Second Conference on Artificial Intelligence Applications, Miami Beach, Florida, December 1985. © IEEE

# A Retrospective View of CACE-III: Considerations in Coordinating Symbolic and Numeric Computation in a Rule-Based Expert System

John R. James*
Piero P. Bonissone**
Dean K. Frederick†
James H. Taylor**

* Department of Electrical Engineering, USMA, West Point, NY 10996
** General Electric CRD, Schenectady, New York 12301
† ECSE Department, Rensselaer Polytechnic Institute, Troy, NY 12181

The experience derived from designing and implementing CACE-III (Computer-Aided Control Engineering third-generation environment) is described. CACE-III is a knowledge-based system that requires the coordination of symbolic and numeric computation. Its purpose is to aid a user in applying available analysis and synthesis software to complete the steps in the control system engineering process. Help is provided in modeling, constraining, diagnosing, specifying, designing and simulating dynamical systems. The description of CACE-III will cover the following topics: partitioning the knowledge corresponding to different stages in the design process, implementing a lead-lag compensator design heuristic, developing a limited form of non-monotonic reasoning to allow for iterations and trade-offs during the design process, translating numerical data into symbolic form, generating a sequence of commands for the numerical routines, and coordinating the numerical computation of the routines with the symbolic computation of the inference engine.

## 1. INTRODUCTION

The field of control systems engineering has experienced a rapid increase in the number of design methods available to a control engineer.[1] Associated with these methods are numerous software packages containing numerical routines useful in applying the various design methods.[8] Our recent efforts[9,10,13,14,15] have led to the construction of CACE-III (a third-generation Computer-Aided Control Engineering environment). CACE-III is a rule-based expert system designed to aid a control engineer in exploiting available software to achieve an acceptable design. In building the expert system, we first used General Electric's DELTA inference engine[2,3] and then transitioned to DELPHI, a proprietary rule-based system shell developed at General Electric from DELTA's original design. While it is still a research tool, CACE-III, with its 300+ rules, has demonstrated a high level of competence in automatically designing lead-lag compensation for a single-input, single-output linear plant.[9]

During the execution of a design sequence, CACE-III (written in FRANZLISP and running under the EUNICE operating system) controls the start-up of the various numerical routines (written in FORTRAN and running under the VMS operating system), and directs the exchange of data required to continue with the problem and implement the interface with the user. As many as four external VAX/VMS subprocesses are created and external programs are started, used, and stopped by the expert system through these subprocesses. More than a hundred individual commands may be automatically given to the numerical routines by the expert system as the compensators for the system are designed. This feature shifts a large amount of complexity of the design problem from the design engineer to the knowledge sources of the expert system. Thus, the added complexity of writing the expert system software is compensated by the reduction in the complexity of the problem presented to the user.

In this paper, we share some of the lessons learned from the past two year's effort spent in achieving an initial level of capabilities. In Section 2, we discuss the development of partitioning the knowledge into separate files (KBs). In Section 3, we discuss the development of a limited form of non-monotonic reasoning to support making changes during the design process. In Section 4, we discuss the translation of numerical data into symbolic form, the generation of a sequence of commands for the numerical routines, and the coordination of the numerical and symbolic computation. We conclude the paper in Section 5 with comments regarding the significance, status, and expected direction of the effort.

## 2. PARTITIONING OF THE KNOWLEDGE IN CACE-III

A discussion of the initial concept for the architecture of the expert system is given.[14,15] This concept featured two data structures to represent the status of the design process (a *problem frame* and a *solution frame*) and six rule bases to govern interaction with the user, updating of the problem and solution frames, and use of various analysis and design routines (Figure 1). Two of the six rule bases (RB 1, RB 2) were to interact with the user to develop the system model, constraints, and specifications. Three rule bases were envisioned as aiding the user in the design process: RB 3 by initializing the solution frame (at the start of design or during iterations or trade-off studies), RB 4 by automatically selecting and executing an appropriate design method, and RB 5 by updating the solution frame. The final rule base (RB 6) was to validate and implement the design. Continuous interaction was expected between the user and RBs 1 and 2 as the design problem was established. Less interaction with the user was expected during the actual design process (mainly, design iteration and trade-off).

### 2.1 DELTA Inference Engine

The initial implementation of a subset of these ideas was based on the DELTA inference engine.[2,3] Figure 2 illustrates the architecture of the inference engine.

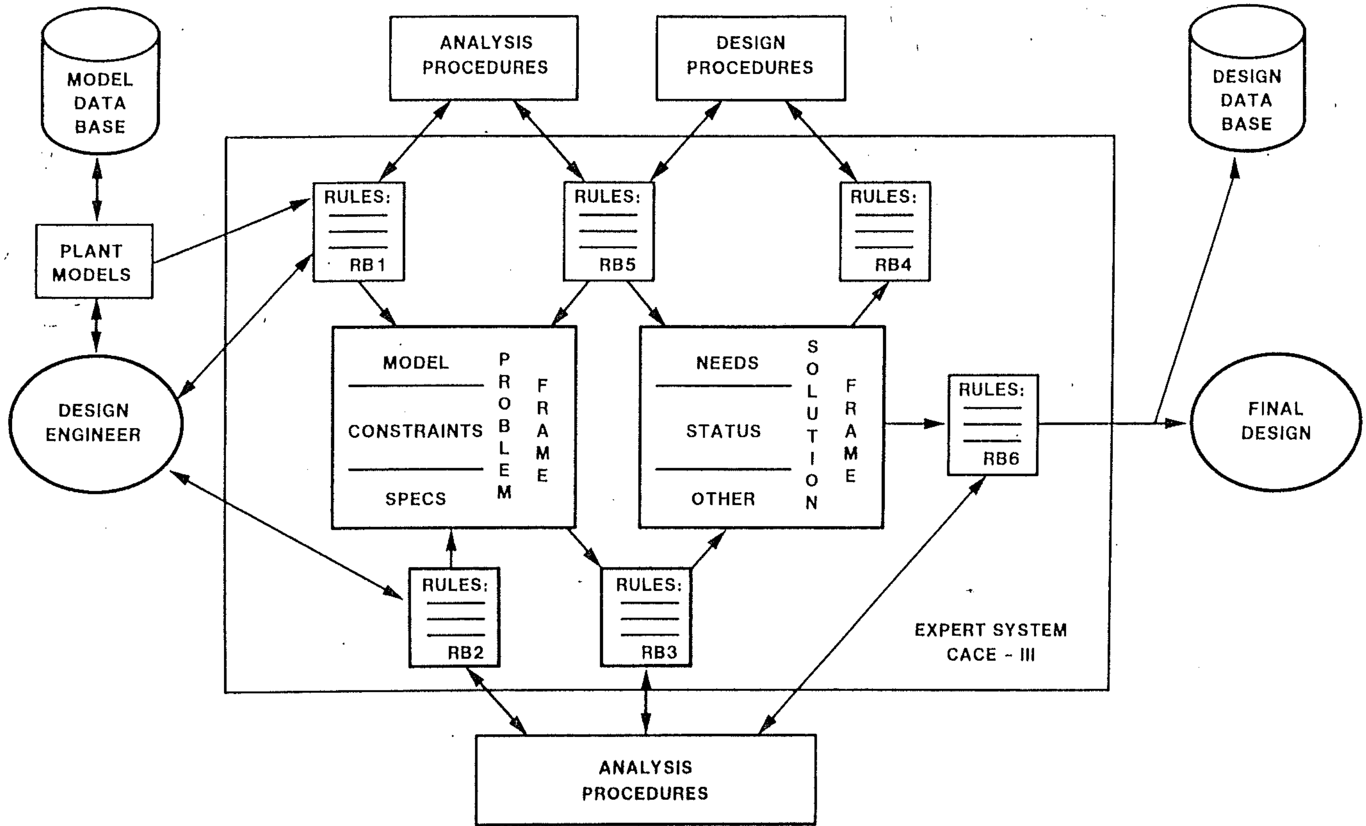


**Figure 1. Original CACE-III Functional Structure**

DELTA inference engine consists of a *backward* and a *forward* rule interpreter. The backward interpreter is activated whenever a new hypothesis set is loaded into the *hypothesis stack* (dotted box in Figure 2). This may be caused by an external invocation of the backward interpreter, e.g., at the beginning of a session, by a recursive call from the backward interpreter itself, during the evaluation of a sub-goal, or by executing the PROVE verb in the right-hand side of a *meta-rule* being fired by the forward interpreter. The rule search order used by the backward interpreter defaults to the classic depth-first strategy.

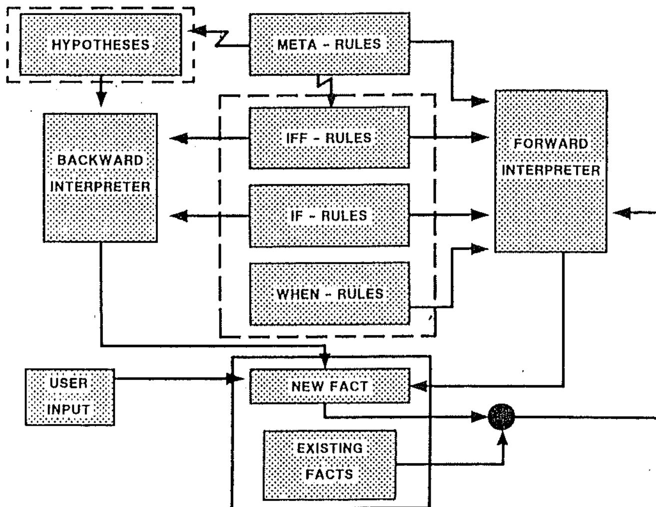


**Figure 2. DELTA and DELPHI Inference Engine Configuration**

The forward interpreter is activated only when a *new fact** is written into the fact set. The new fact is then used by an efficient indexing scheme to retrieve and analyze only those rules that might have been affected by the new fact. If any of these rules has a completely satisfied premise, the forward interpreter will execute its right-hand side, will cause more new facts to be written and will iterate again. This forward-chaining process will stop when no executable rule can be found, and control will be returned to the backward interpreter. The backward interpreter will continue its deductive process, until a hypothesis is proven or the entire hypothesis set has been exhaustively evaluated.

The DELTA inference engine and representation language capabilities were sufficient to implement the first version of CACE-III and to support a limited form of non-monotonic reasoning that will be discussed in Section 3. DELTA was originally designed to be the context independent shell on which a locomotive troubleshooting expert system was successfully implemented.[2,3] The representation language in DELTA provided the knowledge engineer with nine predicate functions, eight verbs, and five utility functions with which conditions, actions and help routines could be described. However, given its original goals, the representation language in DELTA did not provide numerical functions, access to external programs, or the representation expressiveness of predicate calculus. These capabilities were needed to automatically select and run external programs, and to interpret their results. The initial implementation using the capabilities present in DELTA supported the concept of separate rule bases, coexisting in the same file, for modeling, diagnosing constraining, analyzing, specifying,

* A *new fact* is a 3-tuple that did not previously exist in the fact set, or whose certainty measure had been improved.

designing, and simulating a system. The first rule bases to be written were the ones for *specification* and *design.* Provisions were made for expanding the other rule bases and integrating them with the first two. The two rule bases were present in one file with several top-level rules (meta-rules) for determining which rule base would be used.

The DELTA-based implementation of CACE-III differed from the the originally designed system in two aspects. First, a step or stage in the design was completed by a single rule base, rather than having multiple rule bases cooperate to complete such a step. Secondly, a single set of facts reflected the entire status of the problem rather than having two data structures (the *problem frame* and the *solution frame*).

2.2 Transition from DELTA to DELPHI

Reducing the scope of the inference engine search through the sets of rules was a primary consideration in developing CACE-III. Early in the construction of the system we were concerned about the eventual number of rules required to capture the expertise of the design process. The initial structure implemented achieved a partitioning of the knowledge into separate rule bases with selection of the applicable rule base governed by top-level rules. However, all the rule bases were in one file, leading to inefficiency, since rules applicable at only one step in the design process would be checked by the inference engine during the entire process. With the availability of DELPHI, the new inference engine developed to be rule upper-compatible with DELTA, we decided to distribute the knowledge into separate files that could be loaded at different times.[10] The concept of the design process as a series of steps or stages supported this attempt since the knowledge could be separated by stages. In addition, separate rule bases could be developed corresponding to the many different design approaches. A *switching function* was added to DELPHI to support moving to the appropriate knowledge source. Furthermore, DELPHI provided the needed representation expressiveness and the numerical capabilities to interpret the results generated by external programs.

The structure currently implemented[10] differs from the initial implementation in the elimination of a separate rule base for system analysis, in the writing of the remaining rule bases, in the expansion of the few top-level rules into a supervisory rule base, and in the implementation of the rule bases in separate files. Figure 3 illustrates the current architecture of the CACE-III system.

2.3 Implementation Remarks

From this first experience in implementing CACE-III, we could make the following three observations. First, by modifying the architecture of the system (changing from the original architecture depicted in Figure 1 to the new one shown in Figure 3), we reduced the rule interaction problem (i.e., we eliminated the many rule-base switchings that the RB 4/RB 5 partitioning would have involved), which had a major impact in the efficiency of the system. Second, we observed that by separating the knowledge sources into individual files, we achieved not only the efficiency and conceptual structural advantages originally anticipated,[15] but also several beneficial side effects, such as:

1. Rule bases can be written and debugged separately and then integrated into the overall structure. Errors are quickly identified since, at any point in the execution, the number of possible sources of error is reduced. Also, execution of each rule base in isolation during development greatly reduced the run-time.
2. The functional arrangement of the rule bases allows the user to incrementally design the closed-loop control system. For example, the user can complete a model, save the facts, and resume the design process at a later time.
3. Since the supervisor is used to reduce the scope of the search, the running time of the overall system is minimized as the number of different procedures supported by the expert system increases.
4. A goal previously satisfied in a knowledge base can be modified by a different rule base when additional information suggests the need of re-evaluating the goal. For instance, if the design rule base determines that a bandwidth specification cannot be met under the constraint that no more than two leads can be used in the design, we can no longer support the belief that the design specifications can be satisfied. By clearing that assertion, the supervisory rule set will return to the specification rule set and prompt the user to reduce the bandwidth specification. This form of non-monotonic reasoning is further discussed in Section 3.

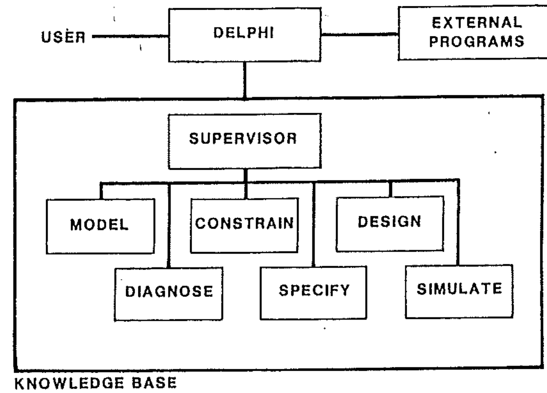


**Figure 3. Current CACE-III Functional Structure**

Finally, the hierarchical system, implemented using a supervisor and partitioned rule bases, can be recursively extended to lower levels using sub-supervisors within the rule bases.

## 3. NON-MONOTONIC REASONING IN CACE-III

### 3.1 Implementation of Limited Non-Monotonic Reasoning

The initial implementation of CACE-III contained two forms of limited non-monotonic reasoning. These two forms enabled the user to change the response to a query, and supported iteration through a design sequence, to allow developing alternative designs.

The above forms depended upon a feature of the DELTA inference engine that allows a rule to be refired if its premise is completely satisfied by the set of facts, and *if at least one of these facts just had its certainty value modified.* By writing rules describing existing data dependencies, it was possible, using the forward chainer, to *clear* the dependent values when any of the independent supporting values were modified. Partitioning the rule base reduced the number of active depen-

dencies and thus reduced the required *reason-maintenance rules.*

To enable the user to change his response to a query, a logical loop was created by allowing an option in a menu to be the return to an earlier option. The undesired response was first cleared and then the fact required to return to the earlier query was written.

A modification of this technique was used to iterate through the design process. The objective was to use a set of rules to implement a heuristic approach for choosing a successively *better* set of design parameters until an acceptable design was achieved. The design method[9] was captured in a set of rules that, using the forward chaining mode, created a series of loops that converged to an acceptable design. In each case, a limited number of the facts to be altered (number of lead compensators, value of the gain of the lead compensator(s), center frequency of the lead(s), and the status of the gain compensator) were updated and the results of altering the parameters were obtained. A small set of rules was used to detect the occurrence of an acceptable result and to exit from the loop by writing an appropriate fact. Table 1 shows an example of such a rule.

Table 1

LEAD-LAG DESIGN COMPLETION RULE

```
(Rule_130
 ("design for frequency-domain specs 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-G-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 VALUES . ALL-ENTERED)
      " Prevent looping on this rule ")
    (SCREEN (?? ?? . ??)
      "Clear the terminal display for the user ")
    (DISPLAY (?? ?? . ??)
      " "
      " LEAD-LAG design conditions are met, namely: "
      " "
      " gain margin        = ?GM  db "
      " bandwidth          = ?CLBW rps "
      " low-frequency gain  = ?LFG db ")
    (ACKNOWLEDGE (L-L-DESIGN SPECS-MET . ACK)
      " Wait for the user to acknowledge the above info. ")
    (WRITE (PROVIDE FINAL . NYQUIST-PLOT)
      " Fire the rule to provide a Nyquist plot ")
    (WRITE (SYSTEM-SESSION REQ . FROM-DESIGNLL)
      " Support the move to the Supervisory set of rules ")))
```

The nine conditions in the premise of the above rule are used to recognize that the lead-lag design has been successfully completed for a type-zero system. Comments concerning each fact (e.g., (SYSTEM TYPE . ZERO)) are enclosed in double quotes. The six actions in the conclusion manage the dialogue with the user or are used to fire other rules that will execute actions needed to continue with the design process. Logical variables are assigned values by the pattern matcher of the inference engine and those values are displayed to the user (e.g., gain margin = 19.5 db).

The current implementation of CACE-III supports a third application of limited non-monotonic reasoning: subsequent revision of choices made during the specification or constraints stages of the design process as the result of problems encountered during the design stage. This technique is made possible by the addition of the switching function[10] to the DELPHI inference engine. In this case the design has proceeded to the point where the user specifications cannot be met using the current constraints. The user is returned to the set of rules where the constraints or specifications are entered after the appropriate intermediate facts are cleared.

This process is implemented in the supervisor set of rules (Figure 4) by sequentially invoking the backward chainer to verify a set of facts. A particular rule (*Rule_300*, illustrated in Table 2) is fired each time the supervisory set of rules is loaded. Since the backward chainer will not try to find a rule to validate a fact that is already in the list of facts, the first hypothesis that has not already been established will be picked by the inference engine for testing. Therefore, if during the design sequence of operations it is determined that a specification cannot be met and should be considered for change, the fact (SPECIFICATION ASSISTANCE . PROVIDED) will be cleared by the design rule set and other facts will be written by the same design rule set to indicate the required specification changes. When the supervisory rule set tries to verify the proper completion of the specification step, it will find that (SPECIFICATION ASSISTANCE . PROVIDED) is no longer true. Thus, the inference engine will again access the specification rule base to re-establish this fact.

### 3.2 Remarks on Limited Non-Monotonic Reasoning

A reliable modeling of certain forms of non-monotonic reasoning used by a design engineer was achieved by using the forward rule interpreter and a small set of *reason-maintenance rules.* This method is not as general as an external Reason Maintenance System (RMS) tracking the support of all the assertions.[11] However, the proposed method did not require any additional overhead caused by special RMS algorithms or backtracking mechanisms.

The hierarchical arrangement of the rule sets, as well as the view of the design process as a series of stages, supported the implementation of these limited forms of non-monotonic reasoning. Since only a few facts were created during each stage of the design process, it was possible to track and reset these facts whenever a change in specifications or constraints was required. During this *undo* process, the extent of rule interaction was limited by the functional scope of each rule base, preventing facts created in one rule base from triggering rules in another rule base.

Three types of conditions must have been met to apply the switching function:

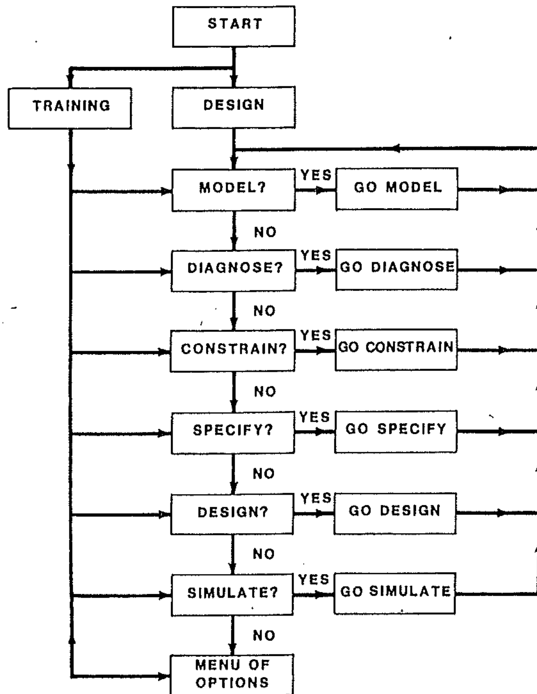


**Figure 4. Flow Chart for the Supervisor Rule Set**

Table 2

Top Level Supervisory Rule

```
(Rule_300 (" The design sequence is being checked ")
 (WHEN
  (TRUE (SYSTEM SESSION . BEGUN)
    " You have selected the controller design session "))
 (THEN
  (CLEAR (SYSTEM SESSION . BEGUN)
    " Prevent looping on this rule ")
  (PROVE (MODELING ASSISTANCE . PROVIDED)
    " Enter the system model ")
  (PROVE (DIAGNOSIS ASSISTANCE . PROVIDED)
    " Diagnose the system ")
  (PROVE (CONSTRAINTS ASSISTANCE . PROVIDED)
    " Enter the system constraints ")
  (PROVE (SPECIFICATIONS ASSISTANCE . PROVIDED)
    " Enter the system specifications ")
  (PROVE (DESIGN ASSISTANCE . PROVIDED)
    " Design to meet the specs ")
  (PROVE (SIMULATION ASSISTANCE . PROVIDED)
    " Simulate the compensated system ")
  (ACKNOWLEDGE (ADVISE DESIGN-VERIF . DONE)
    "    "
    " The controller design and verification is finished. "
    " You may wish to alter the given model, constraints, "
    " and/or specifications and redo the design or repeat "
    " another portion of the process. ")
  (WRITE (DESIGN-SESSION MENU . REQUESTED)
    " Offer the opportunity to make changes ")))
```

1. The problem being addressed must be suitable to be decomposed into stages (or other logically distinct groupings) with *clearly understood results expected at each stage.* Any switching between the rule sets can only occur between these stages.

2. It is important not to satisfy the exit criteria for a rule set until *all* pertinent actions have been completed. This condition is met when the pattern that fires the rule calling the switch function is *unique* for that rule. Proper execution can be achieved by placing a switch function as the *last* action taken by the first rule selected in a rule set.

3. The list of facts is passed as a queue from one rule set to the next. No action should be started in the new rule set until the *entire* list of facts has been entered. Otherwise, some facts may be lost if the next switch occurs before all old facts are entered, or the new rule set will not behave as expected, since all facts would not be available. This condition is met if the new rule set does not have any rule that is fired by any pattern of the current list of facts until the *most recent* fact written by the old rule set has been entered. This can be achieved by carefully naming the facts written in each rule base to ensure that the list of facts that is passed from one rule set to the next will only fire rules within successive knowledge sources *after a key fact (the most recent) has been written.* That is, each rule base is designed so that no rule can be fired until the whole list of facts is loaded and the last fact written by the prior rule base is sufficient to fire the start-up rule in the current rule base.

## 4. INTEGRATING SYMBOLIC AND NUMERIC COMPUTATIONS

### 4.1 Invoking and Interfacing With Numerical Routines

The DELPHI inference engine has a LISP function available to create a VAX sub-process that uses mailboxes for the input and output ports of the sub-process. Read and write functions are also available to communicate with the subprocess via the mailboxes. We use the mailboxes to start up the program (Figure 5), and then transmit commands to the program and receive data from the program using files.

In order to invoke and execute a numerical procedure, a sequence of commands must be generated and passed to the VMS process corresponding to the appropriate numerical program. The DELPHI inference engine has a facility for accessing the LISP *prog* function. In addition to providing low-level numerical capability, this feature is used to access a set of functions written to generate the required sequence of commands. The commands generated by these functions are written in a file which is then read by the numerical routine as a command file.

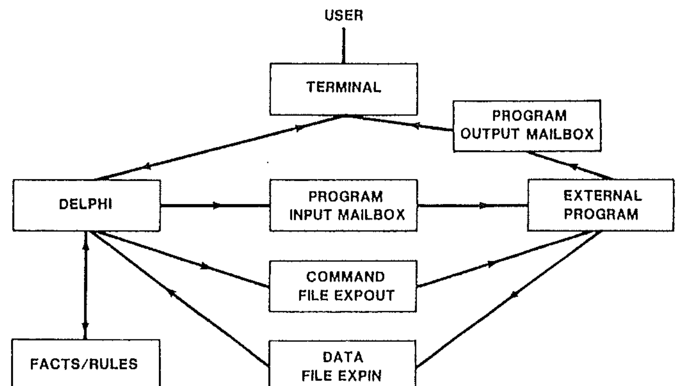


**Figure 5. Running External Programs**

We obtain information *in symbolic form* from the numerical routines. The numerical routines have been altered to cause them to write selected data to files in the 3-tuple format expected by the DELPHI inference engine for a fact. A DELPHI function is then used to write these facts into the set of facts. This data is then used by the inference engine to determine what to do next. If additional numerical processing is required, then this sequence of activity is repeated.

### 4.2 Protocol to Coordinate Symbolic and Numeric Computation

In our implementation, the above procedure demands that:

- VAX/VMS sub-processes be created to start, run and stop CLADP (Cambridge Linear Analysis and Design Program[6]) and other external programs that are sources of data for the expert system.
- A two-way exchange of data be established between the symbolic representation of the expert system and the numeric representation of these external programs.

The data displayed to the user by the expert system and the external programs can be varied from a minimum of menu selection and summaries of numerical results to a complete display of the logical inference steps (to include which rules are being fired and the values assigned to logical variables), the facts being passed from one rule base to the next, and intermediate numerical calculations. The range of information displayed is determined by the user by setting flags.

Coordination of the operation of the expert system with that of the external programs must be established to prevent the expert system and the external program trying to provide information to the user at the same time and to avoid file-accessing contention. We have implemented a protocol that coordinates the expert system with external programs and provides support for two-way communication via files. This protocol[10] uses a handshake file to prevent the interleaving of information as it is displayed to the user and avoids attempting to open and read files before or while they are being written by another process.

## 5. CONCLUSIONS

The development of expert systems programming techniques has resulted in tools that have proven useful in coping with the complexity of the control system design process. By reducing the burden on the design engineer to recall the details of using a particular software package, by verifying the consistencies of the design specifications, and by capturing the heuristics of various analysis and design methods in the form of rule bases, an expert system can be an effective aid in the design process.

CACE-III is the first effort in building an expert system to assist an engineer in the control system design process. The view of the design process as a sequence of stages[15] has supported the partitioning of the knowledge into separate files that are accessed by a switching function under the control of a supervisory set of rules. The need to revise constraints and/or specifications has led to the development of a limited form of non-monotonic reasoning to meet this need. The requirement to automatically access numerical routines has led to the development of a protocol to coordinate numerical and symbolic computation.

We recently expanded the expertise of CACE-III to include the analysis and design of nonlinear systems using the extended SIMNON[7,12] to perform equilibrium finding and linearization as a precursor to design and nonlinear simulation for design validation. Subsequent expansions may include the design of 2-loop digital control systems with spectral separation (fast inner loop, slow outer loop), the design of multiple-input, multiple-output control systems using the CLADP frequency domain approaches,[6] and the design of multiple-input, multiple-output control systems using the LQG/LTR method of Doyle and Stein.[5]

To support these extensions of the problem domain, we may have to extend some of the current capabilities of the underlying DELPHI inference engine. In particular, we may have to upgrade the current numerical representation of uncertainty (described by a scalar value on the interval [-1,1]) by incorporating recently-developed concepts of linguistic likelihood statements and appropriate calculi selection.[4] This feature will provide the user with a *natural* scale of linguistic assessments (uncertainty quantifications) and corresponding combination rules, that will also prove useful in describing subjective assessments of suitability and in solving *trade-off* aggregations typical of the redesign process.

## 6. REFERENCES


[1] J.D. Birdwell et al., "Issues in the Design of a Computer-Aided System and Control Analysis and Design Environment (CASCADE)," ORNL/TM-9038, Oak Ridge, Tennessee, 1984.

[2] P.P. Bonissone, "DELTA: An Expert System to Troubleshoot Diesel Electric Locomotives," *Proceedings of ACM 83*, pp. 44-45, New York, October 24-26, 1983.

[3] P.P. Bonissone and H. Johnson, "Expert System for Diesel Electric Locomotive Repair," *Human Systems Management*, Vol. 4, pp. 255-262, 1984.

[4] P.P. Bonissone and K.S. Decker, "Selecting Uncertainty Calculi and Granularity: An Experiment in Trading-off Precision and Complexity," *Proceedings of the Workshop on Uncertainty and Probability in Artificial Intelligence*, University of California, Los Angeles, August 14-16, 1985.

[5] J.C. Doyle and G. Stein, "Multivariable Feedback Design: Concepts for a Classic/Modern Synthesis," IEEE *Transactions on Automatic Control*, Vol. AC-26, pp. 4-16, 1981.

[6] J.M. Edmunds, "Cambridge Linear Analysis and Design Program," *IFAC Symposium on Computer-Aided Design of Control Systems*, Zurich, 1979.

[7] H. Elmqvist, "SIMNON — An Interactive Simulation Program for Non-Linear Systems," *Proceedings of Simulation 77*, Montreux, Switzerland, 1977.

[8] D.K. Frederick, "Computer Programs for the Simulation and Design of Control Systems," to appear in *Advances in Computer Control Systems, Theory and Applications*, J.B. Cruz (ed.) Greenwich, CT, JAI Press, 1985.

[9] J.R. James, D.K. Frederick, and J.H. Taylor, "On the Application of Expert Systems Programming Techniques to the Design of Lead-Lag Precompensators," *Proceedings of Control 85*, Cambridge, UK, July 1985.

[10] J.R. James, J.H. Taylor, and D.K. Frederick, "An Expert System Architecture for Coping with Complexity in Computer-Aided Control Engineering," *Proceedings of the 3rd IFAC Symposium on CAD in Control and Engineering Systems*, Lyngby, Denmark, August 1985.

[11] D.A. McAllester, "An Outlook on Truth Maintenance," M.I.T. AI Memo No. 551, Massachusetts Institute of Technology, Cambridge, MA, August 1980.

[12] J.H. Taylor, "Environment and Methods for Computer-Aided Control Systems Design for Nonlinear Plants," *Proceedings of the Second IFAC Symposium: CAD of Multivariable Technological Systems*, Purdue University, West Lafayette, IN, USA, pp. 361-367, September 1982.

[13] J.H. Taylor, D.K. Frederick, and A.G.J. MacFarlane, "A Second- Generation Software Plan for CACSD," *Abstracts of the IEEE Control System Society Symposium on CACSD*, Cambridge, MA, September 1983.

[14] J.H. Taylor, D.K. Frederick, and J.R. James, "An Expert System Scenario for Computer-Aided Control Engineering," *Proceedings of the American Control Conference*, San Diego, CA, June 1984.

[15] J.H. Taylor and D.K. Frederick, "An Expert System Architecture for Computer-Aided Control Engineering," *IEEE Proceedings*, Vol. 72, pp. 1795-1805, December 1984.