Chemical Process Control III, Asilomar CA, 12-17 January 1986

EXPERT SYSTEMS FOR COMPUTER-AIDED CONTROL ENGINEERING

James H. Taylor
Control Technology Branch
GE Corporate Research & Development
Schenectady, NY 12345

Abstract. The motivation, history, and process of creating an expert-system environment for computer-aided control engineering are outlined. The resulting expert system architecture is presented, and its generic aspects relative to engineering design are discussed. Developmental issues are also mentioned, including inference and numerical requirements and interfacing with conventional analysis and design software. The status of our project is summarized, and certain lessons regarding the difficulty of achieving certain ambitious goals are presented. Other projects in this area are also referenced.

Keywords. Control system design; CAD environments; expert systems; artificial intelligence; engineering design.

INTRODUCTION

Artificial intelligence in general and knowledge-based expert systems (KBESs) in particular have recently gained prominence in the general area of computer-aided problem solving. The earliest successes were in the area of diagnosis or trouble-shooting (e.g., Mycin, Prospector, Internist, etc.). The application of this technology has been expanding with great rapidity in the last few years; refer, for example, to Hayes-Roth *et al* (1983), Coombs (1984), Michie (1982), IEEE (1984), or Davis and Lenat (1982) for a sampling of recent work. One of the more recent application areas of this technology is engineering design. The latter topic, particularly as it relates to control system analysis and design, is the focus of this presentation. The primary goals are to demonstrate what we have done in this area, to discuss what remains to be accomplished, to overview other related work, and to present a

balanced, realistic view of the promise and cost of the KBES technology in this field.

HISTORY OF GE WORK IN CACE

The Control Technology Branch of GE Corporate Research & Development (**GE CRD**) has had a substantial interest and involvement in Computer-Aided Control Engineering (**CACE**) since about 1980. Some contributions are documented in Taylor (1982), which outlines some functional requirements of a CACE environment, Spang (1982, 1984), which describes the development and capabilities of our conventional environment called the **GE Federated System**, and Taylor and Frederick (1984), which documents a high-level conceptual definition for a KBES for CACE called **CACE-III**.

The General Electric experience and approach to CACE software development has been motivated by the need for an integrated, high-level environment for control engineering activities. When we started in 1980, there was nothing available that met our requirements. There was software, both public-domain and commercial, that could perform most of the functions that were required, but not in a single, convenient environment. Not having the resources to develop "the ideal environment" from scratch, we adopted the strategy of selecting packages with the best possible functional coverage, and "federating" the packages to create the desired capability and unity. The term "federated" signifies that the packages are still available for stand-alone use; generally, however, they are run under a supervisory program that manages the environment.

We developed our software in two major phases: In the first part of the effort, we purchased and/or developed software that could carry out the essential procedures of CACE with numerically robust algorithms and reasonably convenient and consistent user interfaces, and integrated them into a unified conventional system. This effort culminated in a software environment we call the *Federated System*, which is completely developed, documented, and delivered to a user group made up of controls engineers at about ten GE locations.

In the second part of our development, started in 1983 in a low-level exploratory manner, we focussed on making this software more accessible to less-than-expert users and relieving the user of much of the burden of using a sophisticated and complicated environment. We used the expert systems approach to accomplish this (Taylor and Frederick, 1984). This work was motivated by our own experience

with the Federated System and other CACE software, and by the comments and suggestions of our user group.

The outcome of this research is an architecture, a detailed specification, and a prototype system called *CACE-III* which is, in our opinion, a *third-generation* CACE environment; hence its name. In contrast, first-generation tools are single-purpose software products (root locus routines, transfer function plotting programs, etc.) and often batch-oriented, and second-generation environments are functionally broader, interactive, conventional software; according to this classification, the Federated System is a second-generation environment for CACE. This third-generation project is still underway.

The GE Federated System

The Federated System was developed by GE CRD for internal use and for the GE Aerospace Business Group and Aircraft Engine Business Group. This sponsoring user group played a partnership role with GE CRD in defining the system. The basic approach that guided the development of the Federated System was as follows:

- define the required functionality,
- survey available CACE packages and choose a set of them that meets the functional requirements (to the extent possible), with numerically respectable algorithms and convenient user interfaces,
- extend packages or develop additional packages to meet remaining unmet needs,
- develop interface routines that can convert models and data into forms that are appropriate for each package,
- integrate the packages and routines into a unified, effective environment for CACE, and
- document and deliver the resulting software system to our user group.

The initial core package set (selected in 1981) included:

- CLADP, the Cambridge Linear Analysis and Design Program, which implements recent "British School" multivariable frequency-domain design methodologies (Edmunds, 1979),

- SIMNON, a nonlinear simulation package from Lund University (Sweden) that supports both its own modeling language and FORTRAN system models and allows the user to set up and execute simulations in a very flexible, command-driven environment (Elmqvist, 1977),
- IDPAC, a time-series analysis package from Lund University that performs statistical analysis (correlation and spectral) and parametric model identification (least squares and maximum likelihood) (Wieslander and Gustavsson, 1976), and
- SSDP, a modern state-space design package built at GE CRD using the I/O routines of CLADP and underlying numerical software from reputable sources.

More recently, MATLAB (Moler, 1982) was added to the Federated System. These core packages met most of the required functionality with the exception of strong nonlinear analysis and design capabilities and control system implementation.

This environment was integrated and extended to include appropriate model data-base transformations and overall functional compatibility (Taylor, 1982; Spang, 1982, 1984). In some cases, the data-base transformations were simply re-formatting linear system models obtained from mass storage, in other cases the interface was considerably more challenging. The most complicated transformation was from SIMNON nonlinear models to CLADP linear models, which required the installation of a new linearization routine. This capability also underscores the necessity of such interfaces: One of the most time-consuming and error-prone procedures in CACE is the manual generation and entry of linearized models.

Functional compatibility was achieved primarily through the use of interface routines. The most important feature in this sense is the provision for converting the compensator designed in CLADP into the SIMNON modeling language, so that the user can directly validate the design via simulation with the nonlinear plant model. Another feature incorporated for compatibility is equilibrium-finding, which is usually a necessary precondition for linearization.

Two additional capabilities were incorporated to deal with the analysis and design of nonlinear systems: a routine to generate describing function input/output models of nonlinear plants, and design routines to synthesize nonlinear compensators based on these models using a methodology developed at GE CRD (Taylor and Strobel, 1984). These enhancements to the Federated System are described in some

detail in Taylor (1985b).

This approach to the design of a state-of-the-art conventional CACE environment has been highly successful. By making the best possible use of existing software, we have achieved broad and complete coverage of the CACE problem. Throughout this work, we have strived to make this environment as broad, powerful, practical, and "real-problem-oriented" as possible. The two most striking attributes of the Federated System are its breadth and its substantial capabilities to handle nonlinear problems (simulation, analysis, and design).

In the course of this activity, we had to consider a number of issues. In addition to functionality, primary considerations were software package selection, integration (e.g., "federating"), and data-base management (DBM). The solutions to the problems we encountered in these areas are detailed in Spang (1982, 1984). The following additional points should be considered:

- The trade-off between generic versus special-purpose environments may be an important consideration. The Federated System is a general-purpose environment; a narrowly-focussed environment would be quite different. There are two aspects to specificity: a special-purpose package might support just one approach (e.g., linear systems and linear quadratic regulator design), or it might support users from one discipline (e.g., chemical process control). The software selection would be influenced by both factors, and even the content and style of the user interaction could be "tailored" to suit the terminology and procedures generally used in a particular domain of application.

- The need for DBM should be assessed carefully. Few of our applications are multi-disciplinary, so the need for real DBM (e.g., keeping track of numerous models and analysis data files for a single project including the relations among the files) was not particularly stringent. We thus accepted the philosophy of the Federated System's core packages and limited DBM to file-naming conventions for files created by the various procedures. In many situations, a system that goes beyond simple file-naming conventions may be required.

- Finally, the CACE software scene has changed considerable in the last four years; it is quite possible that our software selection today would be different.

The impact of the first two issues has been discussed in Taylor (1985a), where some

specific concepts in the areas of "tailoring" and DBM for integrated flight and engine control are outlined.

While much of the reaction to recently-developed CACE software (the Federated System and other packages) has been very positive, there have been were a substantial number of complaints related to "user friendliness". In the context of our work, these may be summarized as follows: It is difficult to remember all the details (commands and syntaxes, etc.) involved in using a comprehensive set of CACE packages, it is difficult to keep on top of the analysis and design process, data-base management is too rudimentary, it is not always clear what approach or even which package to use, it is difficult to deal with all the error conditions (error messages and corrective actions) and other aspects of using the software, etc. In short, one must be an expert, every-day user to take full advantage of the existing software, which can be a considerable frustration to less-than-expert or occasional users. These considerations prompted us to consider the use of KBES methods to improve the situation.

CACE-III

The basic thrust of our more recent research in CACE is the creation of a unified expert-aided environment for the full spectrum of control engineering activities, from high-fidelity modeling through control system design and validation (Taylor and Frederick, 1984). Two key issues are the complexity of control engineering problem solution and the broad spectrum of the user capabilities we wish to serve. The first point is clearly illustrated by listing the typical activities of a controls engineer: nonlinear modeling, simulation, equilibrium-finding, linearization, analysis of linear plant models, control system design (including trade-off analysis and tuning), and design validation. The latter issue we have summarized by saying that we are striving to create an environment "that can be a teacher to the novice, a partner to the more experienced controls engineer, and an assistant to the expert". In every case, we want to eliminate or reduce as much as possible the amount of unnecessary detail the user has to contend with, especially in the areas of software package usage (choosing the most appropriate algorithm, package and command selection, interpreting error messages and taking corrective action where possible, etc.) and keeping track of models and data (data-base management).

A fundamental conclusion of our research is that this goal can best be achieved by the use of the knowledge-based expert system (KBES) approach. This conclusion was reached by specifying the functional requirements of such an environment and showing how they can be met in a KBES, and contrasting such a system with existing conventional software environments. This comparison was made on the basis of our substantial experience in conventional CACE software development and integration as outlined above; our approach and findings are discussed in more detail in Taylor and Frederick (1984) and below.

In the discussion that follows, we present broad concepts and issues involved in creating an expert-aided environment for CACE. We accomplish this by outlining certain considerations regarding the use of the expert systems methodology, describing the development of CACE-III (a prototype KBES to demonstrate the capabilities and benefits of an expert-aided environment), illustrating the basic elements of a rule-based expert system, and discussing the lessons we have learned in the course of this project. The lessons specifically relate to the cost of developing "real" expert systems, the limitations of basing KBES development on "scenarios", and the difficulty of making software usage details invisible and supporting a variety of user capabilities.

EXPERT SYSTEMS CONSIDERATIONS

Expert or knowledge-based systems are software environments designed to aid in solving problems that require high levels of *expertise*, some degree of *inference* ("reasoning"), the use of *heuristics* (nonrigorous procedures or "rules of thumb"), and the systematic processing of *symbolic information*. Such problems are generally complicated and broad in scope, and are not amenable to clear-cut well-posed algorithmic solutions. Control system engineering is certainly such a task: The types of expertise that are required for CACE problem solving are: development and diagnosis of plant models, formulating a realistic controls design problem, selecting appropriate analysis and design methods, performing design tradeoffs, validating and documenting designs, and making effective use of conventional CACE software. Symbolic information to be processed in CACE includes descriptions of methodologies for CACE, the status of the current problem solution, the names and capabilities of CACE packages, their command sets, syntaxes, and error handling, data and model relations for DBM, etc.

Heuristics play a major role in a human expert's ability to formulate a well-posed CACE design problem, and reasoning capability is advantageous for directing and keeping track of the design process as it progresses. Also, the KBES approach provides a high-level, flexible, and supportive environment that can relieve the user of much of the low-level detail and drudgery involved in using a number of large software packages and in DBM. Further motivation for the use of the expert systems approach may exist for users in multi-disciplinary fields such as integrated flight and engine control: Few individuals can be expected to be expert in all aspects of a multidisciplinary problem, so expert-aiding is often essential to the production of meaningful analysis and designs. For more information on expert systems or for additional background reading in the area, refer, for example, to Hayes-Roth *et al* (1983), Coombs (1984), Michie (1982), IEEE (1984), or Davis and Lenat (1982).

The application of expert systems technology is often based on the idea of providing support for "less-than-expert users". This thought requires some caution, especially in our context. First, there are several factors involved in this phrase: In engineering design, such a user may not be aware of the latest theoretical developments in all fields involved in the problem to be solved, may not have had the experience required to synthesize theory into an effective analysis and design approach, and/or may not be a frequent practitioner of system design or user of the required software. Considering the breadth of knowledge required for CACE, especially in multidisciplinary fields, it is probably not reasonable to expect that many users of a major software environment for this activity will be experts in all aspects of the problem being solved. Providing support for "less-than-expert users" in this sense is desirable, reasonable, and feasible. On the other hand, it should be clearly understood that it is not appropriate to speak of developing an expert-aided CACE environment for use by personnel with little or no knowledge of controls: Too much understanding and common sense are required of the user for this to be practicable.

Another potential problem in the use of expert systems is the possibility of unrealistic expectations. We believe that it *is* realistic to use this methodology to raise the level of interaction for users that are basically competent and to support them in the ways outlined above, but it is *not* reasonable to expect that such a system will be *fool-proof* and able to provide the *optimal* solution to all problems. These factors make it imperative that such a system be designed to keep the user in a position of responsibility and authority. The latter issue - authority - requires that the

system be flexible enough that the user will be supported to the extent needed and possible, without dominating the proceedings and forcing the user to accept undesired or meaningless solutions. It should be appreciated that this may be a major undertaking, and that there have to to be limits to the freedom of the user.

With these clarifications of the basic objective of an expert-aided system for CACE, we proceed to develop the idea further. This concept is based on the motivational issues, general functional requirements, and basic expert system ideas presented in Taylor and Frederick (1984).

CACE-III DEVELOPMENT

Our approach to conceiving and developing an expert-system CACE environment combines top-down and bottom-up CACE process modeling. By the first term, we mean working with experts in the fields of CACE to determine the conceptual and structural aspects of the problem and distilling that information into an expert system architecture. Bottom-up modeling referred to developing "scenarios" that show in detail how the expert system should serve and interact with the user (Taylor, Frederick, and James, 1984; Taylor, 1985a). A combination of these activities, in parallel with the development of corresponding real expert system architectures and rule bases, has proven to be extremely effective. Only top-down modeling is discussed in any detail in this presentation, so that the CACE-III architecture can be well explained and motivated. Certain generic aspects of this work are also pointed out.

Architecture Development

We used top-down modeling of the CACE methodology of a human expert as a way to develop detailed requirements and architectural concepts for CACE-III. The first outcome of this approach was the realization that a *complete, meaningful problem formulation* is a central issue in capturing the control system design process. This may be represented by a "list of facts" or, in artificial intelligence terminology, "frame". The information in this frame may be organized or partitioned into three components with the following informational content:

- *Plant model characterization* - ordinary or partial differential equation models, linear or nonlinear, stable or unstable, minimum or nonminimum phase, (un)controllable and/or (un)observable, et cetera.

- *Constraints* - architectural (e.g., centralized or distributed control), implementation (e.g., analog or digital), parametric (e.g., gains, data rate limits), et cetera.
- *Specifications* - time response, frequency response, performance indices, et cetera; sensitivity, disturbance rejection, robustness, et cetera.

This is illustrated in Fig. 1; such a *problem frame* is a major focal point for CACE.

Given a meaningful control design problem, we saw the human expert working in a parallel construct, which we call the *solution frame.* This is a list of facts that is developed as a "scratch pad" where the expert system keeps track of both the *data base* (models, analysis data) and the *process* (what has been done and what needs to be done, information required for decisions about the selection of design procedures and tradeoff analysis, and a log of the entire transaction). This is shown in Fig. 2.

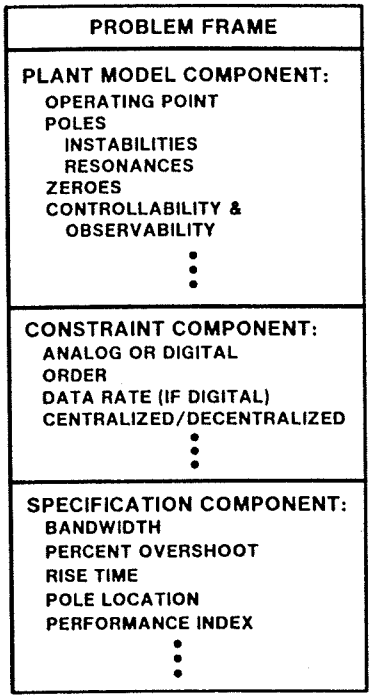

Figure 1. The CACE Problem Frame

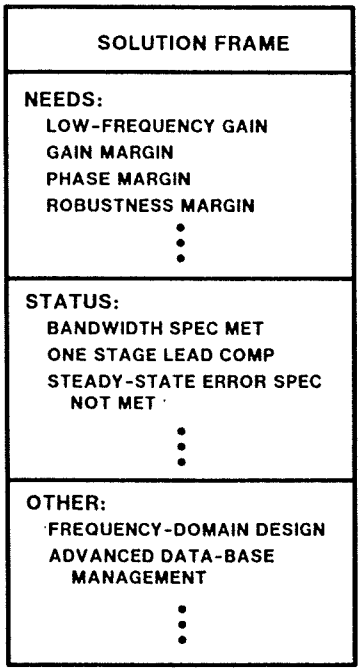

Figure 2. The CACE-III Solution Frame

These structured lists of facts gave us a basis for developing *rule bases.* The process proceeds by asking: What are the *functions that must be implemented* in order to produce and manipulate these facts to solve the problem? This line of thought -

linking the activities of an expert with two key lists of facts (frames) and associated rule bases - gave rise to a functional structure of CACE-III that is depicted in Fig. 3. In particular, we have created a construct in which the rule base is partitioned into six functional parts, as shown, and a seventh supervisory rule base (not portrayed) which is discussed below. The functions of the rule bases in CACE-III may be summarized as follows:

- RB1 governs interactions among the *design engineer, plant models* (nonlinear and/or linear), and the *model component of the problem frame.* This rule base provides support in model development (including diagnostics relating to the physical process and suitability of models for control system design and numerical analysis), and sees to it that all required plant data are added to the knowledge base. Examples of the activities of RB1 are illustrated in scenarios in Taylor, Frederick, and James, 1984, and Taylor, 1985a.

- RB2 governs interactions between the *design engineer* and the *constraint and specification components of the problem frame.* Constraints are requested but are not mandatory; if supplied, these are also written into the list of facts that makes up the problem frame. These rules guide the user in entering design specifications and checks specifications for consistency, completeness, and achievability (realism). For example, consistency checks include being sure that the damping ratio dictated by percent overshoot and rise-time-to-settling-time ratio agree, etc., and realism tests include determining the specifications that can be achieved by a simple design approach (e.g., use of a standard PID design algorithm) and checking to see that the user's specifications are not excessively more demanding.

- RB3 and RB5 govern interactions between the *problem frame* and the *solution frame.* RB3 deals with specifications, constraints, and plant characteristics, and initializes the list of facts in the solution frame describing what needs to be done to achieve design goals. If design iteration or tradeoff analysis is called for, then RB5 supports the user in selecting the specifications to vary (relax or tighten) and modifies the problem frame appropriately, RB3 resets the solution frame accordingly, and the required set of designs is carried out by RB4. Finally, RB5 will present the results of the iteration or tradeoff analysis.

- RB4 governs interactions between the *solution frame* and the available *design procedures.* These rules decide what design approach(es) will best solve the problem

(e.g., by matching approaches with specifications), executes the appropriate procedure(s) and algorithm(s) using conventional design software, and updates the solution frame to reflect the corresponding addition/change in the system. RB4 also performs a preliminary validation by checking that all specifications are met with the linear plant model and controller.

- RB6 governs the final control system *validation process* (which generally involves highly realistic simulation or emulation of the plant and controller), conversion from idealized controller design to *practical implementation*, and *documentation*. The last step involves archiving a record of the design process, including tradeoffs and information supporting all design decisions, and a record of the data base (model and data files, including information regarding assumptions and conditions for validity).

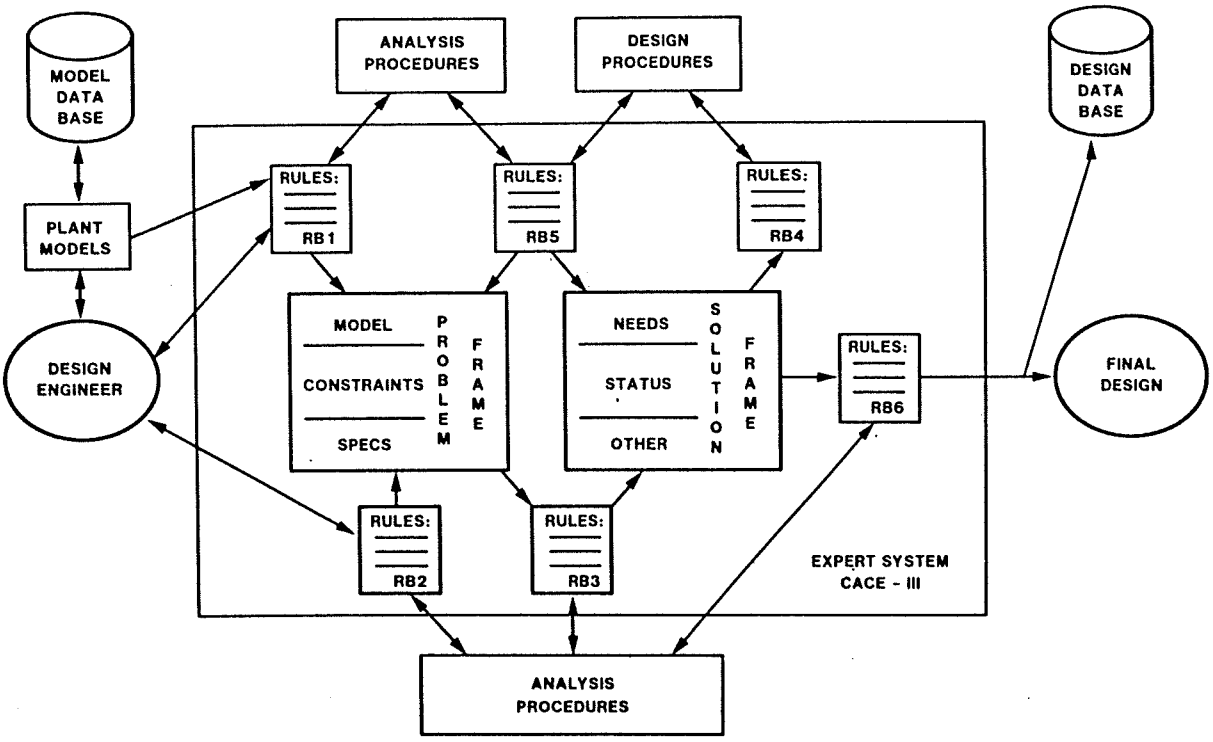

Figure 3. The Architecture of CACE-III

The goal of the first two rule bases is to have a well-formulated problem, thus ensuring a reasonable probability of success in the design phase. They were

conceived as outlined above simply by considering the informational content of the problem frame and determining what functionality must be implemented in order to arrive at a meaningful problem. Observe that RB3, RB4 and RB5 may represent an iterative or dynamic "loop": In some cases the KBES must invoke these rule bases repeatedly until all specifications are satisfied, if possible. Again, these rule bases were established based on our model for how an expert controls engineer sets up and solves design problems. Finally, RB6 provides a more rigorous assurance that the user has a control system that will perform as required, with as little need as possible for additional engineering for implementation, and with the required engineering documentation.

We have found that the rule base partitioning depicted in Fig. 3 serves two important functions: First, it clarifies many conceptual aspects of the CACE process, and thus provides a basis for rule base development. In addition, this structure has proven to be useful for developing "meta-rules" (AI terminology for "rules about rules") that increase the efficiency of the expert system by partitioning the rule base and restricting the scope of the expert system to the rules needed for the task presently at hand. These meta-rules are contained in a seventh supervisory set of rules (James, Taylor, and Frederick, 1985) which monitors the status of the problem solution (as established by RB1-RB6) and determines when a rule base has completed its work and what rule base to invoke next.

The above rule bases cause the KBES to invoke conventional CACE software in performing all analysis and design functions. Expertise regarding the use of this software is "built-in", so that the user need not know command sets, syntaxes, and error messages. To the extent possible, the expert system knows how to interpret error messages and take corrective action. A protocol for coordinating the operation of the expert system and the CACE software is completely designed and implemented (James, Taylor, and Frederick, 1985). Finally, this expert system architecture can also support DBM, in the sense that the expert system has access to the relational information necessary for this function and can either perform the activity itself or supply the information to separate DBM software, as discussed in more detail below.

A Paradigm for CAE

The primary goal of our research was to provide a clear vision of an expert system for CACE that goes beyond the generality that "artificial intelligence promises to revolutionize engineering design". In retrospect, we believe that the CACE-III expert system architecture speaks to some generic aspects of engineering design, thus adding substance to this statement.

Several aspects of the development of CACE-III would seem to pertain to engineering design in general. First, the **motivation** for considering the creation of expert systems for engineering analysis and design in other fields is similar to that for CACE:

- The elements of engineering design are generally closely patterned on the tasks defined above for CACE: model, constrain, and specify to arrive at a well-posed problem; select and execute design approaches; perform trade-off studies if necessary; validate; document; implement.
- As a corollary to item 1, most engineering fields are inherently broad in scope, i.e., encompasses a wide variety of activities, and requires inference, judgement, and heuristics.
- We conjecture that in many areas engineering analysis and design software suffers from the same general deficiencies as we discussed previously - in particular, from being rather low-level, unsupportive, or otherwise not "user-friendly" to the less-than-expert or occasional user.
- Most areas of engineering are "dynamic", in the sense that new methods are continually being created and software is continually being developed to deal with the problem on a broader basis or higher level; the KBES methodology can accommodate evolution and transition new approaches to the user very effectively.
- There is substantial pressure to reduce the engineering time spent in analysis and design, to improve the quality of design, to aid the less-than-expert user, and to better document the design process.

Having summarized our reasons for believing that KBESs for engineering design will prove to be of substantial value in many disciplines, we will conclude by considering the **architecture** of such expert systems:

The structure portrayed in Fig. 3 was developed by a process that might be thought of as "modeling" the procedures of a human expert. For this reason, the similarities in terms of the activities required in engineering analysis and design (the first point above) and in the underlying thought processes of human experts gives us reason to believe that the expert system architecture depicted in Fig. 3 is strongly generic. In other words, it is our belief that the rule base structure of CACE-III can be used by expert system developers from other fields quite directly, merely by substituting the appropriate functions into RB1 through RB6.

AN EXPERT SYSTEM ENVIRONMENT

CACE-III was first developed using a GE CRD expert system environment called DELTA, a rule interpreter or "inference engine" created at GE CRD for diesel-electric locomotive trouble-shooting (Bonissone, 1983). This shell did not adequately support our needs, which included running and communicating with conventional software and performing numerical procedures. We then migrated to a more recently-developed and more powerful GE CRD expert system environment called **Delphi.**

While space does not permit a detailed discussion, we will outline some aspects of the operation of DELTA to the extent required to convey a basic understanding of rule-based expert systems. The goal is to illustrate the spirit of the expert system approach, not a working knowledge of DELTA or Delphi. This overview deals with considerations that are primarily generic to the use of rule-based expert systems, although the details differ from system to system.

Rules

The DELTA inference engine basically works with rules having the form:

IF [$Premise_1$ $Premise_2$ ...]

THEN [$Conclusion_1$ $Conclusion_2$ ...]

Premises may be thought of as defining a condition, situation, or state of a process; conclusions are actions to be taken if the premises of the rule are satisfied. Actions include adding facts to the knowledge base (list of facts), clearing facts from the knowledge base, or asking the user to determine what specific fact(s) to add to the

knowledge base.

Each premise and conclusion in CACE-III has the form of a three-tuple which is true, false, or neutral (of unknown truth), i.e., each three-tuple may take on certainty values of +1, -1, and 0. A three-tuple has the form [OBJECT ATTRIBUTE VALUE]; for example, the fact [SPEC-CL-POLE MAX-REAL-PART UNASSIGNED] +1 denotes that no value has been specified for the maximum real part of the poles of the closed-loop control system being designed.

A *premise* may be that a three-tuple is true,

EQ [OBJECT ATTRIBUTE VALUE],

or false,

NE [OBJ ATTR VAL].

The *conclusions* of a rule may include any of the following actions:

1. DISPLAY, which indicates that the user should be notified about something. This conclusion must be followed by the message to be conveyed (in parentheses);
2. UDO, which indicates that the user should take some action. This conclusion must be followed by a statement of the action to be taken that will be displayed to the user (in parentheses);
3. WRITE [OBJ ATTR VAL], which indicates that a three-tuple should be written into the knowledge base;
4. CLR [OBJ ATTR VAL], which indicates that a three-tuple should be cleared from the knowledge base (certainty value set to 0);
5. ASK [OBJ ATTR VAL], which indicates that the user should be asked to determine the status of the three-tuple (true or false); or
6. MENU, which allows the user to make a selection from a list of options.

Observe that in the case of DISPLAY, UDU, ASK, and MENU each parenthetical expression is a message to the user; in other cases and in the premise portion of the

rule, the phrases in parentheses are simply comments explaining the thinking of the rule base creator. These "comment lines" may serve as the basis for a "why" or "help" facility. Most of these ideas are illustrated in the two sample rules provided in Table 1, which are taken from the CACE-III rule base that governs the specification entry process.

Table 1. TWO RULES FROM CACE-III

```
RULE 106 ( Want to enter max Re part spec; already assigned )
IF:
  EQ [ SPEC-C-L-POLE MAX-REAL-PART REQUESTED ]
    ( User has asked to enter max-real-part )
  EQ [ MAX-REAL-PART VALUE + ]
    ( A value is currently assigned )
THEN:
  CLR [ SPEC-C-L-POLE MAX-REAL-PART REQUESTED ]
    ( Reset - clear the fact that triggered this rule )
  DISPLAY
    ( You want to enter a value for the maximum )
    ( real part of the poles, but a value has been )
    ( assigned previously. )
  ASK [ MAX-REAL-PART MODIFY VALUE ]
    ( Do you wish to replace the current value? {Y or N} ... ENTER: )
```

```
RULE 108 ( Current value of max Re part is to be replaced )
IF:
  EQ [ MAX-REAL-PART MODIFY VALUE ]
    ( User wants to modify the current value )
THEN:
  CLR [ MAX-REAL-PART MODIFY VALUE ]
    ( Reset - clear the fact that triggered this rule )
  CLR [ MAX-REAL-PART VALUE + ]
    ( Delete the old value )
  WRITE [ SPEC-CL-POLE MAX-REAL-PART UNASSIGNED ]
    ( Reset - pave the way for a new assignment )
  WRITE [ SPEC-CL-POLE MAX-REAL-PART REQUESTED ]
    ( Trigger the rule to request entry of new value )
```

Observe that a 'yes' response to the query in the conclusion of RULE 106 results in the three-tuple [MAX-REAL-PART MODIFY VALUE] being written to the list of facts as 'true'. This action satisfies the premise of RULE 108, thus triggering it. RULE 108, in turn, will activate the rule that asks the user for a value of SPEC-CL-POLE MAX-REAL-PART. Note that the '+' sign is a "wild card", denoting "any

value".

Inference Mechanisms

DELTA and Delphi support inference in two modes: *fact driven* and *goal driven.* In the former case, the system systematically checks some or all the rules for premises that are known to be true so that the corresponding fact(s) can be written into the list of facts (knowledge base). In the *goal driven* mode, the system attempts to reach a specific conclusion (write a specific fact) by identifying a rule or rules that contain the desired goal in the conclusion and using its inference capabilities to prove that the premise(s) of the desired rule(s) are satisfied. The AI terminology for these two modes of operation are *forward* and *backward chaining,* respectively. An example of the first inference mode is filling in the plant characteristics in the problem frame, while attempting to achieve the goal "design specifications satisfied" exemplifies backward chaining.

The forward chaining inference mode operates by examining each rule and asking: Is $Premise_i$ known to be true? If the answer to this question is "yes" for every premise of the rule, then its conclusion is carried out, and the inference engine proceeds to consider the next rule. This process may be carried out exhaustively, or it may be limited either by features of the inference engine or by structuring the rules.

Backward chaining starts with a goal: write $Fact_k$ to the list of facts. First, the inference engine seeks a rule that can cause $Fact_k$ to be written, i.e., that has WRITE [$Fact_k$] among its conclusions. Then, a four-stage attempt to satisfy the premises $Premise_i$ of this rule is initiated, by asking:

1. Is $Premise_i$ known to be true (in light of the existing list of facts)?
2. Can $Premise_i$ be inferred from the current list of facts (using another rule or several rules)?
3. Can conventional analysis and design procedures be used to determine if $Premise_i$ is true?
4. Can the user determine if $Premise_i$ is true?

If the answer to any of the above questions is "yes", for each premise $Premise_i$ of the

rule, then the goal has been achieved and $Fact_k$ is written into the knowledge base. In the case of an "if-and-only-if" rule, showing that a premise is false will terminate the backward chaining process. Otherwise, the backward chainer will continue to seek rules that can be satisfied so that $Fact_k$ can be written. The questions are always asked in the above order, so that running external procedures and asking the user are only done as last resorts.

Lists of Facts

The status of a session at any given time, or the outcome of a session when completed, is characterized by the list of facts that has been written in the course of the transaction. Such a list is illustrated in Table 2, which contains the list of facts that might exist in the CACE-III knowledge base at the end of a DIAGNOSE and SPECIFY session. In order to interpret Table 2, we provide the following brief guide to the shorthand notation in our list of facts: NL → nonlinear, FNAME → file name, DIAGN → diagnosis, L → linear, DIAG-DOMINANT → diagonally dominant, SS → steady-state, CHi → channel_i of a multi-input multi-output plant. Observe that the underlying *data*, e.g., the numerical results of the nonlinear system diagnosis and the linearized model and its diagnosis, may be contained in ancillary files. The expert system often does not use this kind of data directly, but it must know where such information can be found so that it can be provided to external analysis and design procedures as required. The basis designation RBi refers to the rule base partitioning outlined above.

The preceding overview outlines the three major elements of a production rule-based expert system: rules, facts, and inference engine. Each component is in fact disarmingly simple. The power of this programming approach arises from this partitioning, which separates the knowledge into rules and facts, and deals with them in a dynamic fashion using a processor (inference engine) which may be simple or quite sophisticated. The capability of the processor is generally driven by the complexity of the problems to be solved; it is clearly beneficial to have the correct "match" to avoid the extremes of "overkill" (using a powerful inference engine to solve simple problems, usually at a high cost in terms of computational and response time) and "brute force" (having to develop an unnecessarily large rule base of simple-minded rules that may also take a long time to process). This division achieves a high degree of transparency, so that the rules can be kept lucid (assuming the expert system

Table 2. THE PROBLEM FRAME AFTER A DIAGNOSE AND SPECIFY SESSION

FACT			BASIS
OBJECT	ATTRIBUTE	VALUE	
PLANT	MODEL	NONLINEAR	User: RB1
PLANT-NL-MODEL	FNAME	EXOREACT	User: RB1
PLANT-NL-MODEL	TIME-TYPE	CONTINUOUS	Inferred: RB1
PLANT-NL-MODEL	STATE-TYPE	CONTINUOUS	Inferred: RB1
PLANT-NL-MODEL	ORDER	2	Inferred: RB1
PLANT-NL-MODEL	INPUTS	2	Inferred: RB1
PLANT-NL-MODEL	OUTPUTS	2	Inferred: RB1
PLANT-NL-MODEL	DIAGN-DATA-FNAME	EXORNDATA	Inferred: RB1
PLANT-NL-MODEL	NL-BEHAVIOR	MILD	Inferred: RB1
PLANT-L-MODEL	FNAME	EXOREACTL	Inferred: RB1
PLANT-L-MODEL	STABLE	NO	Inferred: RB1
PLANT-L-MODEL	CONTROLLABLE	YES	Inferred: RB1
PLANT-L-MODEL	OBSERVABLE	YES	Inferred: RB1
PLANT-L-MODEL	MINIMUM-PHASE	YES	Inferred: RB1
MODEL	DIAGNOSIS	DONE	Inferred: RB1
SENSOR	TIME-TYPE	CONTINUOUS	User: RB2
CONTROLLER	TIME-TYPE	CONTINUOUS	User: RB2
CONTROLLER	STRUCTURE	DIAG-DOMINANT	User: RB2
CONTROLLER	CHANNEL1-IN	U1	User: RB2
CONTROLLER	CHANNEL1-OUT	Y1	User: RB2
CONTROLLER	CHANNEL2-IN	U2	Inferred: RB2
CONTROLLER	CHANNEL2-OUT	Y2	Inferred: RB2
MAX-STEP-SS-ERR	CH1-VALUE	0.25	User: RB2
MAX-REAL-PART	CH1-VALUE	-1.4	User: RB2
MIN-DAMPING-RATIO	CH1-VALUE	1.0	User: RB2
MAX-STEP-SS-ERR	CH2-VALUE	0.5	User: RB2
MAX-REAL-PART	CH2-VALUE	-1.4	User: RB2
MIN-DAMPING-RATIO	CH2-VALUE	1.0	User: RB2
CONTINUOUS-SPEC	ENTRY	DONE	Inferred: RB2
CONTINUOUS-SPEC	ENTRY	REALISTIC	Inferred: RB2
CONTINUOUS-SPEC	ENTRY	COMPLETE	Inferred: RB2
CONTINUOUS-SPEC	ENTRY	CONSISTENT	Inferred: RB2
SPEC-SESSION	TERMINATION	NORMAL	Inferred: RB2

developer has a clearly-enunciated problem and solution formulation) and the inference mechanisms can be developed and validated separately.

Other Inference Engine Requirements

The above discussion provides a basic overview of selected expert system concepts, in particular, of rules, lists of facts (knowledge bases), and two modes of inference. There are other requirements for engineering design that go substantially

beyond the needs for many other well-known expert system applications found in the general literature cited previously. Important additional capabilities are: the ability to run and communicate with external processes, the ability to perform numerical computations and tests, support for a partitioned rule base, non-monotonic reasoning, and the ability to be controlled or "steered" by the user.

In order for CACE-III to execute the analyses required for diagnosis and design, it is necessary for the inference engine to run external processes (conventional CACE software, e.g., the Federated System) that carry out tasks such as calculating the frequency response of the plant, determining gain margin and bandwidth, simulation, etc. CACE-III must initiate the process with the appropriate input parameters, and must be able to access the results of the analysis in order to update the list of facts. The DELTA inference engine we initially used in our research did not have this capability; converting to Delphi allowed us to perform this function.

The DELTA inference engine could not deal directly with numbers either, having only the ability to interpret and use literal strings, such as FOUR, POSITIVE, and BETWEEN-THREE-AND-FIVE. This restriction was not important in the original application (the diagnosis of diesel-electric locomotives; Bonissone, 1983), but in our case it required the use of numerous additional rules and menus for the entry of data in literal form. This limitation also does not exist in Delphi.

We found that CACE calls for a partitioning of the rule base into six parts plus a seventh supervisory rule base ("meta-rule base"). This was not supported by Delphi, so we extended the shell to accommodate this functionality (James, Taylor, and Frederick, 1985).

Non-monotonic reasoning is often required in engineering problem solving. This phrase designates a process in which the problem solution develops for a number of steps (procedures are carried out and facts accumulate), then it is necessary to backtrack and try again taking a different path. This happens whenever specifications cannot be met or a trade-off study is required, for example. In such instances, it is necessary to reset the facts by defining a previous point in the problem solution to be the new "current state" of the KBES and eliminating the facts developed after that point ("retracting belief"). It may be necessary to save the KBES state (in the case of design trade-off study), or the information might be partially or completely discarded (in the case of a dead-end). In any event, the retraction of

belief may be difficult to manage in full generality, but can be implemented if it is permitted only in carefully-defined situations (e.g., design trade-off studies); refer to James, Bonissone, Frederick, and Taylor (1985).

The last requirement -- steerability -- will probably be crucial for the acceptance of any expert system for engineering design such as CACE-III by the engineering community. If a KBES is always in complete control, the engineer will become frustrated and/or bored. Frustration would be caused by having to accept the dictates of the software even when the user's experience contradicts them, while boredom would result from the tedium of being led by the hand through familiar parts of the task (e.g., specification entry).

We have identified the following mechanisms for allowing the designer to interact effectively with CACE-III in the design process:

1. The support provided by CACE-III is in the form of suggestions rather than constraints. If the user wants to ignore or not fully comply with the expert system's recommendations, then the software will not force compliance.

2. A 'why' facility can be invoked to determine the basis for a recommendation, so that the user can judge the advice before deciding whether or not to accept it.

3. A manual fact-entry capability can be added to allow the engineer to write facts directly into the knowledge base. For example, the goal of the specification entry portion of a transaction may be to write the following facts:

   MAX-STEP-SS-ERR VALUE 0.005
   MAX-REAL-PART VALUE -1.4
   MIN-DAMPING-RATIO VALUE 1.0

   Allowing a designer who does not require help to enter these facts directly will streamline the session considerably; the alternatives (e.g., by being led through a series of menus, Taylor, Frederick, and James, 1984) may be much less desirable to the experienced user. The use of a natural language interface would facilitate this type of user interaction greatly; we have not developed this idea at this point in our work.

4. The user can be permitted to invoke any underlying conventional CACE package and do whatever is desired.

These capabilities should help the user to maintain control of the design process in an effective way. The use of a 'why' facility is well known and exists in DELTA and Delphi; the other ideas (especially items 3 and 4) are specifically targeted for engineering users and may not be in commonplace usage. The last two features may also be difficult to implement; see LESSONS below.

The above comparisons of DELTA and Delphi were made primarily to contrast the requirements of an inference engine that is designed for pure symbolic manipulation tasks, such as diagnosis and maintenance, with those of an environment for CAD, where numerical processing and interactions with conventional analysis and design software are essential. The additional capabilities of Delphi are based on the original specification or definition of Delphi and were implemented using standard LISP functionalities.

CACE-III PROJECT STATUS

So far, we have developed a number of the necessary expert-system concepts, architectures, and real software implementations needed for an expert-aided environment for CACE:

- a *definition* for CACE-III, including a specific, detailed outline of the functional characteristics of the expert system and of the rule base,
- an *architecture* for CACE-III, including a partitioned rule-base structure that effectively implements all of the required activities (see Fig. 3) and an inference engine that supports it,
- *working rule bases* for several functions, including equilibrium finding and linearization, linear and nonlinear system model diagnosis, specification development, an automatic control system design procedure (lead-lag compensator design for a single-input/single-output plant, James, Frederick, and Taylor, 1985), and some aspects of validation (simulation with linear and nonlinear plant models).
- a proven means of interfacing the KBES with the required conventional modeling, simulation, analysis, and design software, and handling the interfaces among the various software packages (data and model transformations), and

- mechanisms providing the user with a high-level interface that provides support flexibly and without over-domination.

The CACE-III project we have described above is still in the exploratory or concept development phase. We have a working expert system that can perform major parts of the problem creditably, e.g., designing a controller using a conventional frequency-domain design package; this activity alone requires 70 rules which cause about 100 commands to be issued to the design package CLADP in the course of one design. To illustrate the capabilities of this system, CACE-III designed a high-order lead/lag compensator for a fifth-order plant G(s) with a pair of lightly damped poles; both the plant and compensator are portrayed in Fig. 4. The performance of the resulting closed-loop system is contrasted with that of a constant-gain compensated system in Fig. 5. This design exercise took about 15 minutes on a VAX 11/785, including specification development, design, and validation by simulation (James, Frederick, and Taylor, 1985).

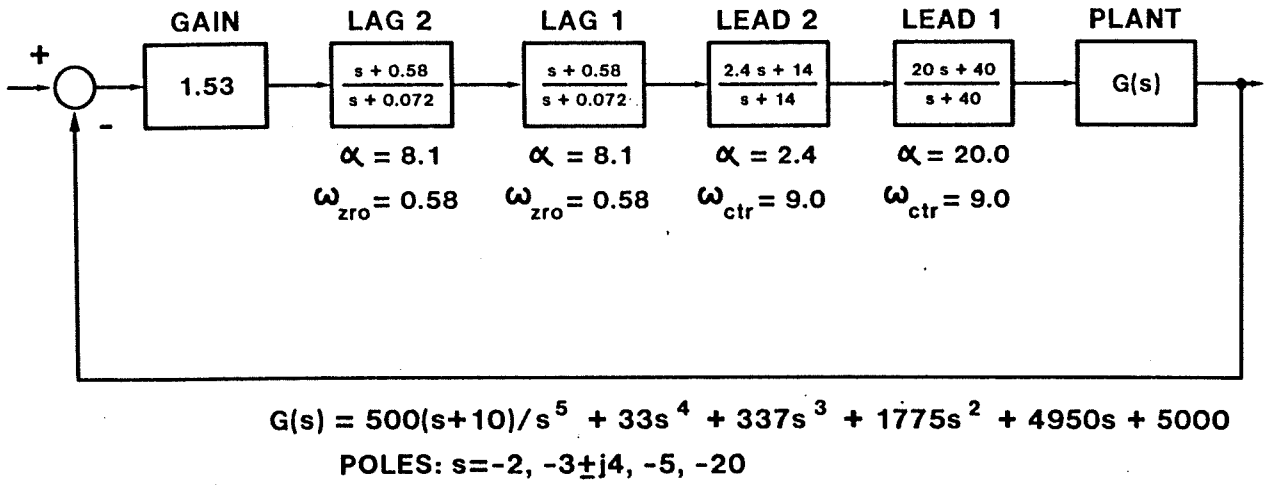

Figure 4. Block Diagram of the Compensated System

In this effort, we have shown how expert systems may be used to provide meaningful solutions to the problems identified above. However, there is unquestionably considerable detail that needs to be filled in and implemented before we have a complete "real system".

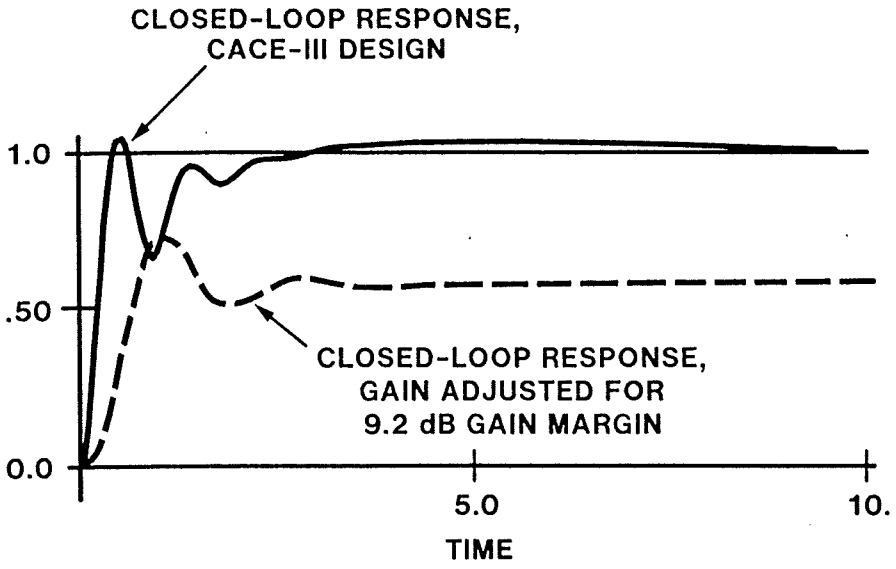

Figure 5. Closed Loop Time Responses

LESSONS AND UNFINISHED BUSINESS

There are many ways in which CACE-III is incomplete. We have come to realize that, in some senses, our present KBES represents only the tip of the iceberg. The following are several areas in which our present treatment is incomplete or totally lacking.

Support for Different User Capabilities

We have discussed our desire to support both the novice ("leading the user by the hand", if necessary) and the expert (without domination and boredom). To date, our primary focus has been on the support of the novice. Some ideas for providing an environment that would be more acceptable to the expert have been generated, including allowing the direct entry of facts (e.g., performance specifications) and allowing the user to invoke software to carry out other more arcane procedures. Neither of these ideas are easy to implement in a satisfactory way.

The direct entry of facts is simple, in the sense that providing the user the ability to input [MIN-DAMPING-RATIO VALUE **1.0**] +1 is clearly feasible. The

problem is that this, in effect, creates a new, very narrow command language that again requires careful attention to syntactic details. For example, [**MINIMUM-DAMPING-RATIO VALUE 1.0**] **+1** would not be a useful input unless the KBES can perform the pattern matching with enough intelligence to recognize that MIN and MINIMUM are synonymous. At worst, the KBES would accept the erroneous fact (which would be ignored, because of the pattern mis-match *vis-a-vis* the rule base); at best the KBES would issue an error message and the user would have to find the correct formulation and enter it again. Neither course of events would be satisfactory.

The only worthwhile solution is to incorporate a natural language interface, so that the user could enter a statement such as **the minimum acceptable damping ratio value is 1.0** and have the input interpreted by the KBES appropriately. The size of this task is directly proportional to the generality of the interface. It is certainly feasible to implement this capability for a well-defined task such as specification entry using a rudimentary grammar, for example; to permit the user to command the KBES in greater generality would be a major undertaking.

Allowing the expert to carry out his own procedures in the middle of a design session and then let the KBES take over is also very difficult to accomplish in general. The expert system must be able to monitor the activity, interpret it, and update the list of facts accordingly as the user works; if this is not done, then the user cannot return to utilize the KBES capabilities based on the work done manually. Therefore, incorporating this capability is limited by our ability to foresee what actions a user might take so that these mechanisms could be built into the KBES, and by the amount of work required for implementation.

To be feasible, this option can be permitted only at well-defined points in the process, and only for well-defined activity. One way to implement this functionality is to incorporate some of the KBES procedural knowledge in the form of "macros" that the expert user can modify and execute from within the KBES; this will ensure that the KBES can understand the expert's activity so that the knowledge base is updated appropriately and the KBES can build on the results.

The above paragraphs outline our ideas for supporting the expert user. Based on the difficulty we see in implementing them with any generality, it seems clear that KBES environments are in fact easier to develop for the novice than for the expert

user.

The Limitations of Scenarios

As mentioned above, the generation of scenarios (e.g., Taylor, Frederick, and James, 1984; Taylor, 1985a) played a large part in the development of CACE-III. The best way to acquire the knowledge that will be captured by any KBES is to carry out the given activity in full detail, with a realistic problem, using exactly the required tools. In our context, we selected a nonlinear plant model and first went through a "dry run" on paper, then developed rule bases to capture the process using software from the Federated System. At this point, we can carry out several of our initial scenarios in full.

There is one limitation or drawback to the use of scenarios: There is a tendency to think that a KBES that can carry out a scenario is a "real system". Often this is not true. In our case, there is often too much "branching", "groping", and "exploration of alternatives" involved in CACE to make a one-scenario KBES of general utility. In other words, an implementation that can only execute a single prescribed sequence of procedures would generally miss much of the judgment and selection of alternatives that has to be performed in the solution of real-life engineering design problems.

Making Software Usage Details Invisible

Eliminating much of the overhead in the use of conventional CACE software is clearly feasible, especially for the non-expert user who carries out procedures that have been anticipated and built into the system. The functions to be assumed by the KBES are invoking packages, executing algorithms, and error handling.

The knowledge of package capabilities and command sets and syntax can be incorporated readily into the KBES. Less-than-expert users would probably never have to issue a command in its native form; instead, the KBES would invoke packages and issue all commands based on the state of the problem solution and/or the wishes of the user. Implementing this capability may require a substantial amount of work, especially for a broad CACE environment such as the Federated System, but the effort is not conceptually demanding.

Exception handling is more difficult but still not a major problem, as long as the use of the software is limited to the activity foreseen by the KBES developers.

Again, there is a lot of attention to detail (starting with a catalog of possible error messages and recommended corrective action), but not much high-level reasoning involved.

There is an additional payoff to incorporating the knowledge of command sets, syntax, and exception handling into the KBES. If this knowledge is well organized, then the conventional software can be regarded as "interchangeable parts" in the sense that the architecture can be made highly modular, allowing the introduction of new packages that may extend the expert system's capabilities or improve its performance (through the use of numerically superior algorithms, for example).

Data-Base Management

The CACE data base has yet to be defined in complete detail; this step is a prerequisite to the rigorous implementation of DBM into a KBES such as CACE-III. The need for such a definition and preliminary ideas regarding its structure and contents have been presented by Maciejowski (1984) and LeVan (1984). Despite the incomplete definition of the data base, it is possible to consider the issue in general terms.

The fact that the CACE-III architecture is based on a conceptual model of CACE processes would seem to make some form of DBM a very natural and direct addition. We consider DBM functions in this context to include keeping track of all the files containing models (nonlinear and linear), the operating points and other conditions of validity of linearized models, analysis results, diagnostics, performance results, et cetera, along with the necessary relational information.

The CACE-III solution frame contains the list of facts in which the expert system keeps track of what has been done and what needs to be done, information required for decisions about the selection of analysis and design procedures and tradeoff analysis, and a log of the entire transaction; at this point in our work, we do not see a need for any DBM information that is not available in this knowledge base. The procedural and relational knowledge of the operation of the software is thus readily available so that the KBES can perform DBM activities itself (keeping track of the data base and writing DBM information into a master file) or invoke a conventional DBM system to carry it out. One use of such a master data-base management file is illustrated in Taylor (1985a). As mentioned previously in the context of software usage, DBM along these lines entails straightforward work that is not

conceptually taxing. A rule base must be developed which "monitors" the activity of the KBES, and, whenever a procedure is performed which creates new information, ensures that the data are written to the appropriate file and/or the file is appropriately named and cataloged with the required relational information.

RELATED WORK

To the best of our knowledge, the first in-depth consideration of the application of KBES technology to CACE was by Taylor, Frederick, and MacFarlane (1983). That presentation was the genesis of our CACE-III project. Since that time, there have been a number of projects related to this concept:

- Gomez *et al* (1984) describes a prototype expert system for the treatment of stochastic control and nonlinear filtering problems. The program is written MACSYMA, LISP, and PROLOG, and accepts user input in natural language and symbolic form. It carries out the basic analysis of the user's problem in symbolic form and produces FORTRAN code implementing the controller or filter algorithm. PROLOG is used to check the well-posedness of the user's problem.

- LeVan (1984) considers the data representation requirements for CACE using frames and slots. The concept of frame is extended to provide slots for default/required information, procedures, nested frames, and ports for the input and output of CACE process "flows". The system modeling approach incorporated in this concept is the bond-graph methodology.

- Birdwell *et al* (1985) describes the overall concept of a KBES for CACE and a prototype system that implements these ideas for modern control and estimation problems (linear quadratic regulator design, Kalman-Bucy filters).

CONCLUSIONS

We believe that the research outlined in this paper has reached the point where the promise of KBES technology to solve many of the problems identified in existing conventional software can be seen to be real. However, the amount of effort involved in realizing such a higher-level environment in a useful form should not be underestimated, and the developer must be careful not to raise unrealistic expectations.

Research in this area is still in its infancy. The next few years should bring about a number of useful systems based on the ideas and work presented here and in the references.

Acknowledgements. Some of the material in the sections titled **EXPERT SYSTEM CONSIDERATIONS** and **AN EXPERT SYSTEM ENVIRONMENT** is taken quite directly from Taylor and Frederick (1984); the section **A Paradigm for CAE** is closely patterned on a section of GE internal report 84CRD127 by Taylor and Frederick. Special mention must be made of the work of John R. James, who, in the course of his PhD research at Rensselaer Polytechnic Institute, did much to bring CACE-III to reality.

REFERENCES

Birdwell, J. D., J. R. B. Cockett, R. Heller, R. W. Rochelle, A. J. Laub, M. Athans, L. Hatfield (1985). Expert systems techniques in a computer-based analysis and design environment. *Proc. Third IFAC Symposium on CAD in Control and Engineering Systems*, Lyngby, Denmark.

Coombs, M. J. (1984). *Developments in Expert Systems.* Academic Press, New York.

Davis, R. and D. B. Lenat (1982). *Knowledge-Based Systems in Artificial Intelligence.* McGraw-Hill, New York.

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

Elmqvist, H. (1977). SIMNON - An interactive simulation program for nonlinear systems. *Proc. Simulation '77*, Montreux, France.

Gomez, C., J. P. Quadrat, A. Sulem, G. L. Blankenship, P. Kumar, A. LaVigna, D. C. MacEnany, K. Paul, and I. Yan (1984). An expert system for control and signal processing with automatic FORTRAN code generation. *Proc. 23rd Conf. on Decision and Control*, Las Vegas, NV.

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

IEEE (1984). *Workshop on Principles of Knowledge-Based Systems.* IEEE Computer Society, IEEE Cat. No. 84CH2104-8.

James, J. R., P. P. Bonissone, D. K. Frederick, J. H. Taylor (1985). A retrospective view of CACE-III: considerations in coordinating symbolic and numeric computations in a rule-based expert system. *Proc. Second Conf. on Artificial Intelligence Applications*, Miami Beach, Florida.

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. *Proc. Control 85*, Cambridge, UK.

James, J. R., J. H. Taylor, and D. K. Frederick (1985). An expert system architecture for coping with complexity in computer-aided control engineering. *Proc. Third IFAC Symposium on CAD in Control and Engineering Systems*, Lyngby, Denmark.

LeVan, O. (1984). An expert system for control system design. *Proc. 18th Asilomar Conf. on Circuits, Systems, and Computers*, Asilomar, CA.

Maciejowski, J. M. (1984). Data structures for control system design. *Proc. EUROCON '84*, Brighton, UK.

Michie, D. (1982). *Introductory Readings in Expert Systems.* Gordon and Breach, London.

Moler, C. (1982). MATLAB users' guide. University of New Mexico Department of Computer Science Technical Report CS81-1 (Revised).

Spang, H. A. (1982). The federated computer-aided control design system. *Proc. Second IFAC Symposium: CAD of Multivariable Technological Systems*, Purdue, West Lafayette, IN, 121-129.

Spang, H. A. (1984). The federated computer-aided control design system. *Proc. of the IEEE*, **72**, 1724-1731.

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

Taylor, J. H., D. K. Frederick, and A. G. J. MacFarlane (1983). A second-generation (sic) software plan for computer-aided control system design. *Abstr. of IEEE Symposium on Computer-Aided Control System Design*, MIT, Cambridge, MA.

Taylor, J. H., D. K. Frederick, and J. R. James (1984). An expert system scenario for computer-aided control engineering. *Proc. American Control Conf.*, 120-128, San Diego, CA.

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

Taylor, J. H. (1985a). An expert system for integrated aircraft/engine controls design. *Proc. National Aerospace and Electronics Conf. (NAECON)*, Dayton, O, 661-669.

Taylor, J. H. (1985b). Computer-aided control engineering environment for nonlinear system analysis and design. *Proc. Third IFAC Symposium on CAD in Control and*

Engineering Systems, Lyngby, Denmark.

Wieslander, J. and I. Gustavsson (1976). IDPAC - an efficient interactive identification program. *Proc. 4th IFAC Symposium on Identification and System Parameter Estimation, Tbilisi, USSR.*