms rule_f3
Yes
At least one mood syndrome caused marked impairment in functioning rule_f4
No
At least two weeks of full depressive syndrome rule_f5
Yes
Depressive syndrome occurred exclusively during Schizoaffective Disorder rule_f6
No
Dysthymic symptoms superimposed on chronic psychotic disorder rule_f9
No
Major depressive syndrome superimposed on chronic psychotic disorder (execute the PSYCHOTIC expert system) rule_f7
Yes
Execute the PERSONALITY and PSYCHOTIC expert systems rule_f6a
Yes
Execute the PERSONALITY and PSYCHOTIC expert systems rule_f9a
No
Two year period of Dysthymia, either prior to Major Depressive Episode, or if after Major Depressive Episode, there must be an intervening period of at least six months without depressive symptoms rule_f8
Yes
DEPRESSIVE DISORDER NOS rule_f7a
Two year period of dysthymic symptoms for more than half the time and never without dysthymic symptoms for more than two months rule_f10
Yes
DYSTHYMIA and MAJOR DEPRESSION rule_f8a
No
MAJOR DEPRESSION rule_f8b
No
Depressed mood lasting less than six months and in reaction to a stressor rule_f11
Yes
DYSTHYMIA rule_f10a
Yes
ADJUSTMENT DISORDER with DEPRESSED MOOD rule_f11a
No
DEPRESSIVE DISORDER NOS rule_f11b
Yes
Manic syndrome occurred exclusively during Schizoaffective Disorder rule_f12
Yes
Execute the PERSONALITY and PSYCHOTIC expert systems rule_f12a
No
Hypomanic syndrome superimposed on chronic psychotic disorder rule_f15
No
Manic syndrome superimposed on chronic psychotic disorder rule_f13
Yes
BIPOLAR DISORDER NOS rule_f13a
Yes
Execute the PERSONALITY and PSYCHOTIC expert systems rule_f15a
No
No
Two year period of Cyclothymia prior to Major Depressive or Manic Episode rule_f14
Numerous Hypomanic Episodes and depressive periods for at least two years and never without moody symptoms for more than two months rule_f16
Yes
CYCLOTHYMIA and BIPOLAR DISORDER rule_f14a
No
BIPOLAR DISORDER rule_f14b
Yes
CYCLOTHYMIA rule_f16a
No
BIPOLAR DISORDER NOS rule_f16b

Figure 2

In a similar vein, the template for a diagnosis fact is:

```
query <rule_f??> diagnosis <diagnosis-1>....<diagnosis-N>
```

where <rule_f??> is once again the current rule number and <diagnosis-X> is the text for the diagnosis. For example, the following is the terminal node for rule_f2a's "yes" response:

```
(query rule_f2a         diagnosis
     "Please execute the ORGANIC expert system."
)
```

Following the display of the above message, Moody will display "Diagnosis Complete" and will then be ready to enter explanation mode.

The explanation facility was implemented as the second phase of the project. Following the activation of a diagnosis node, Moody enters an explanation phase controlled by backward-chaining algorithm. Taking advantage of the tree structure in that every node has only one parent (even if the leaf itself is not unique), the algorithm locates the node with the "yes" or "no" rule number matching the current node's rule number. When the explanation facility is entered after reaching node rule_f2a (see above), the response is:

```
Because you answered or implied 'Yes' to:
Was it an organic factor that initiated
and maintained the disturbance?

Do you want to continue? (y, n, or a)
```

At this point, if the clinician responds "y", Moody will display the parent of this node, the tree's root. If the response is "n", backward-chaining will cease in order to allow the clinician to reset the system for another diagnosis. Finally, if the clinician responds "a" (symbolizing "alter"), the system resumes diagnosis mode and reissues the current query for a new (presumably different) response. This facility allows a clinician to test multiple scenarios for a single patient without having to restart the system.

As already mentioned, every node has only one parent. However, note the word "implied" in the above example. The explanation facility is limited in that it has no visibility of the triggering of a decision node as the result of cumulative symptoms specified by the clinician via the natural language interface.

The natural language interface was implemented as the third phase of the project. The intent was to allow the clinician to provide as much client information as possible up-front. The inference engine would be used to match the specified information to an encoded form of the query at a given node. If the criteria spelled out by the query were met, then the appropriate answer to the query could be implied.

Language information was divided into two rule bases. The first rule base implemented a symbolic link between a category of illness and a rule in the decision tree. The divisions were determined by examining the decision tree and answering the question "at this point in the tree, the client's symptoms are leading to what category of diagnosis?" The intent was twofold: first, to allow enough certainty in the specification of symptoms to enable top levels of the tree, the general information, to be bypassed; second: not to force the clinician to type in the diagnosis through sheer detail. Referring to Figure 2, if a clinician specifies enough information to categorize a client as suffering from major depression, querying can begin at rule_f7. If the client is manic, querying can begin at rule_f12. The template for these facts is:

```
                    Category <category> Rule <rule>

     (e.g.)         Category MAJOR-DEPRESSIVE Rule rule_f7
                    Category MANIC Rule rule_f12
```

The second rule base established a symbolic link between a symptom and a category of illness. Whereas the relationship between categories and rules is one-to-one, the relationship between symptoms and categories is many-to-one i.e. enough symptoms will eventually determine the category of illness. These linkages are expressed using following template:

```
                    Symptom <symptom> Category <category>

     (e.g.)         Symptom "IRRITABLE" Category MANIC
                    Symptom "GRANDIOSITY" Category MANIC
                    Symptom "RAPID SPEECH" Category MANIC
```

Once the language rule bases were established, a mechanism was created to allow the clinician to indicate the symptoms being exhibited by the client. This information is expressed in the form:

```
                    client <symptom> <instances> <duration>
```

where

```
                    client "IRRITABLE" 2 14
```

would be interpreted as meaning the client has had two instances of exhibiting irritable symptoms lasting fourteen days. The <instances> data item is used within the DSM-III criteria to achieve a more detailed diagnosis. Moody does not utilize this information and therefore produces a more rudimentary diagnosis.

In order to support language processing at any point during the diagnostic process, the query algorithm was modified to allow the clinician to not only specify "y" and "n" as answers to questions, but also to suspend the diagnostic process by specifying "s." During this suspended mode, the clinician was free to assert as many client facts as desired. Only upon entering the command to resume execution would Moody act upon the data.

The chosen method of superimposing the language data upon DSM-III criteria tree raised two technical issues worth noting. First, since the CLIPS inference engine does not guarantee the order in which rules fire, it was possible to fire a query rule before a language rule. For this reason, language rules were given a higher salience than query rules. Secondly, even the rudimentary diagnoses made by Moody sometimes required eight left-hand-side pattern matches. Since the number of partial matches was potentially high enough to exhaust the memory on a 640K personal computer, the patterns of highest priority and least likelihood were listed first in order to short-circuit through failed pattern evaluations.

**Moody Results.**

Numerous test runs of Moody were performed, a sample of which is included in Appendix A. In all cases, the diagnoses conformed to DSM-III criteria and agreed with the expert, limited by the depth of knowledge utilized by the system.

Following the testing period, the assistance of several psychologists was sought in order to evaluate the system in a clinical setting. Although all were interested and enthusiastic at the prospect of a finished product, none could afford the time to evaluate it in a clinical trial.

## Conclusions.

Moody satisfied the premise that shortening the decision path would shorten the time to diagnosis. However, it became apparent during the testing process that Moody substituted a different kind of tedium: typing. By using the language interface to eliminate queries, the burden of communication was placed on the clinician whose only mechanism for input was to assert the "client" facts. The primary disadvantage of this mechanism was the lack of a cue to indicate what symptoms were categorized by the system. Moody's vocabulary is limited. Entering symptoms not in the rule base would result in no transfer of control and no feedback to the clinician. In addition, in some cases entering the symptoms took longer than answering the questions. It was for this reason rules were created to assert the "client" facts during the testing phase.

By substituting a menu and mouse interface for the typing interface, three problems would be alleviated. First, the menus would eliminate the need to type in the symptoms while simultaneously providing a list of Moody's vocabulary. For a complete implementation, the vocabulary should be large enough to avoid the problem of forcing a client to be placed in a category by compromise, one of the concerns of Murphy and Pardeck (2). The menu interface combined with a phase-assertion technique would also eliminate the need to use salience to control the inference engine. Finally, the menu interface would act as a shell, obviating the need to leave the clinician at the CLIPS prompt during modes when activity was suspended.

Diagnoses using Moody agreed with the expert, but only to the limit of the shallow knowledge utilized by the system. For this reason, Moody is more suited as a teaching tool for the less experienced clinician. Even though the language interface is cumbersome, the DSM-III decision tree along with the explanation facility could be used to test a junior clinician's particular hypotheses. Finally, the expert expressed the opinion "if it's not easy to use, no one will use it."

In addition to improving the user interface to make the system easier to use, there are two other areas where refinement is needed. Currently, Moody does not prioritize diagnoses. There is the potential for a situation where multiple diagnoses could be activated. It would seem prudent to categorize the client in the more serious category. Coupled with prioritization, the explanation facility should be expanded to encompass "client" facts. The explanation facility should indicate something similar to "Query at rule_??? satisfied by symptoms: symptom1, symptom2, ... , symptomN."

Finally, it should be noted the mechanism of representing the decision tree is generic. There is no reason why a different knowledge base could not be substituted for the DSM-III criteria. The method of circumventing nodes on the tree is portable, but the language criteria is not.

## References

(1) American Psychiatric Association (1987) Diagnostic and Statistical Manual of Mental Disorders, Third Edition, Revised.

(2) Murphy, John W., & Pardeck, John T. (1986) "Computerized Clinical Practice: Promises and Shortcomings." Psychological Reports, Vol. 59, Pp. 1099-1113.

(3) Heath, Christian (1983) "Research note: Computer-aided diagnosis in the consultation." Sociology of Health and Illness, 5-3, Pp. 333-345.

(4) Erdman, Harold P. (1987) "A Computer Consultation Program for Primary Care Physicians." Medical Care, 25-12, Pp. S138-S147.

(5) Giarratano, J., & Riley (1989) Expert Systems Principles and Programming.

**Appendix A -- Execution Listing of Moody**

The annotations for this listing are in Appendix B.

```
001  CLIPS> (load "moody.clp")
002  **************
003  CLIPS> (reset)
004  CLIPS> (run)
005  Loading DSM-III Data ...
006  Loading Language Rules ...
007  **********************
008  Loading Language Data ...
009
010  root:
011  Can the client be described as persistently
012  depressed, elevated, expansive, or of
013  irritable mood? (y, n, or s) y
014
015  rule_f2:
016  Was it an organic factor that initiated
017  and maintained the disturbance? (y, n, or s) n
018
019  rule_f3:
020  Has the client had one or more periods of
021  persistently elevated, expansive, or irritable
022  mood, and associated symptoms? (y, n, or s) y
023
024  rule_f4:
025  Has at least one mood syndrome caused
026  marked impairment in functioning? (y, n, or s) y
027
028  rule_f12:
029  Did the manic syndrome occur exclusively
030  during Schizoaffective Disorder? (y, n, or s) n
031
032  rule_f13:
033  Is manic syndrome superimposed on chronic
034  psychotic disorder? (y, n, or s) y
035
036  rule_f13a:
037  BIPOLAR DISORDER NOS.
038
039  Diagnosis Complete.  Type '(run)' for an explanation.
040  29 rules fired
041  CLIPS> (run)
042
043  Because you answered or implied 'Yes' to:
044  Is manic syndrome superimposed on chronic
045  psychotic disorder?
046
047  Do you want to continue? (y, n, or a) a
```

```

rule_f13:
Is manic syndrome superimposed on chronic
psychotic disorder? (y, n, or s) n

rule_f14:
Has there been a two year period of
Cyclothymia prior to Major Depressive or
Manic Episode? (y, n, or s) y

rule_f14a:
CYCLOTHYMIA and BIPOLAR DISORDER.

Diagnosis Complete.  Type '(run)' for an explanation.
16 rules fired
CLIPS> (run)

Because you answered or implied 'Yes' to:
Has there been a two year period of
Cyclothymia prior to Major Depressive or
Manic Episode?

Do you want to continue? (y, n, or a) y

Because you answered or implied 'No' to:
Is manic syndrome superimposed on chronic
psychotic disorder?

Do you want to continue? (y, n, or a) y

Because you answered or implied 'No' to:
Did the manic syndrome occur exclusively
during Schizoaffective Disorder?

Do you want to continue? (y, n, or a) y

Because you answered or implied 'Yes' to:
Has at least one mood syndrome caused
marked impairment in functioning?

Do you want to continue? (y, n, or a) y

Because you answered or implied 'Yes' to:
Has the client had one or more periods of
persistently elevated, expansive, or irritable
mood, and associated symptoms?

Do you want to continue? (y, n, or a) y

Because you answered or implied 'No' to:
Was it an organic factor that initiated
and maintained the disturbance?

Do you want to continue? (y, n, or a) y
```

```

Because you answered or implied 'Yes' to:
Can the client be described as persistently
depressed, elevated, expansive, or of
irritable mood?

Explanation Complete.
31 rules fired
CLIPS> (reset)
CLIPS> (run)
Loading DSM-III Data ...
Loading Language Rules ...
**********************
Loading Language Data ...

root:
Can the client be described as persistently
depressed, elevated, expansive, or of
irritable mood? (y, n, or s) s

Diagnosis Suspended.  Type '(run)' to continue.
7 rules fired
CLIPS> (watch facts)
CLIPS> (assert (Manic))
==> f-98    (Manic)
CLIPS> (run 1)
==> f-99    (client "IRRITABLE" 1 14)
==> f-100   (client "GRANDIOSITY" 1 14)
==> f-101   (client "RAPID SPEECH" 1 14)
<== f-98    (Manic)
rule firing limit reached
1 rules fired
CLIPS> (unwatch facts)
CLIPS> (run)
Moving from rule root to rule rule_f12 for MANIC reasons.

rule_f12:
Did the manic syndrome occur exclusively
during Schizoaffective Disorder? (y, n, or s) n

rule_f13:
Is manic syndrome superimposed on chronic
psychotic disorder? (y, n, or s) n

rule_f14:
Has there been a two year period of
Cyclothymia prior to Major Depressive or
Manic Episode? (y, n, or s) y

rule_f14a:
CYCLOTHYMIA and BIPOLAR DISORDER.

Diagnosis Complete.  Type '(run)' for an explanation.
26 rules fired
```

```
CLIPS> (reset)
CLIPS> (run)
Loading DSM-III Data ...
Loading Language Rules ...
**********************
Loading Language Data ...

root:
Can the client be described as persistently
depressed, elevated, expansive, or of
irritable mood? (y, n, or s) s

Diagnosis Suspended.  Type '(run)' to continue.
7 rules fired
CLIPS> (watch facts)
CLIPS> (assert (Cyclothymic))
==> f-98    (Cyclothymic)
CLIPS> (run 1)
==> f-99    (client "MARKED IMPAIRMENT" 1 730)
==> f-100   (client "MOOD SWINGS" 1 730)
==> f-101   (client "IRRITABLE" 1 730)
==> f-102   (client "GRANDIOSITY" 1 730)
==> f-103   (client "RAPID SPEECH" 1 730)
<== f-98    (Cyclothymic)
rule firing limit reached
1 rules fired
CLIPS> (unwatch facts)
CLIPS> (run)
Moving from rule root to rule rule_f14 for CYCLOTHYMIC reasons.

rule_f14:
Has there been a two year period of
Cyclothymia prior to Major Depressive or
Manic Episode? (y, n, or s) y

rule_f14a:
CYCLOTHYMIA and BIPOLAR DISORDER.

Diagnosis Complete.  Type '(run)' for an explanation.
27 rules fired
CLIPS> (run)

Because you answered or implied 'Yes' to:
Has there been a two year period of
Cyclothymia prior to Major Depressive or
Manic Episode?

Do you want to continue? (y, n, or a) y

Because you answered or implied 'No' to:
Is manic syndrome superimposed on chronic
psychotic disorder?

```

```
Do you want to continue? (y, n, or a) y

Because you answered or implied 'No' to:
Did the manic syndrome occur exclusively
during Schizoaffective Disorder?

Do you want to continue? (y, n, or a) a

rule_f12:
Did the manic syndrome occur exclusively
during Schizoaffective Disorder? (y, n, or s) y

rule_f6a:
Please execute the PSYCHOTIC expert system.
In addition, Personality Disorders may also
be indicated.  Please execute the PERSONALITY
expert system.

Diagnosis Complete.  Type '(run)' for an explanation.
23 rules fired
CLIPS> (reset)
CLIPS> (run)
Loading DSM-III Data ...
Loading Language Rules ...
**********************
Loading Language Data ...

root:
Can the client be described as persistently
depressed, elevated, expansive, or of
irritable mood? (y, n, or s) s

Diagnosis Suspended.  Type '(run)' to continue.
7 rules fired
CLIPS> (watch facts)
CLIPS> (assert (Psychotic))
==> f-98    (Psychotic)
CLIPS> (run 1)
==> f-99    (client "SCHIZOPHRENIC" 1 14)
<== f-98    (Psychotic)
rule firing limit reached
1 rules fired
CLIPS> (unwatch facts)
CLIPS> (run)
Moving from rule root to rule rule_f9a for PSYCHOTIC
reasons.

rule_f9a:
Please execute the PSYCHOTIC expert system.
In addition, Personality Disorders may also
be indicated.  Please execute the PERSONALITY
expert system.

Diagnosis Complete.  Type '(run)' for an explanation.
```

```
8 rules fired
CLIPS> (reset)
CLIPS> (run)
Loading DSM-III Data ...
Loading Language Rules ...
**********************
Loading Language Data ...

root:
Can the client be described as persistently
depressed, elevated, expansive, or of
irritable mood? (y, n, or s) s

Diagnosis Suspended.  Type '(run)' to continue.
7 rules fired
CLIPS> (assert (Depressive))
CLIPS> (watch facts)
CLIPS> (run 1)
==> f-99    (client "DEPRESSED MOOD" 1 14)
==> f-100   (client "POOR CONCENTRATION" 1 14)
==> f-101   (client "INDECISIVE" 1 14)
==> f-102   (client "APATHETIC" 1 14)
==> f-103   (client "WEIGHT LOSS" 1 14)
<== f-98    (Depressive)
rule firing limit reached
1 rules fired
CLIPS> (unwatch facts)
CLIPS> (run)
Moving from rule root to rule rule_f7 for DEPRESSING reasons.

rule_f7:
Is major depressive syndrome superimposed
on chronic psychotic disorder? (y, n, or s) n

rule_f8:
Has the client had a two year period of Dysthymia,
either prior to a Major Depressive Episode, or if
after a Major Depressive Episode, was there an
intervening period of at least six months without
depressive symptoms? (y, n, or s) y

rule_f8a:
DYSTHYMIA and MAJOR DEPRESSION.

Diagnosis Complete.  Type '(run)' for an explanation.
30 rules fired
CLIPS> (reset)
CLIPS> (run)
Loading DSM-III Data ...
Loading Language Rules ...
**********************
Loading Language Data ...

```

```
root:
Can the client be described as persistently
depressed, elevated, expansive, or of
irritable mood? (y, n, or s) s

Diagnosis Suspended.  Type '(run)' to continue.
7 rules fired
CLIPS> (watch facts)
CLIPS> (assert (Dysthymic))
==> f-98    (Dysthymic)
CLIPS> (run 1)
==> f-99    (client "DEPRESSED MOOD" 1 730)
==> f-100   (client "SUICIDAL" 1 730)
==> f-101   (client "DECREASED APPETITE" 1 730)
<== f-98    (Dysthymic)
rule firing limit reached
1 rules fired
CLIPS> (unwatch facts)
CLIPS> (run)
Moving from rule root to rule rule_f8 for DYSTHYMIC reasons.

rule_f8:
Has the client had a two year period of Dysthymia,
either prior to a Major Depressive Episode, or if
after a Major Depressive Episode, was there an
intervening period of at least six months without
depressive symptoms? (y, n, or s) y

rule_f8a:
DYSTHYMIA and MAJOR DEPRESSION.

Diagnosis Complete.  Type '(run)' for an explanation.
20 rules fired
CLIPS> (run)

Because you answered or implied 'Yes' to:
Has the client had a two year period of Dysthymia,
either prior to a Major Depressive Episode, or if
after a Major Depressive Episode, was there an
intervening period of at least six months without
depressive symptoms?

Do you want to continue? (y, n, or a) a

rule_f8:
Has the client had a two year period of Dysthymia,
either prior to a Major Depressive Episode, or if
after a Major Depressive Episode, was there an
intervening period of at least six months without
depressive symptoms? (y, n, or s) n

rule_f8b:
MAJOR DEPRESSION.

```

```
Diagnosis Complete.  Type '(run)' for an explanation.
17 rules fired
CLIPS> (dribble-off)
```

## Appendix B -- Execution Listings (Appendix A) Annotations

Lines 1-62 of the listing show the normal loading and execution of Moody without utilizing the language interface. At line 37, a diagnosis of BIPOLAR DISORDER NOS (Not Otherwise Specified) is reached. At line 41, demonstration of the explanation facility begins. At line 47, the alter option of the explanation facility is demonstrated. Query rule_f13 is repeated, and a different answer (see line 34) is provided. At line 62, a new diagnosis is reached. Please note the diagnosis at line 59 is reached after seven queries (ignoring the backtracking).

Lines 63-109 demonstrate a complete explanation that chains back to the root node.

Lines 117-155 demonstrate Moody's language interface. Line 120 demonstrates the use of the "suspend" option. In order to facilitate testing, additional rules were created to trigger major events. Line 125 shows the assertion of the Manic fact. This fact serves to trigger a rule asserting the "watched" client facts. Line 136 shows the transfer of control from the root to rule_f12 because the asserted symptoms implied manic behavior. Once again, CYCLOTHYMIA and BIPOLAR DISORDER are indicated (line 152). However, the diagnosis is reached after only three queries.

Lines 163-194 demonstrate a complex inference in which a great deal of detail is specified. The same diagnosis as above is reached after only one query.

Lines 198-209 illustrate the explanation facility's lack of visibility into the language interface. As already mentioned, the explanation facility operates on the assumption the current node was reached via its parent.

Lines 236-262 demonstrate a case where the language interface causes a branch directly to a leaf node. As far as the DSM-III is concerned, mood disorders are rarely accompanied by psychotic symptoms.

Lines 270-370 exercise the remaining language-based diagnoses.

# A CLIPS Expert System for Clinical Flow Cytometry Data Analysis

[1]G.C. Salzman, Ph.D., [2]R.E. Duque, M.D., [3]R.C. Braylan, M.D., [4]C.C. Stewart, Ph.D.

[1]Life Sciences Division, Los Alamos National Laboratory, Los Alamos, NM 87545

[2]Norwood Clinic, 1528 N. 26th St., Birmingham, AL 35234

[3]Department of Pathology, University of Florida College of Medicine, Gainesville, FL 32610

[4]Roswell Park Cancer Center, Laboratory of Flow Cytometry, 666 Elm Street, Buffalo, NY 14263

## Abstract

An expert system is being developed using CLIPS to assist clinicians in the analysis of multivariate flow cytometry data from cancer patients. Cluster analysis is used to find subpopulations representing various cell types in multiple datasets each consisting of four to five measurements on each of 5000 cells. CLIPS facts are derived from results of the clustering. CLIPS rules are based on the expertise of Drs. Stewart, Duque, and Braylan. The rules incorporate certainty factors based on case histories.

## Introduction

Flow Cytometry [1-3] has become an accepted technique in the clinical laboratory for rapidly classifying cell types in blood, bone marrow, and some solid tumor samples. Cells are labeled with fluorescent cell-type-specific markers and then passed one-at-a-time through a focused laser beam. In commercial flow cytometers typically used in clinical laboratories three colors of fluorescence can be detected from a cell as well as light scattered in the forward direction and at right angles to the laser beam. Multiple, fluorescent-labeled monoclonal antibodies are used to tag the cells, which are then analyzed one at a time at rates of several thousand cells a second. Samples can be processed through the flow cytometer at rates of more than one a minute. Clinicians are being overwhelmed by the large amount of data that must be analyzed to provide the information needed to assist in disease diagnosis.

## Immunophenotyping

Immunophenotyping is the science of using antibodies to identify cells. The outer membrane of a cell contains many structurally specific molecules called surface antigens. These antigens have specific sites called epitopes to which the antibodies bind. Monoclonal antibodies are molecules derived from cells all having a common parent. These antibodies can be tagged with fluorescent dyes that are excited at the same 488 nm argon laser wavelength, but that emit their fluorescence at different wavelengths. Each fluorescence detector has a filter to accept the fluorescence from only one of the markers. Each cell can be tagged with from one to three different monoclonal antibodies. The tags bound to the cell surface enable identification of the cell type.

## Cluster Analysis

The flow cytometer [4] converts the four to five signals from each cell into digital values and stores them in a correlated fashion called list mode so that the relationships among the four to five variates for each cell are preserved. The data can be displayed as a group of from six to ten bivariate dot plots. K-means cluster analysis [5-6] is carried out on each dataset to find the means of each subpopulation of cells in the sample. Heuristics, developed to assign numerical thresholds to the words negative, dim, and bright regarding fluorescence values and to the words low, medium, and high for scattered light values, are used to translate the mean values of the clusters for each variate into symbolic facts for use by the CLIPS rules. The rules are a direct translation of the reasoning process used by the clinical laboratory personnel in processing the data by hand. The rule syntax is clear enough that the rules can be examined by these people and changed to reflect new knowledge.

## Bivariate Plots

Figure 1 shows the six bivariate dot plots from four variate flow cytometry measurements on the blood of a patient with acute leukemia. The 1.2 label in the lower left hand bivariate plot indicates that the x-axis is variate 1, which is FSC, the intensity of forward scattered light and that the y-axis is variate 2, which is SSC, the intensity of light scattered by a cell at $90^o$ to the laser beam axis. Variate 3 is FL1, the emission from a fluorescent-labeled monoclonal antibody. Variate 4 is FL2, which is propidium iodide (PI) fluorescence. PI stains the DNA of dead cells, which can then be excluded from further analysis. The bivariate plots are arranged so that variate 1 is the x-axis for column 1, variate 2 is the x-axis for column 2, and variate 3 is the x-axis for column 3. The cluster analysis program has labeled the data for each cell with a letter corresponding to its cluster association. The clustering algorithm has been instructed to find three clusters in these data. Each ellipse is centered on a cluster and is two standard deviations wide in the x-direction and two standard deviations wide in the y-direction. The important bivariate plot is the one in the upper right hand corner in which x is 3 and y is 4. Clusters B and C represent cells that have taken up PI and so are dead. Cluster A would be called "negative" for variates 3 (monoclonal antibody) and 4 (PI). The axes for the fluorescence measurements (FL1, FL2 and FL3) span four decades on a logarithmic scale. Figure 2 shows the bivariate plots for the same patient using a different monoclonal antibody for variate 3 (FL1). Here cluster A in bivariate 3.4 is "dim" for variate 3 and "negative" for variate 4. Figure 3 shows the bivariate plots for the same patient for another monoclonal antibody for variate 3. Cluster A in bivariate 3.4 (upper right hand corner) is "bright" for variate 3 and "negative" for variate 4.

## Results and Discussion

After the cluster analysis has been run on the samples for a patient, a C function translates the means and standard deviations into CLIPS facts, which can then be pattern matched against the conditions in the rules. The first prototype [7] used a rigid decision tree on five variate data in which multiple monoclonal antibodies were used to label the cells. Only nine of eleven acute leukemia cases were correctly assigned. Misclassifications occurred because of the boundaries between "negative" and "dim" and "dim" and "bright" were fixed. Variability in staining intensity and laser power were sufficient to move the means of clusters so that the incorrect choice was sometimes made at a decision node.

A second prototype is now being developed that incorporates uncertainty through the use of certainty factors and measures of belief [8-10]. Facts and rules are selected for evaluation in the order that gives the greatest improvement in certainty for one of the possible outcomes. This approach has been successfully used recently in the diagnosis of colonic lesions [11]. The certainty factors are being assigned initially from *a priori* probabilities calculated from a database containing a large number of case histories. The results for this second prototype will be reported at the meeting.

## References

1. Howard M. Shapiro. Practical Flow Cytometry, 2nd Ed., Alan Liss, NY, 1988.
2. M.A. Van Dilla, P.N. Dean, O.D. Laerum, M.R. Melamed (eds). Flow Cytometry: Instrumentation and Data Analysis. Academic Press, NY, 1985.
3. Andrew Yen. Flow Cytometry: Advanced Research and Clinical Applications. Vols I and II. CRC

Press, Boca Raton, FL, 1989.
4. FACSCAN, Becton-Dickinson, Inc., San Jose, CA.
5. H. Spath, Cluster dissection and analysis, theory, FORTRAN programs and examples. Halsted Press, John Wiley and Sons, NY (1985).
6. J.A. Hartigan, Cluster algorithms. John Wiley and Sons, NY, 1975.
7. G.C. Salzman, C.C. Stewart, R.E. Duque, "Expert Systems for Flow Cytometry Data Analysis: A Preliminary Report," New Technologies in Cytometry and Molecular Biology, Gary C. Salzman, Editor, Proc. SPIE 1206, (in press) 1990.
8. J. Giarratano and G. Riley, Expert Systems, Principles and Programming. PWS-KENT Publishing Co., Boston, 1989.
9. E.H. Shortliffe and B.G. Buchanan, "A model of inexact reasoning in medicine," Mathematical Biosciences 23, 1975.
10. E. Rich, Artificial Intelligence, McGraw-Hill, NY, 1983.
11. J.E. Weber, P.H. Bartels, W. Griswold, W. Kuhn, S.H. Paplanus, A.R. Graham. Colonic Lesion Expert System. Performance Evaluation. Analytical and Quantitative Cytology 10, 150-159, 1988.

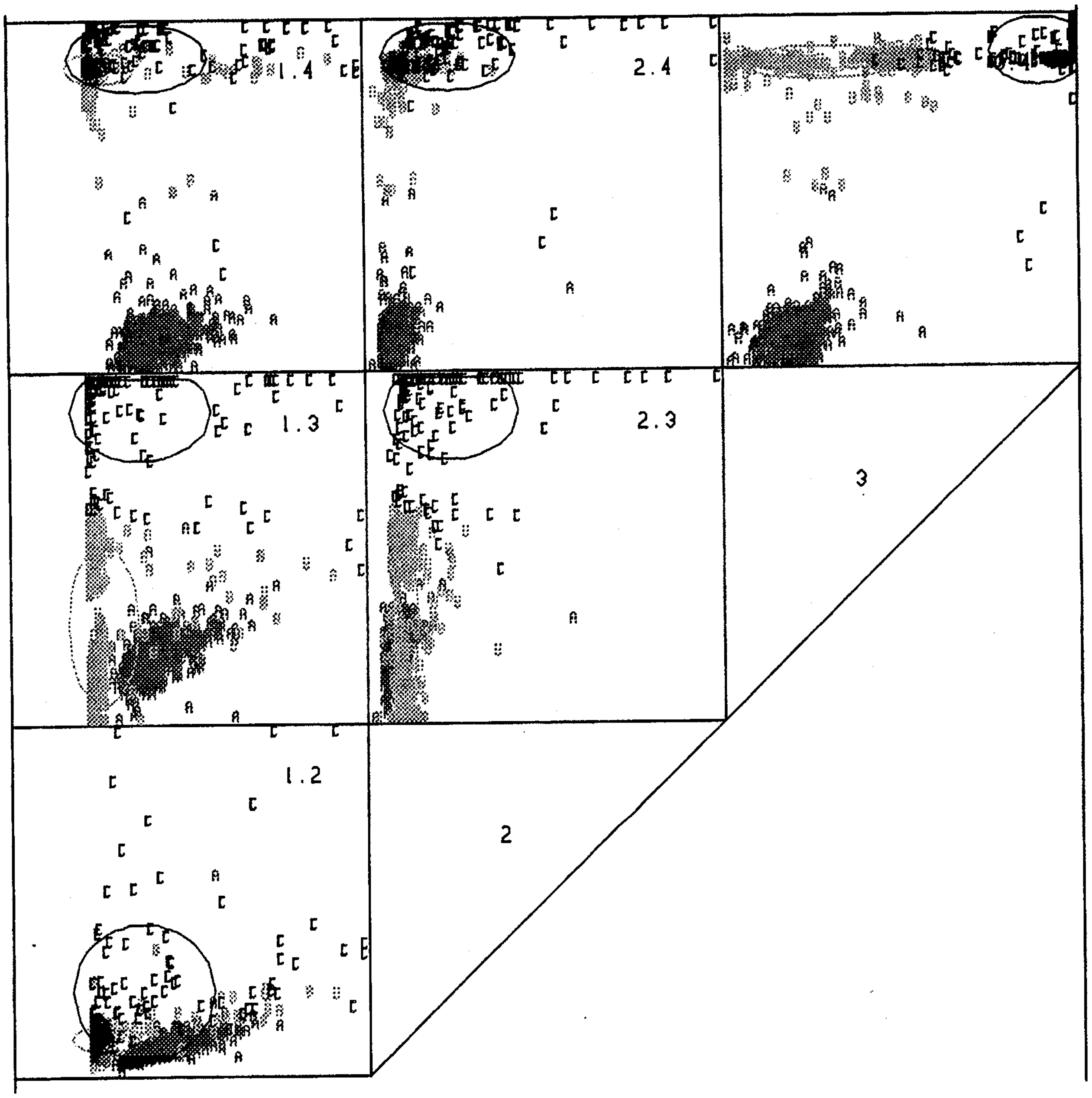


Figure 1. Bivariate dot plot of of four variate data flow cytometry data from a leukemia patient. The two numbers in the upper right hand corner of each box (X.Y) are the x-axis variate number followed by the y-axis variate number. Cluster analysis has divided the data into three clusters. Each data point is labeled with a letter designating its cluster membership. Variate one is forward scatter (FSC), variate 2 is side scatter (SSC), variate three is fluorescence one (FL1), a monoclonal antibody, and variate four is fluorescence two (FL2), which is the dye propidium iodide (PI) that stains the nuclei of dead cells. The population A in bivariate 3.4 in the upper right hand corner is "negative" for the monoclonal antibody of interest (FL2), and "negative" for PI.

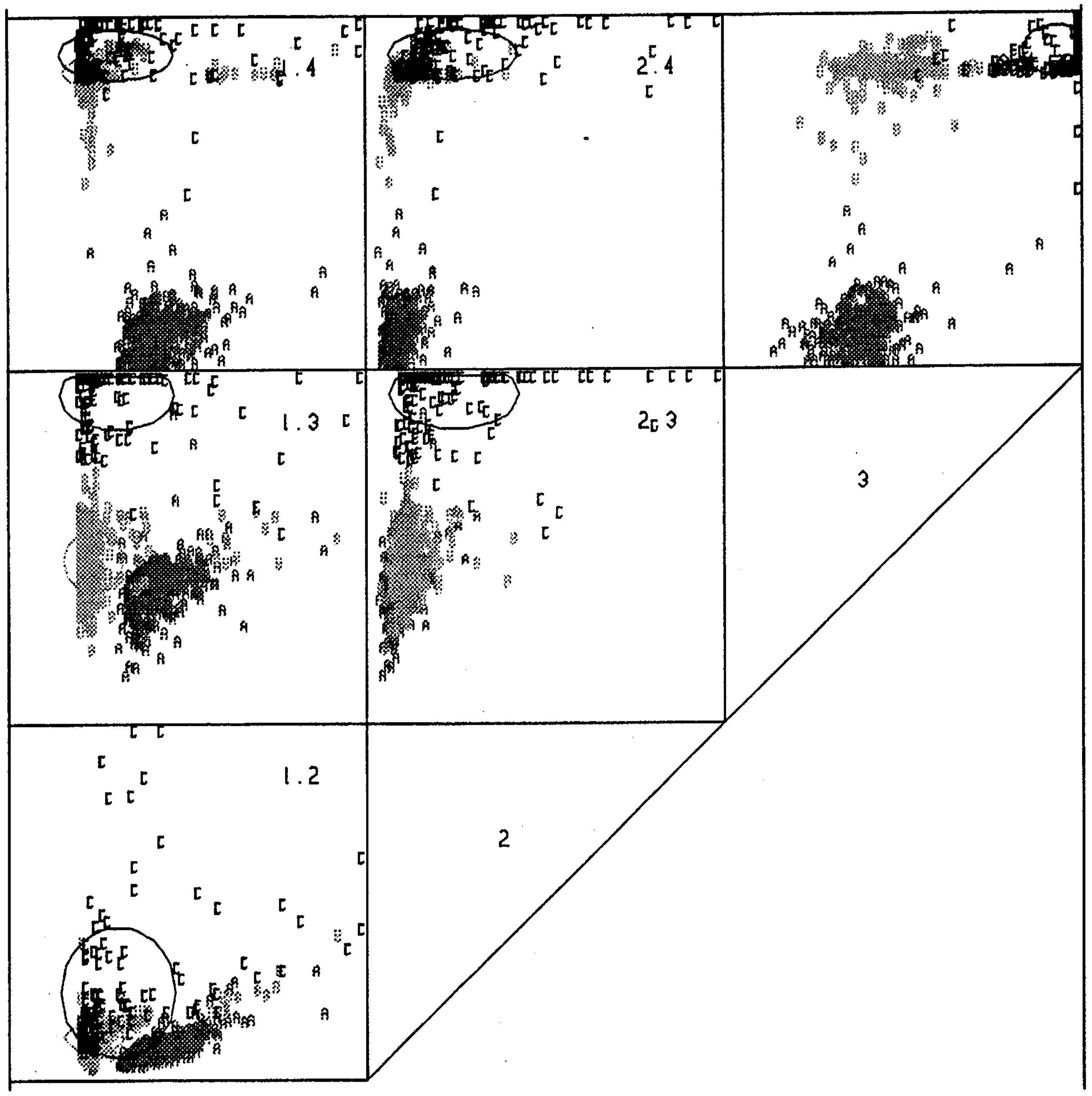


Figure 2. Same as Figure 1 except that a different monoclonal antibody has been used for FL1 (variate 2). Here cluster A in bivariate 3,4 is "dim" for the antibody.

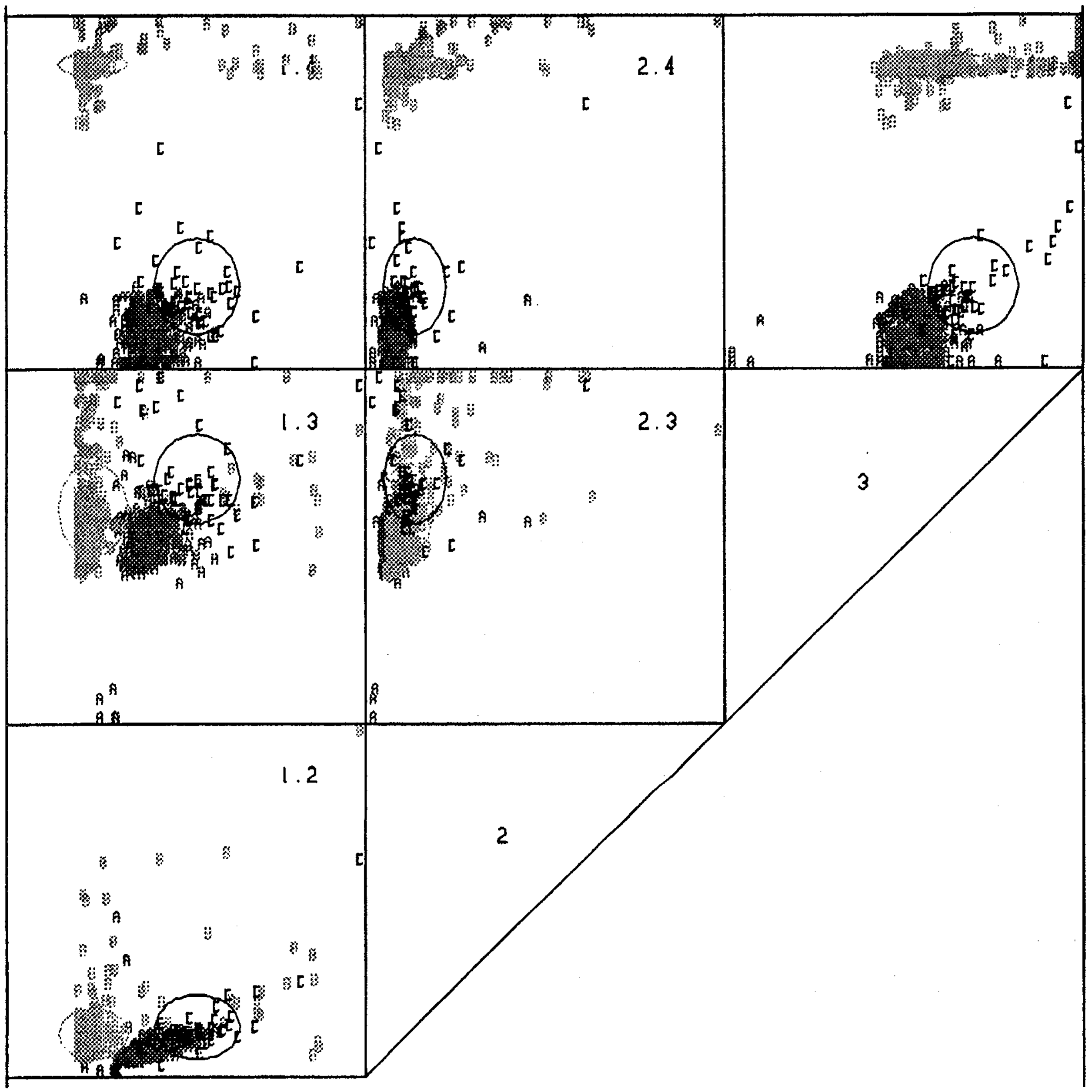


Figure 3. Same as Figure 1 except that another different monoclonal antibody has been used for FL1 (variate 2). Here cluster A in bivariate 3.4 is "bright" for the antibody.

# A CLIPS EXPERT SYSTEM FOR MAXIMIZING ALFALFA (*MEDICAGO SATIVA* L.) PRODUCTION

B.A. Engel, D.D. Jones, and R.L. Rhykerd
L.M. Rhykerd, C.L. Rhykerd, Jr. and C.L. Rhykerd[1]

## Abstract

An alfalfa management expert system originally developed by Purdue University agricultural scientists on the PC Plus™* expert system shell from Texas Instrument has been updated and successfully converted to CLIPS (C Language Integrated Production System). This reduces the cost and legal restrictions associated with making the expert system available to agribusiness industries, extension personnel and farm managers and operators. The expert system includes recommendations concerning soil drainage, liming, P and K fertilization, weed control, variety selection and seeding rate including pure live seeds.

## Introduction

Alfalfa is one of a very few crops grown in every state in the United States. It is our most productive legume and, under good management, will produce more protein per acre than any other crop. Alfalfa is used primarily as a livestock feed and when managed properly has the highest feeding value of all forage species. The annual production of alfalfa in the United States is in the neighborhood of 80 million tons. At the present market price of about $100 per ton, the annual value of the alfalfa crop in the United States is approximately 8 billion dollars.

Forage experts have recently developed technology that will increase alfalfa yields as much as 100 percent or more. Unfortunately many seed dealers and agricultural advisors are not knowledgeable about the management inputs required to produce high alfalfa yields. The availability of an alfalfa management expert system would enable seed dealers and agricultural advisors to provide the latest alfalfa technology to alfalfa producers.

The knowledge-based expert system (ES) is the practical application of artificial intelligence research. Development of expert systems was initiated only about a decade ago and the agricultural application of expert systems has taken place primarily in the past six years.

Personnel in the Agricultural Engineering Department at Purdue University have been leaders in the development of artificial intelligence applications in agriculture. The department has a long history in the application of computer technology to agriculture. The development of agricultural ES applications began at Purdue in the Agricultural Engineering Department in 1984. One of the early applications was the Grain marketing Advisor (GMA) [1]. GMA assists grain producers in selecting the best grain marketing strategy for the situation described. Numerous additional agricultural ES applications have been developed and continue to be developed with Purdue agricultural scientists serving as domain experts.

1 Departments of Agricultural Engineering and Agronomy, Purdue University, West Lafayette, Indiana 47907

* Reference to a company or trade name does not imply approval or recommendation of a product by Purdue University.

Development of the alfalfa establishment and management expert system reported here dates back to 1988 [2]. The primary object of the expert system was to assist Indiana farmers in selecting the best alfalfa variety, or varieties, to meet the farmer's particular site specific requirements. Knowledge engineering, the process of building an expert system, requires a development tool (expert system shell), a domain expert, and a knowledge engineer [3]. The domain expert for the original PC Plus™ program was C.L. Rhykerd and the knowledge engineers were R.L. Rhykerd, L.M. Rhykerd and C.L. Rhykerd, Jr. C.L. Rhykerd has been involved with the research and teaching of alfalfa production for over 30 years.

It quickly became apparent that this variety selection expert system could easily be expanded to make other management decisions at the time of seeding alfalfa to further increase production [4]. The alfalfa establishment expert system reported in 1989 [5] worked well for making recommendations for the successful establishment of alfalfa in Indiana. However, the costs associated with the purchase of a commercial shell and required runtime fees severely limited the potential for distribution and use of this alfalfa establishment expert system [6].

## Development of Alfalfa Expert System

Because of the problems associated with the practical application of expert systems, the Artificial Intelligence Section at NASA/Johnson Space Center developed CLIPS. Since CLIPS is not copyrighted or licensed, it eliminates most of the costs that have been associated with the use of commercial shells and runtime fees and is easily integrated with existing or conventional software systems. The potential significance of the development of CLIPS is stressed by Robert T. Savely, Head of Artificial Intelligence Section at Lyndon B. Johnson Space Center "Although CLIPS is but one of the first steps, it is an important step in the evolution of a technology that may be the most important advance in the history of mankind" [7].

The objective of this study was to convert the alfalfa establishment expert system built on the PC Plus™ expert system shell from Texas Instruments [5] to CLIPS. The conversion to CLIPS permitted updating and expansion of the knowledge base including the calculation of pure live seed since mathematical calculations were difficult to make in the commercial expert system shell. PC Plus™ is a parameter driven, forward and backward chaining, rule-based expert system tool and CLIPS is a forward chaining, pattern matching, rule-based expert system tool [6].

The knowledge base for the alfalfa establishment expert system was developed for Indiana but can easily be modified for another state or region. Management considerations and the sequence in which they were built into the knowledge base are as follows:

- soil drainage (phytophthora resistance)
- soil pH
- soil P test
- soil K test
- use of crop
- weed control
- expected longevity of stand (anthracnose resistance)
- variety recommendation
- method and rate of seeding
- pure live seed (% germination and purity provided by user)

## Knowledge Base Conversion

The first step in the conversion of knowledge bases is the transformation of the knowledge representation syntax [8]. Because PC Plus™ and CLIPS both use parameter-like declarations and production rules to represent knowledge, the transformation was literal. Rules of the CLIPS version are very similar to the rules in PC Plus™.

To illustrate, example syntax in both development tools will be briefly reviewed. PC Plus™ has a very structured knowledge representation. Parameters are described in frame-like structures with attributes that specify their *value*, the *prompt* by which their value is asked for or the rules that update their value, a *help* statement, a description of their *possible values*, *what rules use them* and *other characteristics*. The production rules are used to update parameter values or to inform the user of results. They contain a premise, a conclusion and other characteristics that specify when and how they should be used [9]. For example, the declaration of the parameter SOILPH in PC Plus™ is:

```
(SOILPH
   USED-BY  (RULE001  RULE006  RULE008  RULE004  RULE009  RULE010  RULE011  RULE012
   RULE013  RULE015  RULE016  RULE017  RULE018  RULE019  RULE020  RULE005  RULE014
   RULE021)
   TRANSLATION (The pH of the soil in the field to be seeded.)
   PROMPT (Is the pH of the soil from the field you want to seed above 6.6?)
   TYPE YES/NO)
```

A typical example of a PC Plus™ rule is RULE001 from the alfalfa establishment expert system. It indicates that alfalfa should not be seeded in a field when the soil pH is less than 6.6.

```
(RULE001
   PREMISE ($AND (NOTSAME FRAME SOILPH YES)) SUBJECT ALFALFA-RULES)
   ACTION (DO-ALL (CONCLUDE FRAME SEED
      "Do not seed alfalfa until the pH value meets the suggested
           value." TALLY 100)
                  (CONCLUDE FRAME ADVICE
      "The pH of the soil should exceed 6.6 in order to grow a productive
           crop of alfalfa." TALLY 100))
```

CLIPS uses facts and rules. Facts can be used to describe parameter values and properties. Rules are used to retract, add or modify facts and for input/output purposes. Since no predetermined parameter description was imposed by the language, one was developed [10]. Thus the declaration of the parameter SOILPH in CLIPS becomes:

```
(SOILPH
   prompt "Is the pH of the soil from the field you want to seed above 6.6?"
   expect YES NO
   help
   why
   value
   value-type YES
   default
   range
   certainty-range
   unknown
   gprompt  ghelp  gwhy)
```

One of the limitations of CLIPS for agricultural ES is its lack of an end user interface [10]. A package of C-based PC user interface development functions were developed and integrated into CLIPS [10]. These functions are driven with a parameter-like fact structure as shown above. The string following *prompt* is used to query the user for the parameter value. Values following *expect* are placed in a menu from which the user selects the appropriate value. The interface functions return the value selected from the menu which becomes part of a fact in CLIPS. A complete description of the interface functions and their use is provided in Engel et al. [10].

The CLIPS syntax of RULE001 is:

```
(defrule RULE001
  ;group of rules:ALFALFA-RULES
    (SOILPH NO)
  =>
    (assert (SEED "Do not seed alfalfa until the pH value meets
        the suggested value." cf 100))
    (assert (ADVICE " The pH of the soil should exceed 6.6 in order
        to grow a productive crop of alfalfa." cf 100)))
```

Two programs were developed to convert knowledge bases from PC Plus™ into CLIPS. One transforms rules, and the second transforms parameter declaration frames into CLIPS facts. The use of these tools allows the syntactic transformation of a moderately large knowledge base (around 100 parameters and 200 rules) in a couple of hours instead of several days.

Besides the processing speed, the automation of knowledge base transformation has other advantages.

- It allows identification of syntax errors in the original knowledge base.
- If there are no syntax errors and if the program is correctly written, it insures correct output without typing errors.
- Translation errors are more easily detected as they are repeated for all similar inputs. A manual transformation with extensive editing and typing can create additional errors, each one being isolated and difficult to detect.

The conversion of the syntax is only the first step of the transformation process. Next, rules must be added to insure that logical flow of the program is conserved since the inferencing strategy is different. The original expert system engine used a backward chaining inferencing method while CLIPS provides only a forward chaining inference engine. Since most of the questions in the original knowledge base were asked in the same order no matter what the user's responses, only a few additional rules were required in the converted knowledge base to maintain the same logical flow.

In the process of knowledge base transformation, some knowledge is lost because of the differences in knowledge representation and reasoning process between the expert system shells. However, some can also be gained through the correction of errors and updating of the original knowledge base. The knowledge that was lost in this knowledge base conversion was the ability to explain *how* answers are determined and thus was minimal. Our experiences indicate this feature is seldom used except by the knowledge engineer during knowledge base debugging. The conversion process discovered several errors in the original knowledge base. These errors consisted of syntax errors and redundancies or inconsistencies in rules. After re-examining the alfalfa establishment knowledge base, the expert discovered that not all of the rules were necessary and was able to better define the knowledge base. The CLIPS development tool permitted the addition of the calculation of pure live seed. Providing the alfalfa producer a seeding rate based on pure live seed will increase his chances of obtaining an excellent stand of alfalfa. The knowledge base conversion process provides an opportunity for an expert to rethink and to update his domain problem solving knowledge.

The final step in knowledge base conversion is testing. Since a relatively small number of inputs are required by the expert system, all possible input combinations were tried with the original and converted knowledge bases. The converted knowledge base for alfalfa management provided the same results as the original. For a more in depth discussion of the knowledge base transformation process, see Engel, et al. [8].

This alfalfa management CLIPS expert system is in its final stages of development and should be ready for use in the next growing season. It can easily be adapted to other states or farming regions by modifying the If-Then Rules and adding, or deleting, alfalfa varieties to the knowledge base.

The potential application of ES technology in agriculture are numerous. Commercial ES development and delivery tools have limited the application of this technology in agriculture. CLIPS overcomes many of these limitations. The alfalfa management system demonstrates the potential of CLIPS-based ES for agriculture.

## Literature Cited

1. Thieme. R.H., J.W. Uhrig, R.M. Peart, A.D. Whittaker, and J.R. Barrett. 1987. Expert system techniques applied to grain marketing analysis. *Computers and Electronics in Agriculture* 1:299-308.
2. Rhykerd, R.L., D.D. Jones and C.L. Rhykerd. 1988. An expert system for selecting alfalfa varieties. Proc. Amer. Forage & Grassl. Council, pp. 135-137.
3. Barrett, J.R. and D.D. Jones. 1989. Knowledge engineering in agriculture. Monograph No. 8, Amer. Soc. Agric. Engineers, St. Joseph, MI 214 pp.
4. Rhykerd, R.L., C.L. Rhykerd, Jr., D.D. Jones and C.L. Rhykerd. 1988. Using artificial intelligence to maximize alfalfa production. Proc. Ind. Acad. Sci. (In Press).
5. Rhykerd, R.L., C.L. Rhykerd, Jr., C.L. Rhykerd and D.D. Jones. 1989. An expert system for the successful establishment of alfalfa. Proc. Amer. Forage & Grassl. Council, pp. 230-234.
6. Engel, B.A. and D.D. Jones. 1989. Expert systems development in CLIPS. Paper No. 89-7584, Amer. Soc. Agric. Engineers. Presented at Winter Meeting, New Orleans, LA.
7. Giarratano, J. and G. Riley. 1989 Expert systems -- principles and programming. PWS-KENT Publishing Co., Boston, MA. 632 pp.
8. Engel, B.A., C. Baffaut, J.R. Barrett, J.B. Rogers, D.D. Jones. 1990a. Knowledge transformation. *Applied Artificial Intelligence* 4:67-80.
9. Personal Consultant Plus™ (1986), Texas Instruments Inc., Austin, Texas.
10. Engel, B.A., C.C. Rewerts, R. Srinivasan, J.B. Rogers, and D.D. Jones. 1990b. CLIPS interface development tools and their application. Proceedings of First CLIPS Users Conference.

# A DECISION SUPPORT SYSTEM FOR DELIVERING OPTIMAL QUALITY PEACH & TOMATO

C. N. THAI
Assistant Professor

J. N. PEASE
Programmer III

Agricultural Eng. Dept.

R. L. SHEWFELT
Associate Professor
Food Sci. & Tech. Dept.

UNIVERSITY OF GEORGIA
GEORGIA AGRICULTURAL EXPERIMENT STATION
GRIFFIN, GA 30223–1797.

## INTRODUCTION

Several studies have indicated that color and firmness are the two quality attributes most important to consumers in making purchasing decisions of fresh peach and tomato (*References 1 & 2*). However, at present, retail produce managers do not have the proper information for handling fresh produce so it has the most appealing color and firmness when it reaches the consumer. This information should help them predict the consumer color and firmness perception and preference for produce from various storage conditions. Since 1987, for 'Redglobe' peach and 'Sunny' tomato, we have been generating information about their physical quality attributes (firmness and color) and their corresponding consumer sensory scores (*References 3, 4, 5 & 6*). This article reports on our current progress toward the goal of integrating such information into a model–based decision–support system for retail level managers in handling fresh peach and tomato.

## MATERIALS AND METHODS

First of all, we wanted to use expert systems technology to create our prototype decision–support system for ease of updating future knowledge sources and friendly user interface. We also wanted our system to be "self–sufficient", i.e. no need for a run–time shell, therefore we needed to embed the inference engine and knowledge base into our particular application. For these reasons, we chose the CLIPS system, version 4.3, from N.A.S.A. (*Reference 7*) as our expert system development tool.

Our expert system is named FP–DSS (for Fresh Produce Decision Support System) and has 3 major components:

A) Physical Models.
B) Sensory Models.
C) Optimality Conditions.

### A) PHYSICAL MODELS

#### Peach firmness and color

Changes in firmness and color of 'Redglobe' peaches were studied under different constant and variable storage temperatures from $5^{o}C$ to $21^{o}C$. Firmness was measured as the maximum force F (in Newtons) during destructive punching by a 10 mm diameter steel probe. Color was measured in the Hunter (L,a,b) color space (*Reference 8*) as hue angle H ($= \tan^{-1}(b/a)$), chroma C ($= \sqrt{a^2 + b^2}$), and value L (= L, unchanged). The physical definitions of the three color components Hue, Chroma and Value are as follows:

a) Hue is actually what is commonly referred to as color. Hue angle varies continuously from $0^{o}$ to $360^{o}$. A hue angle of $0^{o}$ (or $360^{o}$) corresponds to red, while a hue angle of $90^{o}$ corresponds to yellow and $180^{o}$ corresponds to green, and so forth with blue at $270^{o}$.

b) Chroma measures the vividness of colors and varies from 0 to 60. A low chroma makes all colors look grayish, and a high chroma yields strong and distinct colors.

c) Value measures the lightness of colors and varies from 0 to 100, with 0 corresponding to black and 100 corresponding to white.

The following relationships were found to apply:

*for firmness F*

$$F = F_o \cdot e^{[m \cdot T + n] \cdot d} \quad \text{.......} \quad [1]$$

where:

$F$ = predicted firmness at day d (N).
$F_o$ = initial firmness value at day d = 0 (N).
d = storage time (day).
T = storage temperature ($^{o}C$).
m and n are constant coefficients reported in *Reference 9*.

(with goodness of fit $R^2 = 0.7182$).

*for hue angle H*

$$H = H_o - (a_p \cdot T + b_p) \cdot d \quad \text{.......} \quad [2]$$

where:

H = predicted hue angle at day d (degree).
$H_o$ = initial hue angle at day d = 0 (degree).
$a_p$ and $b_p$ are constant coefficients reported in *Reference 9*.

(with goodness of fit $R^2 = 0.9731$).

*for chroma C*

$$C = k_i + k_{co} \cdot C_o + k_{ho} \cdot H_o + k_1 \cdot H + k_2 \cdot H^2 \quad \text{.......} \quad [3]$$

where:

C = predicted chroma at day d (dimensionless).
$C_o$ = chroma at day d = 0 (dimensionless).
$k_i$, $k_{co}$, $k_{ho}$, $k_1$ and $k_2$ are constant coefficients reported in *Reference 9*.

(with goodness of fit $R^2 = 0.7243$).

*for value L*

$$L = q_i + q_{lo} \cdot L_o + q_{ho} \cdot H_o + q_1 \cdot H + q_2 \cdot H^2 + q_3 \cdot H^3 \quad [4]$$

where:

$L$ = predicted value at day d (dimensionless).
$L_o$ = Value at day d = 0 (dimensionless).

$q_i$, $q_{lo}$, $q_{ho}$, $q_1$, $q_2$, and $q_3$ are constant coefficients reported in *Reference 9.*

(with goodness of fit $R^2 = 0.7686$).

**Tomato firmness and color**

*Reference 4* reported on a 1987 study on color changes of two types of 'Sunny' tomatoes, "vine–ripened" and "ethylene–ripened", under constant and variable storage temperatures in the range of 15°C to 25°C. In that study, polynomial regression equations were used to model H, C and L with $R^2$ values from 0.54 to 0.66. *Reference 6* has details of our 1989 study on the changes in firmness and color of 'Sunny' tomatoes under constant and variable storage temperatures, over the same range of 15°C to 25°C. Firmness and color parameters are defined as previously for peach. In *Reference 6*, new and more accurate color models were developed, thus data from *Reference 4* were re–analysed under this new modeling framework and used in FP–DSS.

The following relationships were found to apply:

*for firmness F*

$$F = F_o \cdot e^{(\omega \cdot T + \epsilon) \cdot d} \quad [5]$$

where:

$\omega$ and $\epsilon$ are constant coefficients reported in *Reference 9.*

(with goodness of fit $R^2 = 0.5716$).

***Eq. [5] is used for both types of tomatoes in FP–DSS.***

*for hue angle H*

$$H = \frac{H_{mg} + H_{rr} \cdot \left[ \frac{H_{mg} - H_o}{H_o - H_{rr}} \right] \cdot e^{[(aT^2 + bT + c) \cdot d]}}{1 + \left[ \frac{H_{mg} - H_o}{H_o - H_{rr}} \right] \cdot e^{[(aT^2 + bT + c) \cdot d]}} \quad [6]$$

where:

$H_{mg}$, $H_{rr}$, a, b, c are constant coefficients reported in *Reference 9.*

(with goodness of fit $R^2 = 0.9759$ for vine–ripe and 0.9030 for ethylene–ripe).

*for chroma C*

$$C = k_o + k_1 \cdot H + k_2 \cdot H^2 + k_3 \cdot H^3 + k_4 \cdot H^4 + k_5 \cdot H^5 + k_6 \cdot H^6 \quad [7]$$

where:

$k_o$, $k_1$, $k_2$, $k_3$, $k_4$, $k_5$, and $k_6$ are constant coefficients reported in *Reference 9.*

(with goodness of fit $R^2 = 0.9016$ for vine–ripe and 0.9541 for ethylene–ripe).

_for value L_

$$L = L_o \cdot \left[ \frac{1.0 - \delta \cdot e^{-(\gamma \cdot H)}}{1.0 - \delta \cdot e^{-(\gamma \cdot H_o)}} \right] \quad [8]$$

where:

$\gamma$ and $\delta$ are constant coefficients reported in *Reference 9.*

(with goodness of fit $R^2 = 0.9304$ for vine–ripe and 0.8871 for ethylene–ripe).

In short, with given product initial firmness and color attributes, the above models can predict the future development of those physical quality attributes. Next we need to link these physical attributes to their sensory equivalents.

## B) SENSORY MODELS

### Peach firmness and color

The relevant data set comes from *Reference 5* and consists of 29 peaches, each with their physical firmness and color measurements and corresponding sensory firmness and color scores. These sensory scores were the average score given by eight sensory panelists judging the same fruit. The panelists were instructed to place a mark for their judgment on a line 150 mm in length with anchor points at 138 for excellent color or real firm, at 75 for good color or moderately firm and at 12 for fair color or not firm. *Reference 5* presented the statistical approach to modeling, whereas the best polynomial regression equation linking sensory scores to physical measurements was determined,and by using the ***author's judgments*** for the selection of various regressors to be included in the regression model. These outputs will be referred to as "regressed scores". The regression equations for sensory firmness and color were of a linear type:

$$SF = 85.01 + 0.3598 \cdot F \quad [9]$$

where:

SF = sensory firmness score (mm). (with $R^2 = 0.6095$), and,

$$SC = 377.07 - 3.1510 \cdot H \quad [10]$$

where:

SC = sensory color score (mm). (with $R^2 = 0.5900$).

In this study, we also use "Neural Networks" as a pattern recognition and modeling tool. For lack of space, we have to refer the reader to *References 10 and 11* for background materials on this subject. For peach firmness, we used a feedforward network having 1 input neuron for the physical firmness measurement (i.e. puncture force measurement), 2 hidden layers of 2 neurons each, and 1 output neuron for the sensory firmness score. For peach color, we used a network having 3 input neurons for the physical color measurements (i.e. Hue, Chroma and Value), 2 hidden layers of 6 neurons each, and 1 output neuron for the sensory color score. The Generalized Delta Rule procedure for supervised learning was used (*Reference 12*) in determining the optimal weights for the interneurons connections. These outputs will be referred to as "learned scores". Using SAS statistical software (*Reference 13*), Pearson correlation coefficients were computed to evaluate the linkage between actual sensory scores and respectively with regressed scores and learned scores:

Peach firmness

actual–regressed : $R^2 = 0.7807$.

actual–learned : $R^2 = 0.8081$.

Peach color

actual–regressed : $R^2 = 0.7681$.

actual–learned : $R^2 = 0.7877$.

Thus, in general, we have achieved better correlation with neural nets.

**Tomato firmness and color**

A similar scale to the one for peach was used for sensory evaluation of tomato firmness and color, except the anchor points were set up differently: 138 for full red or very firm, 75 for pink to light red or moderately firm and 12 for not red or very soft. *Reference 3* reported the following equation for tomato sensory firmness scores vs. their physical measurements:

$SF = 10.3 + 1.70 \cdot F$ .................................................................................................... [11]

(with $R^2 = 0.81$).

Recently, we have obtained a new data set (not yet published) consisting of 115 tomatoes, each with their **physical color** measurements (H,C,L) and corresponding **sensory color** scores by each of eight panelists judging the same fruit. *Reference 6* showed that C was strictly dependent on H, and that L was both dependent on its initial value $L_o$ and H (see Equations [6], [7] & [8]). Thus, we formulated a regression equation for sensory color scores SC in terms of hue H, value L and interaction H*L, and obtained the following results with this new data set and using the average score from the eight panelists:

$SC = 305.3583 - 2.4171 \cdot H - 5.2648 \cdot L + 0.04266 \cdot H \cdot L$ ................................................ [12]

(with $R^2 = 0.9619$).

Equations [11] and [12] are used in FP–DSS, as the "regressed scores".

Applying the same neural net architecture as defined for peach color (3–6–6–1), we obtained the following **Pearson correlation coefficients** between **actual** individual panelist scores and respectively with scores as given by Eq. [12] and the neural net:

Regressed scores

**0.95695** – **0.96075** – 0.94983 – **0.95754** – 0.96803 – 0.91806 – 0.94926 – **0.95870**.

Learned scores

0.95591 – 0.95903 – **0.95456** – 0.95689 – **0.96894** – **0.92040** – **0.95561** – 0.95384.

Thus the regression and neural net approaches are equivalent to each other overall, as each approach is better than the other one only for 50% of the time.

## C) OPTIMALITY CONDITIONS

Production rules were used to implement optimality conditions which are defined as follows:

**Peach:** $50 \leq SF \leq 100$ and $100 \leq SC \leq 150$.
(Learned sensory scores used in FP–DSS).

**Tomato:** $50 \leq SF \leq 100$ and $50 \leq SC \leq 100$.
(Regressed sensory scores used in FP–DSS, as we did not have a neural net for tomato firmness).

## FP–DSS USAGE

FP–DSS can be run on any MS–DOS personal computer with VGA graphics recommended, but not necessary. The executable file is 329,016 bytes long, and the ASCII rule base is 12,825 bytes long. A typical user session with FP–DSS can be described as follows:

a) Display ***Welcome*** banner while the knowledge base is being loaded and parsed.

b) Show ***Menu*** to choose the working scenario among 3 possible & mutually exclusive choices:
   1 – Find optimal sale dates for a given storage temperature.
   2 – Find optimal storage temperatures for a target sale date.
   3 – To exit FP–DSS.

c) If **scenario 1** is chosen, ***Menus*** instruct the user to enter the considered **produce type** among "*Peaches*", "*Vine–ripe Tomatoes*" and "*Ethylene–ripe Tomatoes*", next to enter the **initial color** and **firmness** measurements, and then the **preferred storage temperature**. Safeguards are built in to guide the user in giving realistic values. Next, FP–DSS passes these parameters to the main C program which then sets up the necessary neural networks and reads in previously learned weights for the "sensory" models. Then, FP–DSS uses previously discussed physical and sensory models to update the knowledge base and display the resulting sensory scores for firmness and color for each day from day 0 to day 9 (A VGA graphics capability is needed to see the predicted color for each day). It also asserts these scores into the CLIPS facts list, and then returns control to CLIPS which activates rules to decide on and to display any optimal solution found. Finally, CLIPS clears all facts, resets the facts list and returns to Step **b)**.

d) If **scenario 2** is chosen, similar actions take place, except the **targeted sale date** is asked for instead of the **preferred storage temperature**. Peach has possible storage temperatures from $5^{o}$C to $21^{o}$C, while Tomato storage temperatures range from $15^{o}$C to $25^{o}$C. The knowledge base has updated information on all possible temperatures by steps of $1^{o}$C, but for lack of space on the computer screen, only selected storage temperatures results are displayed. However, all temperatures are considered in determining the optimal storage temperature(s). Finally, FP–DSS frees the knowledge base, resets CLIPS and returns to Step **b)**.

e) If **scenario 3** is chosen, FP–DSS returns to DOS.

Table 1 lists the results of a session with FP–DSS with different produce types and initial conditions. These results show that it is possible, but not always, to get a product that has both firmness and color ratings preferred by consumers.

## PROGRAMMING HIGHLIGHTS

As our previous experiences were in backward chaining production systems, our thought process had to be completely reoriented when working with CLIPS. Flow control is no longer dependent on **rules positioning**, but on **control variables** (or facts) on the LHS.

Embedding CLIPS into standard C programs was quite painless, although we would like to be able to pass **integers** in future versions. We have created 2 *user's functions*: one called ***c–bounds*** to compute the upper and lower 95% confidence levels for Tomato Chroma C based on the given initial hue angle $H_0$, the other called ***generate*** to create the *future* knowledge base by applying the above physico–sensory models at all allowed time periods or storage temperatures. This knowledge base consists of several multifield facts of the type *"gentd ... "* representing the scenario chosen, the appropriate day or temperature, both regressed and learned sensory scores for firmness and color.

We have also noted that care needs to be used when asserting keyboard input through expansion of "=" and "***read***" into a template field. For example, we had defined a template as follows:

```
(deftemplate session
        (field scenario
        (type NUMBER) (default 0))
        ... )
```

and the following rule to ask the user for input into the field *scenario*:

```
(defrule get_scenario_wrong
        ?st-dss <- (start-dss)
        =>
        (system "cls")
        (fprintout t "What would you like to do ?" crlf)
        (fprintout t "1 - Find sale dates for given storage temperature," crlf)
        (fprintout t "2 - Find storage temperatures for given sale date," crlf)
        (fprintout t "3 - To exit." crlf crlf)
        (retract ?st-dss)
        (assert (session (scenario =(read))))
        (assert (scenario-got)))
```

The above rule will assert the user input into the field ***scenario*** with no error message, however its value is being stored as a character thus creating errors when numerical operations are performed with it at a later stage. Thus it seems that templates do not enforce type checking rigorously enough. The following rule works better by first *binding* *(read)* into a variable *?sc#* before asserting it into the template *(session)*, thus ensuring the **numerical** status of the field *scenario*:

```
(defrule get_scenario_right
        ?st-dss <- (start-dss)
        =>
        (system "cls")
        (fprintout t "What would you like to do ?" crlf)
        (fprintout t "1 - Find sale dates for given storage temperature," crlf)
        (fprintout t "2 - Find storage temperatures for given sale date," crlf)
        (fprintout t "3 - To exit." crlf crlf)
        (retract ?st-dss)
        (bind ?sc# (read))
        (if (or (or (= ?sc# 1) (= ?sc# 2)) (= ?sc# 3))
        then
                (assert (session (scenario ?sc#)))
                (assert (scenario-got))
        else
                (fprintout t "!!! INCORRECT SCENARIO ENTERED, please reenter")
    (readline)
    (assert (start-dss))))
```

The previous rule also shows the usefulness of the capability to use "***if ... then ... else***" constructs on the RHS as it allows combining the tasks of ***asking for input*** and ***input checking*** in the same rule.

Finally, as we wanted our application to loop back to the starting rule after each user session, we had to clear and then reset our facts list with a few ending rules. For this task, we had to combine ***control variables*** and ***salience*** in order to achieve the proper sequence of events and to circumvent the side effect of firing unwanted rules. For example, consider the following rules:

```
(defrule optimality_for_tomato_scenario_2                    ;this rule is applied to
        (sim-done)                                           ;each gentd fact.
        (session (scenario 2) (produce 2|3) (t-or-d ?sd))
        (gentd ? ?td ?rsc ? ?rsf ?)
        =>
        (if (and (and (>= ?rsc 50) (<= ?rsc 100))
                 (and (>= ?rsf 50) (<= ?rsf 100)))
        then
                 (assert (optimal-found))
                 (fprintout t "Optimal storage temperature= " ?td " C for D= " ?sd crlf)
        else
                 (assert (no-optimal))))

(defrule no_optimals                          ;this rule fires only after the previous rule
        (declare (salience -5))               ;applies to all gentd facts and only once,
        ?noopt <- (no-optimal)                ;as (no-optimal) is retracted after use.
        (not (optimal-found))
        =>
        (fprintout t "NO OPTIMAL SOLUTION WAS FOUND" crlf)
        (fprintout t "FOR THE GIVEN INITIAL CONDITIONS" crlf)
        (retract ?noopt))

(defrule clean_up_2                           ;to retract (no-optimal) if the previous rule
        (declare (salience -20))              ;is not fired because
        ?no-opt <- (no-optimal)               ;(optimal-found) exists.
        =>
        (retract ?no-opt))

(defrule clean_up_3                           ;to retract (optimal-found) last as not to
        (declare (salience -25))              ;trigger the firing of the rule no_optimals
        ?opt-fnd <- (optimal-found)           ;as a side effect.
        =>
        (retract ?opt-fnd))

(defrule clean_up_4                           ;loop to retract all gentd facts.
        (declare (salience -25))
        ?results <- (gentd ? ? ? ? ? ?)
        =>
        (retract ?results))
```

```
(defrule back_to_beginning                      ;to clear completely the facts list and to
        (declare (salience -30))                ;reset to start by asserting (start-dss).
        ?ses <- (session)
        ?sim <- (sim-done)
        =>
        (readline)
        (fprintout t crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf)
        (fprintout t crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf)
        (fprintout t "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" crlf crlf)
        (fprintout t "REINITIALIZING TO BEGINNING OF FP-DSS" crlf crlf)
        (fprintout t "!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!" crlf crlf)
        (fprintout t crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf crlf)
        (retract ?ses)
        (retract ?sim)
        (readline)
        (assert (start-dss)))
```

## CONCLUSIONS

Although economic components are not yet implemented in FP–DSS, the current version shows that it is physically feasible with current technology to manage the storage of Peach and Tomato so that both firmness and color attributes appeal to the consumer preferences. CLIPS version 4.3 still needs some type checking and a garbage collection utility for cases having recurring user sessions without exiting CLIPS, as found in this application.

## REFERENCES

1. Butler, C. L. 1988. What do consumers want when buying peaches ?. p 854–856. In: The Peach – world cultivars to marketing (N. F. Childers and W. B. Sherman, Eds.). Horticultural Publications, FL..

2. Resurreccion, A. V. A. and R. L. Shewfelt. 1985. Relationships between sensory attributes and objective measurements of postharvest quality of tomatoes. J. Food Sci. 50:1242–1245, 1256.

3. Shewfelt, R. L., S. E. Prussia, A. V. A. Resurreccion, W. C. Hurst, and D. T. Campbell. Quality changes of vine–ripened tomatoes within the postharvest handling system. J. Food Sci. 52:661–664,672.

4. Shewfelt, R. L., C. N. Thai, and J. W. Davis. 1988. Prediction of changes in color of tomatoes during ripening at different constant temperatures. J Food Sci. 53:1433–1437.

5. Thai, C. N. and R. L. Shewfelt. 1990. Peach quality changes at different constant storage temperatures: empirical models. Transactions of the ASAE (In Press).

6. Thai, C. N., R. L. Shewfelt, and J. C. Garner. 1989–b. Tomato color and firmness changes under different storage temperatures. ASAE Technical Paper 896596, ASAE, St. Joseph, MI 49085–9659.

7. National Aeronautics and Space Administration. 1989. CLIPS User's Guide, version 4.3. Artificial Intelligence Section, Lyndon B. Johnson Space Center.

8. Hunter, R. S. 1975. The measurement of appearance. John Wiley and Sons, Inc., New York, 348p.

9. Thai, C. N., J. N. Pease, and R. L. Shewfelt. 1989. Inventory control strategies for delivering optimal quality peach and tomato: Part I – Quality Management. ASAE Technical Paper No. 897566. ASAE, St. Joseph, MI 49085–9659.

10. Aleksander, I. (editor) 1989. Neural Computing Architectures. The MIT Press, Cambridge, Massachusetts, 320p.

11. Wasserman, P. D. 1989. Neural Computing, Theory and Practice. Van Nostrand – Reinhold, New York, 230p.

12. Pao, Yoh–Han. 1989. Adaptive Pattern Recognition and Neural Networks. 1989. Addison–Wesley, Reading, Massachusetts, 309p.

13. SAS Institute Inc. 1985. SAS User's Guide: Statistics, Version 5 Edition. SAS Institute Inc., Cary, North Carolina, 956p.

## TABLE 1

PREDICTED OPTIMAL SALE DATES AND STORAGE TEMPERATURES

(for given $F_o$, $H_o$, $C_o$, $L_o$)

| Produce Type | $F_o$ | $H_o$ | $C_o$ | $L_o$ | Scenario 1 Stor. T = 21°C | Scenario 2 Target D = 4 |
|---|---|---|---|---|---|---|
| Peach | 100 | 104 | 50 | 70 | None (> Day 9) | None (> 21°C) |
| | 100 | 90 | 50 | 70 | None (> Day 9) | None (> 21°C) |
| | 100 | 80 | 50 | 70 | Day 7 —> 9 | None (> 21°C) |
| | 80 | 80 | 40 | 60 | Day 7 —> 9 | None (> 21°C) |
| | 80 | 70 | 40 | 60 | Day 1 —> 9 | 5°C —> 21°C |
| Vine–ripe Tomato | 62 | 104 | 20 | 45 | Day 3 —> 4 | 16°C —> 21°C |
| | 62 | 90 | 19 | 45 | Day 2 —> 3 | 16°C —> 18°C |
| | 62 | 80 | 18 | 45 | Day 2 —> 3 | 16°C —> 17°C |
| | 62 | 70 | 17 | 45 | Day 2 | 16°C |
| | 50 | 60 | 16 | 45 | Day 1 —> 2 | 15°C —> 16°C |
| Gas–ripe Tomato | 62 | 120 | 20 | 45 | Day 6 —> 7 | None (> 25°C) |
| | 62 | 110 | 19 | 45 | Day 4 —> 5 | 19°C —> 25°C |
| | 62 | 100 | 18 | 45 | Day 3 —> 5 | 16°C —> 25°C |
| | 62 | 90 | 17 | 45 | Day 3 —> 4 | 16°C —> 25°C |
| | 50 | 80 | 17 | 45 | Day 2 —> 3 | 15°C —> 19°C |

**REPORT DOCUMENTATION PAGE**

| | | |
|---|---|---|
| 1. Report No. NASA CP 10049 | 2. Government Accession No. | 3. Recipient's Catalog No. |
| 4. Title and Subtitle First CLIPS Conference Proceedings August 13-15, 1990 | | 5. Report Date July 1990 |
| | | 6. Performing Organization Code PT4 |
| 7. Author(s) Joseph C. Giarratano Chris Culbert, NASA-JSC | | 8. Performing Organization Report No. S-609 |
| 9. Performing Organization Name and Address Software Technology Branch Lyndon B. Johnson Space Center Houston, TX 77058 | | 10. Work Unit No. |
| | | 11. Contract or Grant No. |
| 12. Sponsoring Agency Name and Address NASA-JSC Cosponsored by COSMIC | | 13. Type of Report and Period Covered Conference Publication |
| | | 14. Sponsoring Agency Code |

15. Supplementary Notes

16. Abstract

Papers presented at the First CLIPS Conference held at the Lyndon B. Johnson Space Center (JSC), August 13, 14, and 15, 1990, are documented herein. CLIPS is an expert system tool developed by NASA-JSC. During the three days of the conference, approximately 96 technical papers were presented by experts from NASA, other Government agencies, universities, and industry. Technical topics included CLIPS, expert systems, and robotics.

Subject category: 61

| | |
|---|---|
| 17. Key Words (Suggested by Author(s)) CLIPS, expert systems, knowledge-based systems, Space Shuttle, Space Station Freedom, intelligent tutors, verification and validation, simulation | 18. Distribution Statement Unclassified - unlimited |

| | | | |
|---|---|---|---|
| 19. Security Classification (of this report) Unclassified | 20. Security Classification (of this page) Unclassified | 21. No. of pages 943 | 22. Price |

For sale by the National Technical Information Service, Springfield, VA 22161-2171

JSC Form 1424 (Rev Jan 88) (Ethernet Jan 88)

**DO NOT REMOVE SLIP FROM MATERIAL**

Delete your name from this slip when returning material to the library.

| NAME | DATE | MS |
|---|---|---|
| ~~Sand Hurst~~ | ~~7/28/92~~ | ~~130~~ |
| Library | | 185 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

NASA Langley (Rev. Dec. 1991) RIAD N-75