AN EXPERT SYSTEM ARCHITECTURE FOR COPING WITH COMPLEXITY IN COMPUTER-AIDED CONTROL ENGINEERING

John R. James, Department of Electrical Engineering, United States Military Academy, West Point NY 10996 USA

James H. Taylor, General Electric Corporate Research and Development, Schenectady, NY 12345 USA

Dean K. Frederick, Electrical, Computer, and Systems Engg., Rensselaer Polytechnic Institute, Troy NY 12181 USA

Abstract. We discuss an architecture for an expert system to assist a control engineer in coping with the complexity of computer-aided control engineering, with major emphasis placed on implementation. Issues to be treated include: rule base organization, knowledge to be contained in various rule bases, a mechanism to switch between rule bases as the problem solution proceeds, a protocol to coordinate the symbolic computations of the inference engine with the numeric computations of conventional analysis and design software, support of non-monotonic reasoning to permit retracting and revising steps in the design process, and conversion between numeric and symbolic data. We present our approach to these issues, and give examples of rules used to represent operational knowledge and to guide the solution of the problem.

Keywords. Artificial intelligence; expert systems; computer architecture; computer-aided system design; control system synthesis.

1. INTRODUCTION

Control engineering problems and the software tools with which to solve them are becoming ever more complex. The development of more powerful analysis and design procedures together with the rewards for their effective use serve to motivate the creation of these new tools. Unfortunately, as their number, capabilities, and complexity increases, the possibility that any individual will be able to apply the available range of techniques and software effectively is decreasing. We believe that the expert systems approach provides a solution to this difficulty.

The use of expert systems to represent and apply expert knowledge in a particular domain is becoming increasingly widespread (Hayes-Roth, 1983 and Wallich, 1984). Our recent efforts (Taylor, 1983 and 1984 a,b; James, 1985) have focussed on the use of expert systems to aid a control engineer in exploiting available software to achieve an acceptable design. In this research, we use General Electric's DELPHI inference engine in combination with the following conventional analysis and design software tools:

a. the Cambridge Linear Analysis and Design Program (CLADP) (Edmunds, 1979), which performs design, analysis and simulation functions for both single-input, single-output and multiple-input, multiple-output systems in either the time or frequency domain,

b. SIMNON (Elmqvist, 1977), which supports the simulation of nonlinear systems,

c. extensions to SIMNON (Taylor, 1982), which provide equilibrium finding and linearization about the equilibrium point for nonlinear system models, and

d. a routine to categorize eigenvalues, to provide the basis for diagnosing the time-domain characteristics of the system model by the expert system.

In addition, we developed a set of LISP functions which support the various actions described in Sections 2 thru 4 below, and extended CLADP and SIMNON to support the exchange of data between the expert system and the external processes.

Integrating the numerical capabilities of the above software with the knowledge representation and deductive mechanisms of the DELPHI inference engine results in a high-level combination of large-scale numerical processing with large-scale symbolic processing. During the execution of a design sequence, we use the expert system (written in FRANZ LISP and running under the EUNICE operating system) to control the start-up of the various numerical routines (written in FORTRAN and running under the VAX/VMS operating system), direct the transformation and exchange of data required to proceed with the problem, and implement the interface with the user. As many as four external VAX/VMS sub-processes are created, and external programs are started, used, and stopped by the expert system through these subprocesses. To illustrate the scope of this activity, more than a hundred individual commands may be issued by the expert system as control system compensators are designed. Such a software system thus achieves a transfer of much of the burden of the complexity of the control engineering problem from the shoulders of the design engineer to the expert system.

This paper is organized as follows: In Section 2 we discuss a partitioning of an expert's control engineering knowledge into various rule bases and the organization of these rule bases. The implementation of the design process and the role of a switching function in that implementation are discussed in Section 3. The protocol used to coordinate the symbolic manipulations of the inference engine with the numeric computations of the conventional software is treated in Section 4. We conclude in Section 5 with an outline of the key issues and our approach to addressing them, and salient conclusions.

2. KNOWLEDGE PARTITIONING

The rule-base architecture is shown in Fig. 1. The production rules which comprise this structure were developed to realize the concepts presented in the Computer Aided-Control Engineering (CACE) architecture of Taylor (1984b). The present structure differs slightly from what was originally proposed. The principal difference is that the design process

has been implemented in one rule base rather than two, to eliminate unnecessary rule-base switching.

As the number of design procedures grows and the complexity of the other portions of the expert system increase, partitioning the knowledge into modules will be an essential requirement to reduce the time required to execute various phases of the analysis and design process. In addition, modularity will make the expert system easier to develop, modify, and maintain.

The six operational rule bases of Fig. 1 are integrated through a set of supervisory rules (meta-rules) which checks the current state of the problem solution and decides what needs to be done next. This implementation of the expert system architecture was also suggested in (Taylor, 1984b). Invoking the next applicable rule base is accomplished by use of a switching function which is further discussed in Section 3.3.

The knowledge contained in the rule bases (Fig. 1) is as follows:

SUPERVISOR: The function of this rule base is to permit the user to select a training or design session, and to direct its execution, Fig. 2. This selection allows any of the sets of rules to be invoked individually (the TRAINING option), or aids the user in carrying out the overall process (the DESIGN option). The training option makes use of archival lists of facts associated with the design of lead-lag precompensators for a third-order, type-one plant, to allow the user to explore the workings of each individual rule base. Specific functions are: starting and stopping the session; permitting the user to select a training or design session; selecting the appropriate rule base to continue the session when one set of rules has completed its work (during a DESIGN session); saving the current list of facts if the user desires to interrupt the session and continue at a later time; and permitting the user to revise the problem formulation as the result of further analysis and design. (This is an example of non-monotonic reasoning. That is, we are able to "revise a belief" that specification development has been completed, and return to update the specifications.)

MODEL: This rule base will either aid the user in building a CLADP description of a linear model, or request the user to enter the file name of a previously-developed CLADP model. Similar ability to define a nonlinear SIMNON model followed by equilibrium finding and determination of a linearized model is also supported.

DIAGNOSE: The function of this rule base is to analyze and characterize the model of the plant. Currently s-domain information is provided on: the stability of the open-loop system; whether the system is type-zero, one or two; whether the plant can be approximated as an overdamped or underdamped second-order system; and whether the plant has a dominant pole or pole pair. Also, the following frequency-domain information is provided: open-loop bandwidth (type-zero system); uncompensated gain and phase margin; low-frequency gain (type-zero system) or velocity constant (type-one system); closed-loop bandwidth with unity feedback; closed-loop bandwidth with 15 dB of gain margin; approximate bandwidth achievable with two leads; and approximate low-frequency gain or velocity constant achievable with two lags.

CONSTRAIN: The function of this rule base is to enable the user to enter constraints. These rules currently support frequency-domain parametric constraints on the compensator dc gain, phase lead added by the leads, or low-frequency gain added by the lags.

DESIGN: This rule base aids the user in applying CLADP procedures to design lead-lag precompensation for a single-input, single-output linear plant. A detailed description of the heuristic used to automatically design the compensator is given in (James, 1985). A synopsis of the heuristic is as follows:

a. Add lead precompensators to adjust the open-loop phase angle at the desired closed-loop bandwidth. This places the high-frequency portion of the Nichols locus in approximately the right region.

b. Add or adjust a constant-gain precompensator to meet the gain margin.

c. Adjust the actual closed-loop bandwidth by making incremental changes in the lead compensator pole/zero ratios.

d. Repeat steps b. and c. until the gain margin and bandwidth specifications are met. When both conditions are satisfied, the high frequency portion of the Nichols locus is correct.

e. Add lag precompensation to adjust the low frequency gain (for a type-zero system) or velocity constant (for a type-one system). The lag(s) are placed so the high frequency portion of the Nichols locus is not disturbed significantly.

SIMULATE: This rule base facilitates the simulation of the closed-loop system. CLADP is used to provide the step response of the system with a linear plant model, and SIMNON is used to perform more general and realistic simulations of the response of the system with linear or nonlinear plant models.

3. IMPLEMENTING A STRUCTURED PROCESS

The user directs the design process of the expert system by providing names of data files, menu selections, and responses to queries. While the user is in charge of the problem solution, the expert system implements a specific train of thought to aid in achieving the solution (e.g., it checks that reasonable specifications are entered before starting the design of a compensator).

The knowledge summarized in Section 2 is needed to implement such a train of thought. This knowledge is symbolically represented in a partitioned sets of rules and a list of facts. A discussion of the syntax of production rules and lists of facts and how they can be used to represent knowledge can be found in (Taylor, 1984b) or (Johnson, 1983).

The expert system carries out the design process by applying the appropriate rule base to the problem. This is achieved by recognizing when the current rule base has completed its tasks, switching to the supervisory rule base, and determining which rule base should be invoked next. Implementing the transition from one rule base to another in DELPHI involves a careful hand-over of the current list of facts from one rule base to the next. A LISP switching function has been written to perform these actions.

To convey a clearer understanding of how we have controlled the logical flow associated with the design process, we will first discuss forward chaining through a rule base, then discuss backward chaining, and then present some lessons learned about switching between rule bases.

3.1 Forward Chaining

Most frequently, the inference engine uses the list of facts to determine the current situation and then take the appropriate action(s) to proceed with the next step in the process. This inference mechanism is called forward chaining. For example, the rule which recognizes that the lead-lag design criteria

have been met and writes the facts that will display summary information to the user, direct the expert system to provide a Nyquist plot of the compensated system, and then return to the supervisory rule base, is as follows:

```
(Rule_130 (" design for GM, lfg, clbw is done ")
 (WHEN
  (TRUE (SYSTEM TYPE . ZERO)
   " Have a type-zero system ")
  (TRUE (PHASE-AT-OMEGA-BW VALUE . ADJUSTED)
   " Have adjusted the phase at omega bw ")
  (TRUE (EXCESS-GAIN-MARGIN VALUE . -3<..<+3DB)
   " Excess gain margin is ok ")
  (TRUE (LOW FREQUENCY . ADJUSTED)
   " Low frequency gain is ok ")
  (TRUE (CLOSED-LOOP-BW VALUE . ADJUSTED)
   " Closed-loop bandwidth is ok ")
  (TRUE (GAIN-MARGIN FINAL-VALUE . (? GM))
   " The final gain margin is known ")
  (TRUE (CLBW FINAL-VALUE . (? CLBW))
   " The final closed-loop bandwidth is known ")
  (TRUE (LFG FINAL-VALUE . (? LFG))
   " The final low-frequency gain is known ")
  (TRUE (DESIGN-FACTS VALUES . ALL-ENTERED)
   " We have entered all required facts "))
 (THEN
  (CLEAR (DESIGN-FACTS . ALL-ENTERED)
   " Prevent looping on this rule ")
  (SCREEN (?? ?? . ??)
   " Clear the terminal display for the user ")
  (DISPLAY (?? ?? . ??)
   "  "
   " LEAD-LAG design specifications are met: "
   "  "
   " gain margin        = ?GM  dB "
   " bandwidth          = ?CLBW  rps "
   " low-frequency gain  = ?LFG  dB ")
  (ACKNOWLEDGE (L-L-DESIGN CONDNS-MET . ACKNLD)
   " Wait for the user to acknowledge info ")
  (WRITE (PROVIDE FINAL . NYQUIST-PLOT)
   " Fire the rule to provide a Nyquist plot ")
  (WRITE (SYSTEM-SESSION REQD . FROM-DESIGNLL)
   " Move to the Supervisory rule base ")))
```

The nine conditions in the premise of the above rule (i.e., those elements between WHEN and THEN) are used to recognize that the lead-lag design has been successfully completed for a type-zero system. Comments associated with each fact are enclosed in double quotes. The six statements in the conclusion (i.e., those elements following THEN) perform those actions needed to continue with the design process or are used to fire other rules which will perform such actions. Some quoted material (e.g., "Prevent looping on this rule") comprise comments, while some phrases are displayed to the user (e.g., "LEAD-LAG design specifications are met:"); this is governed by the verb (CLEAR or DISPLAY, respectively). Symbols are assigned values by the inference engine (e.g., (? GM) may have the value 19.5 dB), and those values are also displayed to the user (e.g., line 5 of the DISPLAY action above results in the message

gain margin = 19.5 dB

being issued). The forward chainer starts work as soon as a rule has its premise satisfied, and continues until no rule has its premise satisfied.

3.2 Backward Chaining

A rule can also recognize that the current situation calls for establishing a goal to be achieved by the inference engine. This inference mechanism is called backward chaining. For example, the rule which repeatedly checks for the completion of the major steps in the design process, and thus is the heart of Fig. 1, does so by invoking the backward chainer to verify a list of facts:

```
(Rule_300 (" Design process being checked ")
 (WHEN
  (TRUE (SYSTEM SESSION . BEGUN)
   " You have selected a DESIGN session "))
 (THEN
  (CLEAR (SYSTEM SESSION . BEGUN)
   " Prevent looping on this rule ")
  (PROVE (MODELING ASSISTANCE . PROVIDED)
   " Has the system model been entered? ")
  (PROVE (DIAGNOSIS ASSISTANCE . PROVIDED)
   " Has the system been diagnosed? ")
  (PROVE (CONSTRAINT ASSISTANCE . PROVIDED)
   " Are system constraints entered? ")
  (PROVE (SPECIFICATION ASSISTANCE . PROVIDED)
   " Are system specifications entered? ")
  (PROVE (DESIGN ASSISTANCE . PROVIDED)
   " Has the design met the specs? ")
  (PROVE (SIMULATION ASSISTANCE . PROVIDED)
   " Is simulation of the CL system done? ")
  (ACKNOWLEDGE (ADVISE DESIGN-VERIF . DONE)
   "  "
   " Control system design and verification
   " is finished. You may wish to alter the
   " given model, constraints, and/or specs
   " and redo the design, or repeat another
   " portion of the process, or exit. ")
  (WRITE (DESIGN-SESSION MENU . REQUESTED)
   " Allow user to modify or quit. ")))
```

This rule is fired each time the supervisory rule base is loaded. The backward chainer considers each PROVE statement in turn. If it is already established in the list of facts, the backward chainer moves on to the next statement; the first hypothesis that has not already been established will be picked by the inference engine for testing. The supervisory rules are formulated so that this results in invoking the appropriate operational rule base. The inference engine will proceed in this way until the entire set of hypotheses has been satisfied and therefore the goal can be said to be achieved.

Generally, this rule results in the progression MODEL, DIAGNOSE, CONSTRAIN, SPECIFY, DESIGN, SIMULATE. However, this is not a rigid regimen: if during the design process it is determined that a specification cannot be met and should be changed, the fact (SPECIFICATION ASSISTANCE . PROVIDED) can be cleared by a rule in the design rule base. Other facts can be written to indicate which specification needs attention. When the supervisory rule base tries to verify that (SPECIFICATION ASSISTANCE . PROVIDED) is a fact, it will no longer be in the list of facts. Thus, the inference engine will again invoke the the specification rule base and work to validate this fact.

Comparing the premise of this rule to that of the example used for forward chaining, we see that only a single condition is needed to recognize that the steps in the design sequence need to be verified. The PROVE verb is used to indicate to the backward chainer that the associated fact (e.g., (MODELING ASSISTANCE . PROVIDED)) is a hypothesis to be verified. The backward chainer will then check to see if the fact is already known to be true or false. If the fact is not known, then the backward chainer will proceed to look for rules which will write this fact in their conclusions and work to establish the premise of those rules. This involves switching in the modeling rule base and proceeding until (MODELING ASSISTANCE . PROVIDED) is true.

3.3 Using a Rule-Base Switching Function

Rules in each operational rule base recognize that it is time to return to the supervisory rule base to proceed with the overall problem. The facts are then saved in a file, the supervisory rule base is loaded as the current rule base, and the facts are made available to the supervisor. The next applicable rule base is selected by the supervisory set of rules. The

six operational rule bases are contained in separate files which are invoked using the switching function. This process continues until the expert system can do no more or the user elects to end the session. As we have seen above, Rule_300 performs this top-level control of the process.

This approach is useful if the problem being addressed can be partitioned into "phases" with well-defined initial and final conditions, and if the phases are of sufficient complexity that the overhead involved in switching is warranted. The knowledge partitioning discussed in Section 2 satisfies these conditions.

The exit criterion for a rule base must not be satisfied until all required actions have been completed. This condition is met when the pattern which fires the rule in which the switch function is called occurs only for that rule. We normally achieve this by executing a switch as the last action taken by the first rule in a rule base.

The list of facts is passed as a queue from one set of rules to the next, and DELPHI starts to check rules for firing as facts are being loaded. In this case, one must be careful not to start any actions in a new rule base until the entire list of facts has been entered. (Otherwise some facts may be lost if a subsequent switch occurs before all of the old facts are entered, or the new set of rules will not behave properly since all of the facts would not be available.) Therefore, each rule base is designed such that its start-up rule cannot be fired until the last fact written by the prior rule base is loaded.

The seven rule bases now total approximately 300 rules; using a command file to run through a typical design sequence takes about thirty minutes on an unloaded VAX 11/785. About half of this time is required to compile the rules as we switch between operations. This time would be reduced significantly if the DELPHI inference engine supported accessing a compiled version of the rules.

4. PROTOCOL FOR COORDINATING SYMBOLIC AND NUMERIC COMPUTATIONS

The operation of the expert system in concert with conventional analysis and design software requires that:

a. VAX/VMS sub-processes be created to start, run and stop external programs which are sources of data for the expert system,

b. a two-way transformation of data be set up between the symbolic representation of the expert system and the numeric representation of these external programs, and

c. coordination of the operation of the expert system with external programs be established to prevent the expert system and the external programs clashing (e.g., trying to provide information to the user at the same time, or returning to symbolic manipulation before numeric data has been completely supplied).

The DELPHI inference engine has a LISP function available to create a VAX sub-process which uses mailboxes for the input and output ports of the sub-process. Read and write functions are also available to communicate with the sub-process via the mailboxes. We use the mailboxes to start up programs (Fig. 3), and then transmit commands to them and receive data from them using files.

We have implemented a protocol which coordinates the expert system with external programs and provides support for two-way communication via files. Fig. 4 is a timing diagram for running CLADP which depicts the actions taken to implement the protocol. A detailed discussion of this timing diagram follows:

a. The user starts the expert system (point A on Fig. 4) and directs its activity until data is needed from an external program. The expert system then writes the required CLADP commands to the file EXPOUT (points B,C), creates a VAX/VMS sub-process, starts CLADP through the sub-process, and begins to wait (point D) until it detects that CLADP has created the handshake file (point J).

b. Once CLADP is started, it reads the sequence of commands from the file EXPOUT (points E,F), performs the required calculations, writes the results to the file EXPIN (points G,H), creates the handshake file (point I), and begins to wait (point J) until it detects the deletion of the handshake file (point P).

c. When the presence of the handshake file is detected (point J), the expert system reads the symbolic data written to the file EXPIN by CLADP (e.g., (GAIN-MARGIN DB-VALUE . 8.5), points K,L), and continues with the inference process until additional data is needed. At this time, the sequence repeats as described above (points M thru Q).

This sequence of writing commands to EXPOUT and symbolic information to EXPIN while coordinating the operation based on the existence of a handshake file continues until CLADP is no longer needed. At that time the expert system can stop CLADP and the VAX/VMS sub-process is then available to run another program.

Coordinating several processes is identical to this outline; for each process there is an associated file set for handshake and input and output data transfer. Symbol/numeric transformations are required throughout this process: The expert system must know how to write CLADP and SIMNON commands in order to achieve the required results, and CLADP and SIMNON were extended to convert numerical data back into symbolic form (e.g., (GAIN-MARGIN DB-VALUE . 8.5) has to be written into the file EXPIN in the above example).

5. CONCLUSION

The application of the expert systems approach has proven to be useful in coping with the complexity of computer-aided control engineering (CACE) procedures. By reducing the burden on the design engineer to recall the details of using software packages effectively, and by containing the heuristic procedures involved in various analysis and design approaches, an expert system can be an effective aid in the CACE process.

Partitioning the CACE problem into distinct well-defined subprocedures and corresponding rule bases is one key to addressing the complexity issue. This can be achieved by "modeling" the computer-aided control engineering effort along functional lines (Taylor, 1984a,b). In implementing this feature, we have found that switching between rule bases has several benefits, namely:

a. Top-level goals can be reevaluated continually by the supervisor, based on the results achieved by the different operational rule bases. For instance, if work reveals that a bandwidth specification cannot be met under the given conditions, the supervisor may re-invoke the SPECIFY rule base and ask the user to reduce the bandwidth specification.

b. Incremental design is supported, since the rule bases are functionally arranged and the list of facts can be saved to initiate another session. For example, the user can complete a model and diagnosis session, save the facts, and restart the expert system at a later time. Several specification sets and solutions can be studied, starting from the same initial state.

c. Rule bases can be written and debugged separately, and then integrated into the overall structure. The time re-

quired to make trial runs is greatly reduced, and errors can be identified more quickly.

d. The supervisor is used to limit the scope of search, so the running time of the complete expert system should be kept to reasonable bounds even as the number of procedures supported by the expert system increases.

Using the switching technique to implement the rule base partitioning requires careful attention to detail, as indicated in Section 3.3.

We have presented architectural and implementation issues involved in developing an expert system to create a higher-level, flexible environment to help the control engineer solve CACE problems. However, it should be emphasized that the development of an expert system is not a panacea. The creation of an expert system has its own set of difficulties, the most burdensome being the creation of the architectural implementation and knowledge base. The process of translating human expertise into a rule base (knowledge acquisition) is not well understood (Hayes-Roth, 1983). However, we believe that "modeling" the computer-aided control engineering effort along functional lines (Taylor, 1984a,b) is a necessary first step in knowledge representation and acquisition that greatly alleviates this problem. The reduction in the difficulty of the problem presented to the user justifies the increased complexity of the software tool which implements the design environment.

ACKNOWLEDGEMENT

The research reported in this paper has been substantially aided by the contributions of others. Dr. Piero Bonissone has assisted from the beginning of the project, providing the initial inference engine and his experience with expert systems development. More recently, Dr. Melvin Simmons, Dr. Dale Gaucus, and Steven Kirk assisted in implementing the expert system using the DELPHI inference engine. Also, David Kassover has helped in making modifications to CLADP and SIMNON. These individuals were employed by General Electric Corporate Research and Development (David Kassover as a consultant). Their support is gratefully acknowledged.

REFERENCES

Edmunds, J. M. (1979). Cambridge linear analysis and design program. *IFAC Symposium on Computer Aided Design of Control Systems*, Zurich.

Elmqvist, H. (1977). SIMNON - An interactive simulation program for nonlinear Systems. *Proceedings of Simulation 77*, Montreux.

Hayes-Roth, F., D. A. Waterman and D. B. Lenat (1983). *Building Expert Systems.* Addison-Wesley, Reading, MA.

James, J. R., D. K. Frederick and J. H. Taylor (1985). On the application of expert systems programming techniques to the design of lead-lag precompensators. *Proceedings of the Control 85 Conference*, Cambridge, UK.

Johnson, H. E. and P. P. Bonissone (1983). Expert system for diesel electric locomotive repair. *Journal of FORTH Applications and Research, 1*, No. 1.

Taylor, J. H. (1982). Environment and methods for computer-aided control systems design for nonlinear plants. *Proceedings of the Second IFAC Symposium: CAD of Multivariable Technological Systems*, West Lafayette, IN. 361-367.

Taylor, J. H., D. K. Frederick and A. G. J. MacFarlane (1983). A second- generation software plan for CACSD. *Abstracts of the IEEE Control System Society Symposium on CACSD*, Cambridge, MA.

Taylor, J. H., D. K. Frederick and J. R. James (1984). An expert system scenerio for computer-aided control engineering, *Proceedings of the American Control Conference*, San Diego, CA. 120-128.

Taylor, J. H. and D. K. Frederick (1984). An expert system architecture for computer-aided control engineering. *IEEE Proceedings, 72*, 1795-1805.

Wallich, P. (1984). Technology '84 software. *IEEE Spectrum, 21*, No. 1, 47-49.

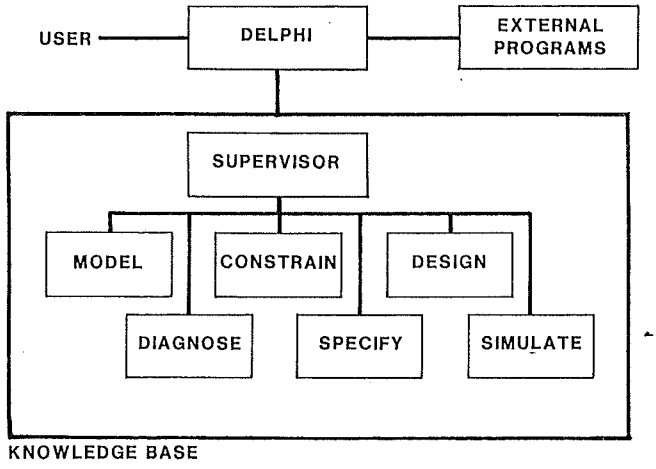


Fig. 1. Expert system structure

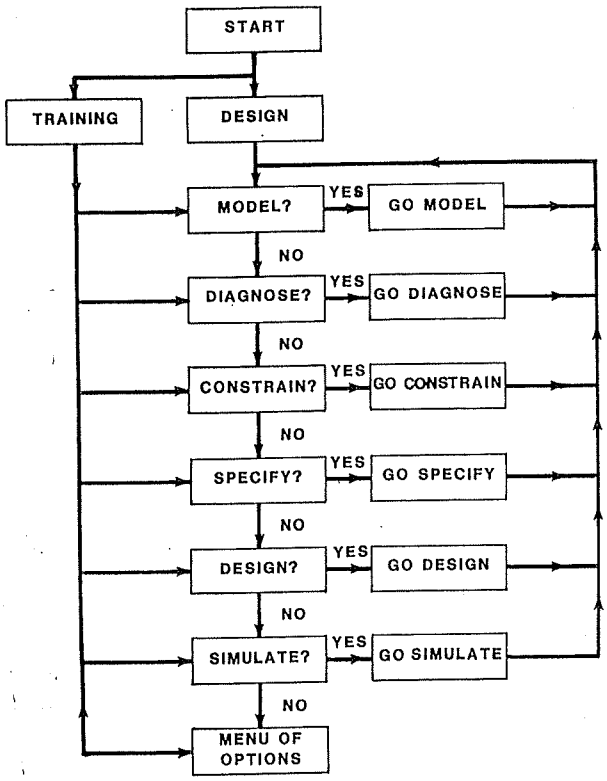


Fig. 2. Flow chart for the supervisor set of rules

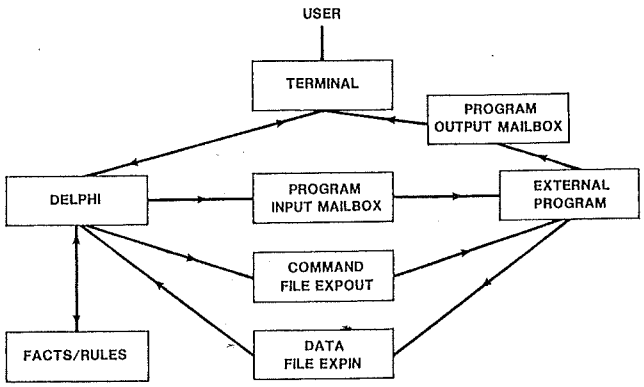
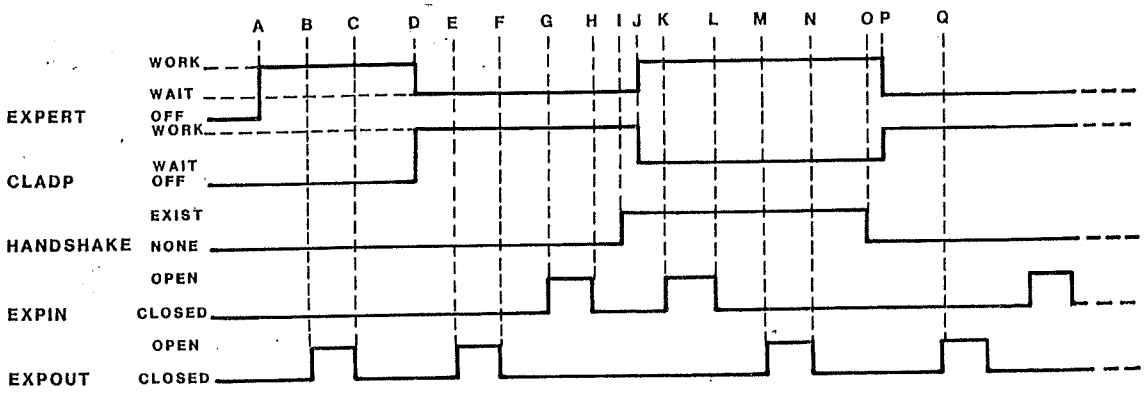


Fig. 3. Running external programs



A - USER BEGINS EXPERT
B - EXPERT OPENS EXPOUT AND WRITES COMMANDS
C - EXPERT CLOSES EXPOUT
D - EXPERT BEGINS CLADP VIA MAILBOX AND BEGINS TO WAIT
E - CLADP OPENS EXPOUT AND READS COMMANDS
F - CLADP CLOSES EXPOUT
G - CLADP OPENS EXPIN AND WRITES RESULTS
H - CLADP CLOSES EXPIN
I - CLADP CREATES AND CLOSES HANDSHAKE
J - EXPERT SYSTEMS RETURNS TO ACTIVE STATE, CLADP BEGINS TO WAIT
K - EXPERT OPENS AND READS EXPIN
L - EXPERT CLOSES EXPIN
M - EXPERT OPENS EXPOUT AND WRITES COMMANDS
N - EXPERT CLOSES EXPOUT
O - EXPERT DELETES HANDSHAKE
P - CLADP RESUMES WORK, EXPERT WAITS
Q - CLADP OPENS EXPOUT AND READS COMMANDS

Fig. 4. Protocol timing diagram