An Expert System Architecture for Computer-Aided Control Engineering

JAMES H TAYLOR, MEMBER, IEEE, AND DEAN K FREDERICK, MEMBER, IEEE

Invited Paper

We propose the development of a rule-based expert system to create a third-generation man/machine environment for computer-aided control engineering (CACE). The breadth of the CACE problem is of particular concern, and provides a major motivation for the use of artificial intelligence. This approach promises to provide a high-level design environment that is powerful, supportive, flexible, broad in scope, and readily accessible to non-expert users.

We focus primarily on the high-level requirements for an improved CACE environment, and on the expert system concepts and structures that we have conceived to fulfill these needs. Our chief goal is to determine what artificial intelligence has to contribute to such an environment, and to provide as definite and credible a vision of an expert system for CACE as possible. The main product of this effort is an expert system architecture for CACE.

I. INTRODUCTION

To begin, we will develop the motivation for an advanced computer-aided control engineering (CACE) environment. Primarily, this will involve defining the problem, surveying the status of existing CACE software, and outlining what we perceive to be its significant shortcomings. To a great extent, these deficiencies are caused by the very broad scope of CACE. The rest of the presentation will address an approach to solving the deficiencies that we have identified.

To establish a context for discussing CACE environments, we must first outline our definition of the "control engineering problem." We have enumerated in Table 1 a set of substantial tasks that we currently include in the scope of control engineering. We are using the broad terminology CACE rather than the more standard term computer-aided control system design (CACSD) in order to highlight the wide range of the problem under consideration.

Having defined the problem in broad terms, we can identify two phases of development or "generations" of software for CACE. First-generation software, developed in the 1960s and early 1970s, generally consisted of one or a few single-purpose routines for the support of analysis and design. Examples would be a program for generating Bode, Nichols, or Nyquist plots for user-specified transfer function models, a root locus routine, or software for finding the eigenvalues of a system dynamic matrix. Second-generation CACE software we will define to be comprised of packages of more than a few routines, integrated into a more easily used environment with a single database, and encompassing a good portion of the analysis and design process. A pioneering development in this phase is the UMIST suite [1].

According to these definitions, the development of second-generation software for CACE is well advanced at this time. Many powerful control system analysis and design packages have been created; see, for example, [2], [3] for descriptions of CACE software and [4], [5] for an indication of the present state of the art. The package with the broadest scope at this juncture is the General Electric Federated System [6]; in fact, it is a "super-package" comprised of four software systems for identification, simulation, frequency-domain design, and state-space design, with a unified database and command-driven environment. Generally speaking, creating software packages of this generation has reached such a state that we believe that the field can be considered to be quite mature.

Table 1 Control Engineering Activities

1	Modeling the Plant to be Controlled
2	Determining the Characteristics of the Plant Model
3	Modifying the Configuration to Make the Plant More Amenable to Control (e.g., Moving or Adding Actuators or Sensors)
4	Formulating the Elements of the Design Problem
5	Checking to See that the Design Problem Is Well-Posed; Especially, Determining the Realism of Specifications in Light of the Given Plant Model and Design Constraints
6	Executing Appropriate Design Procedures
7	Performing Design Tradeoffs, if Necessary
8	Validating the Design
9	Providing Complete Documentation of the Final Design
10	Implementing the Final Design

There remains a great deal to be done, however, in terms of providing a *powerful, general environment* for the *less-than-expert user*. By the term "general," we mean a CACE environment that includes the full gamut of control system design activities listed in Table 1. There are several factors involved in the phrase "less-than-expert user": such a user may not have had the experience required to synthesize control theory into an effective design approach, may not

Manuscript received March 30, 1984; revised June 20, 1984

J. H. Taylor is with General Electric Corporate Research and Development, Schenectady, NY 12345, USA.

D. K. Frederick is with Rensselaer Polytechnic Institute, Troy, NY 12181, USA.

0018-9219/84/1200-1795$01.00 ©1984 IEEE

be aware of recent developments in the field, and/or may not be a frequent practitioner of control system design or user of CACE software. In any case, most less-than-expert users find that second-generation CACE software is difficult to use effectively. Specifically, the problems are:

1. Most CACE packages are specialized to one aspect of the overall design process, to one "domain" (time or frequency), and often to one particular synthesis or design approach. Therefore, many designers will not be able or willing to restrict themselves to the use of a single package. Software for nonlinear simulation, linearization, linear analysis, and linear controller design is required as an absolute minimum for a realistic approach to CACE [7]; even these capabilities are rarely available in one package.
2. The problem-solving environment provided by most current packages is generally potent but rather low-level. This consideration may even be compounded by the first point, in the sense that a user may need to know several hundred commands, perhaps in several syntaxes, in order to be able to utilize a comprehensive package or collection of packages effectively.
3. Most packages provide little or no guidance. The user must formulate a meaningful model of the plant, pose the design problem appropriately, know exactly what procedures to execute, and have the experience and practical feel for the complete control engineering problem outlined above to be able to judge the meaningfulness of the results obtained at each step and know how to proceed.
4. Most packages provide little or no useful documentation of the design process. The user must keep a record of the design procedure and of tradeoff studies and other considerations involved in arriving at the final controller design.
5. Most packages provide little or no useful support in validating and implementing the final design. It is up to the user to recognize the need for validation and to know how to accomplish that vital step, and it is up to the user to decide how to realize the theoretical controller design in hardware.

These considerations may not concern the expert, who can formulate a well-posed problem with facility, who is well versed in multivariable control system design, and who is fluent in the use of available control design packages. However, for many real-world control system design problems the knowledge, comprehension, and attention to detail required to keep on top of the design process in existing CACE environments will surely tax many users. The end result of the deficiencies itemized above is generally frustration, "wheel spinning," and often an unwillingness to expend the effort required to become a knowledgeable software user or to maintain this skill.

We believe that the problems outlined above are serious in light of the current state of the art in control theory and CACE. In fact, there is good reason to believe that this situation will worsen, as a result of the following trends:

1. New control system design approaches are constantly being added to the control engineer's repertoire, and software for their application is being developed.
2. More of the control engineering problem (Table 1) is being carried out in CACE software environments.

In support of the first factor, we can point to recent developments in unifying classical and modern design [8], in the algebraic function theoretic control design approach [9], in the rational stable factorization method [10], and in conceiving promising general approaches for the design of controllers for nonlinear systems [11]–[12]—and this list is not exhaustive by any means. While these results are obviously welcome, they will inevitably complicate CACE for the reasons mentioned above. The nonlinear systems research will particularly strengthen the need for a better control system design environment, because the design process for nonlinear controls is substantially more complicated than that for the linear case [7].

The second point in our case for a growing need for a better CACE environment again refers to the increasing scope of the "control engineering problem." The fact that more aspects of the problem are now being included in the realm of CACE is readily apparent in the literature [4], [5]. The most recent major extension is in the area of software development for controller implementation [13]–[15]. In comparison with this aspect of the control design problem, many see conventional software for controller design as treating only the "tip of the iceberg." In addition, software for model development [16], [17] should be considered to be an integral part of CACE, not merely a precursor. This activity may also be a larger and more important effort than the narrow problem of designing a compensator for a given plant model. Adding software for computer-aided controller implementation and for modeling support will provide a strong impetus to the development of better CACE environments.

For the above reasons, we believe that the need for an improved CACE environment is quite clear. In particular, we have in mind an environment that treats the "whole problem," that allows the user to work at a high level, that provides guidance and support to the less-than-expert user, and that provides access to a wide variety of analysis and design methods without the necessity of mastering a large ensemble of low-level commands. This need has led us to consider the use of a rule-based expert system as a way to address the deficiencies identified in the preceeding discussion. We believe that the resulting software concept goes well beyond the second-generation phase of CACE mentioned previously; for this reason, we consider it to be a third-generation environment which we thus refer to as CACE-III. It is based on a "model" of a human expert's approach to engineering design, which proceeds systematically through the steps outlined in Table 1. Because this model and approach are general, the resulting architecture should have generic value in the area of developing expert systems for engineering design [18].

In the remainder of this paper, we discuss an expert system approach to CACE, present the architecture of a third-generation environment that is currently under development, and outline (in functional terms) substantial portions of the rule base. We also treat some essential features of a rule-based expert system for engineering design. We then close with a brief status report, summary, observations, and conclusions.

II An Expert System Approach to CACE

Expert or knowledge-based systems are software environments designed to aid in solving problems that require high levels of expertise, "reasoning" (inference), and heuristics. Such problems are generally complicated and broad in scope, and are not amenable to clear-cut well-posed solutions. Control system design, in the broad sense outlined in Table 1, is such a task. For a more detailed discussion on the subject of expert systems, or for additional background reading in the area, refer to [19], [20]. Alternatively, one may refer to [21] for a brief overview of expert systems and a more extensive bibliography than that provided here. A short presentation of selected rule-based expert system concepts, to illustrate how one can endow software with expertise and inference capability and to discuss relevant environmental issues, is given in Section IV.

One way to visualize CACE-III is to describe the resulting environment as a "shell" for existing conventional CACE software. This shell would support control system design by performing or helping to perform the functions listed in Table 1. The real key to this environment is that it should create an agreeable interface for the less-than-expert and non-everyday user.

Based on Table 1, it is clear that the major prerequisites for this development are:

1. an overall approach to CACE, in high-level conceptual terms,
2. a comprehensive collection of control system analysis and design methodologies,
3. high-quality conventional CACE software, and
4. a clear understanding of the use and limitations of this approach and software.

The overall approach to CACE outlined in this paper is our enunciation of a rather traditional systematic plan of attack applied to the problem defined in Table 1. This will be developed in Section III as a basis for the CACE-III architecture. The collection of design methodologies may be characterized as *algorithmic formulations of classical and modern control system design approaches*. The knowledge or rule base of CACE-III is thus comprised of this systematic plan of attack, plus rules that embody "good control engineering practice" and the heuristics ("rules of thumb") used by expert control engineers in the complete control engineering effort, plus knowledge of the proper selection and execution of the appropriate analysis and design methodologies.

Our concept for an improved environment for CACE is thus a synthesis of classical and modern control theory and artificial intelligence. Therefore, it may be pertinent to ask: what exactly does the expert system approach have to offer in the context of CACE? We have alluded to some of the considerations that have led us to use this approach, but we would like to detail the more important factors in our case in more explicit terms.

The main issue in this context is that it is difficult to add expertise and support features to conventional software, *especially where judgment, heuristics, and inference capabilities are involved*. A high-level environment for design must incorporate such knowledge, in recognition of the fact that engineering design is not just "number crunching." These issues are central to the problems identified in Section I.

Other issues are more subjective and perhaps open to debate, because they depend on the expert system approach and environment, which is not standardized at this time [19]–[22]. However, it is our experience that a rule-based expert system can be endowed with greater flexibility than conventional software, because of its knowledge base, inference capability, and more natural interaction with the user. Also, it appears that expert systems are inherently able to deal effectively with broad-scope procedures such as CACE, while conventional software tends to become unwieldy. Finally, an expert system is easier to expand than a conventional program, in the sense that the mechanics of adding rules that embody "new expertise" is straightforward, especially if the knowledge is "modular." Some environmental issues that have a bearing on these considerations are discussed in Section IV-D.

In summary, we enumerate the following areas of CACE where we believe artificial intelligence (AI) can provide useful contributions: the types of expertise that are required for CACE are diagnosing the plant model, setting up a realistic design problem, selecting appropriate design methods, performing tradeoffs, validating the design, implementing the controller, and using conventional CACE software. (The use of a synthesis technique may lessen the requirement for judgment and inference somewhat; in true design, however, these capabilities generally play a very substantial role.) Heuristics are certainly a major factor in a human expert's ability to formulate a well-posed design problem. Reasoning capability will prove to be highly advantageous for directing and keeping track of the design process as it progresses. Since these considerations are at the heart of many of the problems we identified in the context of existing CACE software (Section I), we believe that there is ample motivation for considering developing a rule-based expert system for CACE.

III An Expert System Architecture for CACE

We will develop the structure of CACE-III by considering a fundamental question: how might an expert control engineer accomplish the tasks listed in Table 1? We find it convenient to address this question in two stages, involving concepts we call the *problem frame* and the *solution frame*.

A. The Problem Frame

A central consideration in the control system design process is obtaining a *complete, well-posed problem formulation*. This may be represented by a "list of facts" or, in artificial intelligence (AI) terminology, "frame." The information in this frame may be organized or partitioned into three categories, which can be illustrated by posing the following questions:

1. *Plant Characteristics*: Is the plant that is to be controlled linear or nonlinear? How nonlinear is it? Can it be linearized? Is it stable or unstable? Are there any right-half-plane zeros? Are there resonances? Is the plant controllable and observable?
2. *Constraints*: What are the limitations with respect to controller implementation (analog or digital), complex-

ity (order), structure (e g , decentralization), data rates or sampling time (if digital)?

3. *Specifications*: How must the control system behave or perform? Common indications or measures of performance include rise time, bandwidth, percent overshoot, pole position, gain margin, phase margin, quadratic performance indices, sensitivity, and robustness

The answers to these and similar questions make up the knowledge required for the problem frame partitioning portrayed in Fig 1

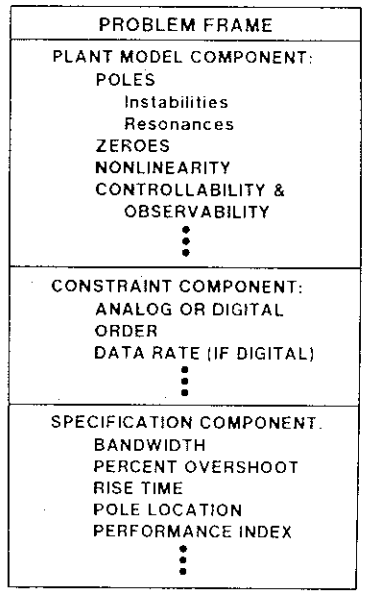

Fig. 1. Partitioning of the problem frame

Such a frame is a major focal point for CACE, as can be observed from the fact that the first five functions in Table 1 relate to this information Therefore, developing the list of facts that comprises the problem frame must be a central activity of CACE-III This is accomplished via the interaction of the user and the expert system

B. An Expert System Structure for Problem Frame Development

The above considerations lead directly to an architecture for an expert system to support the process of problem formulation Such a structure is depicted in Fig 2 The two rule bases (RBs) shown there serve the following purposes:

1 RB1 governs interactions among the *design engineer*, *plant models* (both nonlinear and linear), and the *model and constraint components of the problem frame*. This rule base provides support in model development and supplies the facts required to characterize the plant model and design constraints in the problem frame Model development support might include: direction in the use of identification procedures [23], suggesting modifications to correct deficiencies such as a lack of controllability and/or observability, and making recommendations concerning the numerical tractability of the model These functions are a major part of the "diagnostic" capability of CACE-III An example of model descriptor "facts" in the problem frame is provided in Section IV (Table 3)

2 RB2 governs interactions between the *design engineer* and the *specification component of the problem frame*. These rules guide the user in entering design specifications and checks those specifications for consistency, completeness, and workability The rules governing realism are based mainly on determining what is achievable for the closed-loop system given the data in the model and constraint components of the problem frame; therefore, the user will not be allowed to specify without having the latter information in the list of facts

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

C. The Solution Frame

In the second part of the execution of a control engineering problem, we see the human expert developing and working in a parallel construct, which in CACE-III we call the solution frame We implement this in the form of a second list of facts which serves as a "scratch pad" where the expert system keeps track of what has been done and what needs to be done, and where information required for decisions about the selection of design procedures and tradeoff analysis resides A sketch of this part of the knowledge base is provided in Fig 3 The relative lack of detail (compared to the problem frame) is due in large part to the fact that a set of automated design procedures has not yet been established

D. The Complete Expert System Architecture

According to our scenario, the human expert control system designer continues the design process by completing the remaining tasks outlined in Table 1, working in an information framework analogous to the solution frame A complete expert system for CACE must mechanize these functions, as well as those shown in Fig 2, and must know how to use conventional CACE software packages to perform the appropriate tasks

The line of thought provided by the above summary of the activities of a human expert has given rise to a complete functional structure of CACE-III that is depicted in Fig 4 In particular, we have created a construct in which the rule base is partitioned into six parts, as shown The functions of rule bases RB3 through RB6 may be outlined as follows:

1 RB3 governs interactions between the *problem frame* and the *solution frame* These rules deal with specifications, constraints, and plant characteristics, and set up or modify the list of facts in the solution frame describing what needs to be done to achieve design goals An example of the role of this rule base is as follows: given a steady-state error specification, RB3 would initiate the evaluation of the low-frequency gain $G(0)$ of the linearized plant model, and determine whether or not it is necessary to increase the low-frequency open-loop gain.

2 RB4 governs interactions between the *solution frame* and the available *design procedures*. For example, if RB3 determined that the low-frequency open-loop

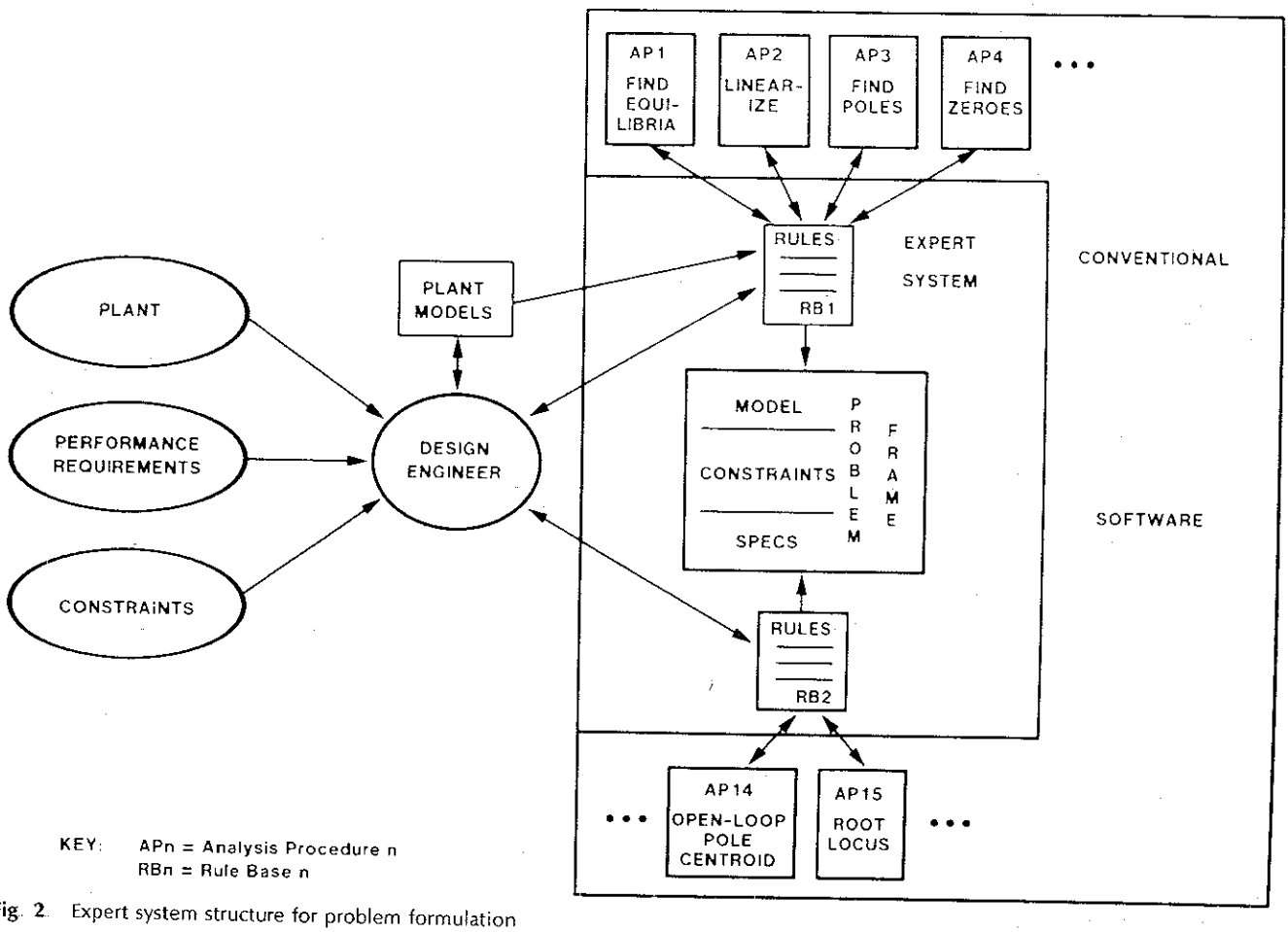

Fig. 2. Expert system structure for problem formulation

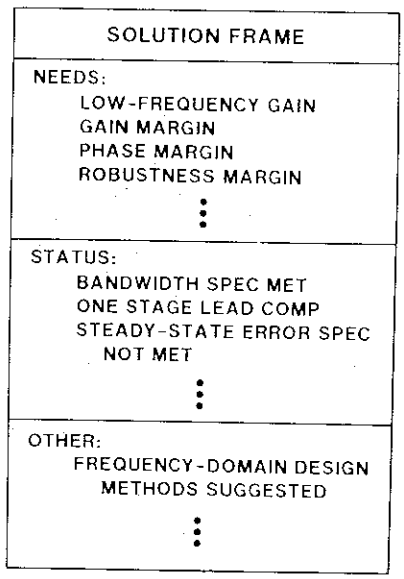

Fig. 3 Partitioning of the solution frame

gain needs to be raised, then RB4 may invoke a design procedure to increase the static gain in the forward path or to design a lag compensator, as appropriate

3. RB5 governs interactions between *design procedures* and the *problem and solution frames*. Every time a design procedure is executed, the solution frame must be updated to reflect the corresponding addition/change in the open-loop transfer function (e.g., additional gain or dynamic compensator block). This rule base will also supervise tradeoff analysis by selecting specifications to be relaxed and modifying the problem frame appropriately if the original specifications cannot be met
4. RB6 governs the final control system *validation process* and conversion from idealized controller design to *practical implementation*. Validation would consist primarily of analysis and simulation to ensure that the design requirements had been achieved. In performing this step, the most realistic model of the plant would be combined with a detailed representation of the controller, and the performance of the system determined. Implementation, in the present microprocessor age, generally means that the ultimate output of CACE-III will be a code to be downloaded to a read-only memory (ROM) that will provide the actual controller mechanization. Software to permit this final step is currently under development (cf. [13]–[15])

RB3 serves primarily to "initialize" the list of facts that makes up the solution frame, or to modify that list if performance tradeoffs must be considered. Observe that RB4 and RB5 represent an iterative "loop"; these rule bases are cycled through until all specifications are satisfied or until it has been determined that the specifications cannot be met. The former rule base decides what needs to be done, while RB5 keeps an account of the status of the

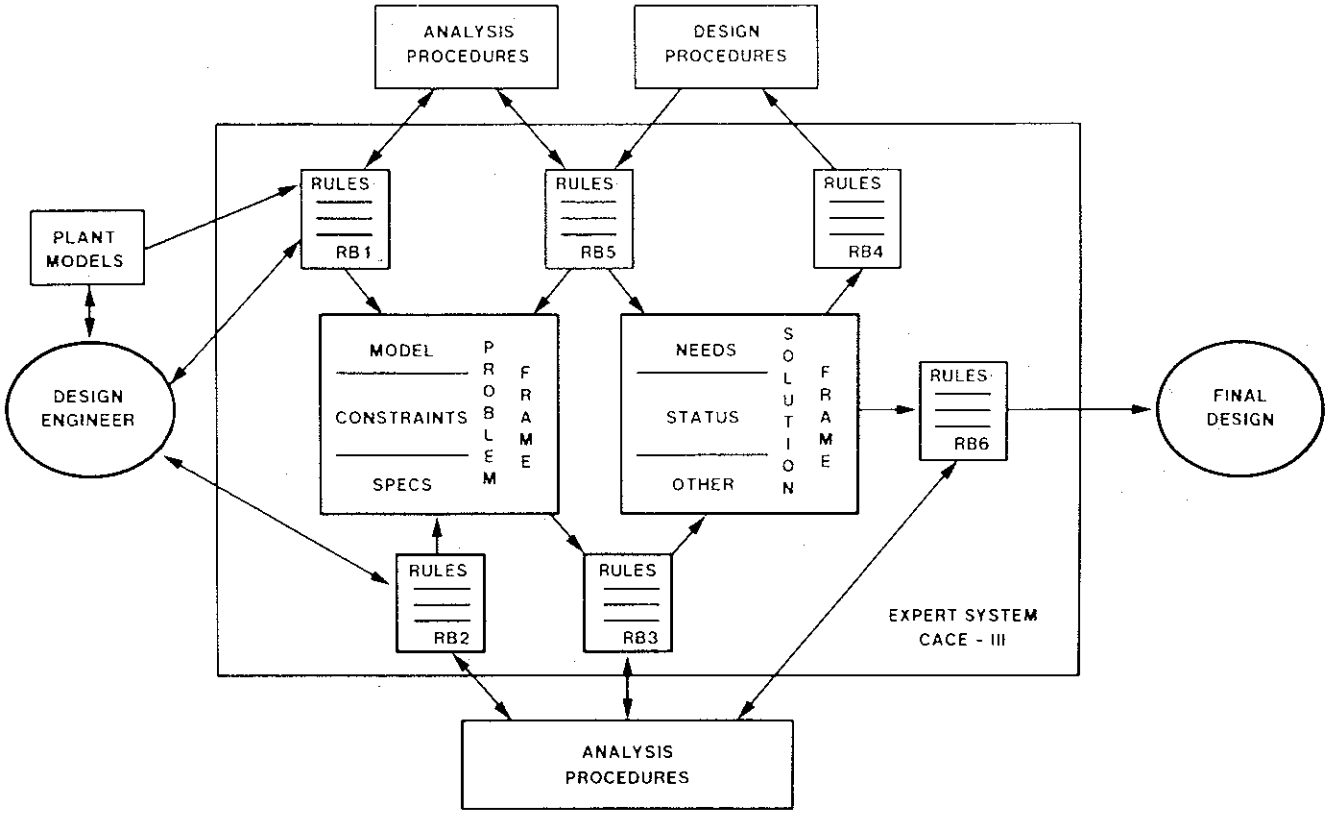

Fig 4. Complete functional structure of CACE-III

solution as it is being obtained. This latter rule base may cause RB2 and RB3 to modify the problem frame if it is necessary to relax specifications or perform tradeoffs.

In the context of developing RB4 and RB5, we must observe that most design procedures are not readily available in algorithmic form. To some extent, this may be due to the difficulty of reducing engineering judgment to algorithms; to some extent this may be due to the fact that control theorists have left this step up to engineers in industry. It is true that industrial practitioners have obviously succeeded in bringing theory to practice—but their specific approaches or "algorithms" are undocumented and thus this expertise is difficult to access. We have explored this issue in some detail, by developing a rule base that mechanizes a classical control design approach to synthesizing a lead/lag compensator for a single-input single-output plant [24]. We selected this problem area because the classical control design approaches are the most "intuitive" and least well-codified compared with more recent developments. We will not describe this work in any detail here; suffice it to say this effort is neither simple nor inordinately difficult.

Finally, note that all of these rule bases will be required to invoke existing conventional CACE software for many of the indicated analysis and design functions. Expertise regarding the use of this software will be incorporated as an integral part of CACE-III.

The rule base structure depicted in Fig. 4 serves two important functions: first, it clarifies many conceptual aspects of the design process, and thus provides a basis for rule base development. In addition, this structure can 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.

IV Expert System Features

We are currently undertaking the creation of a demonstration or prototype version of the CACE-III environment using DELTA, a rule interpreter or "inference engine" developed at GE for a diesel-electric locomotive troubleshooting aid [25]. While space does not permit a detailed discussion, we will outline some aspects of the operation of DELTA, to the extent required for tutorial purposes. Our goal is to convey the spirit of the expert system approach, not a working knowledge of DELTA. 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.

A. Rules

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

$$\text{IF } [\text{Premise}_1\ \text{Premise}_2 \cdots]$$
$$\text{THEN } [\text{Conclusion}_1\ \text{Conclusion}_2 \cdots]$$

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;
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 of a three-tuple 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. Most of these ideas are illustrated in the two sample rules provided in Table 2, which are taken from the rule base RB2 that governs the specification entry process.

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."

B. Inference Mechanisms

DELTA performs inference functions 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 a 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.

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 launched, by asking

a Is Premise_i known to be true (in light of the existing list of facts)?
b Can Premise_i be inferred from the current list of facts (using another rule or several rules)?
c. Can conventional analysis and design procedures be used to determine if Premise_i is true?
d 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.

Table 2 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)
```

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.

C. 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.

Table 3 The Problem Frame after a DIAGNOSE and SPECIFY Session

Fact	Basis	
PLANT MODEL NONLINEAR	User-asked	RB1
PLANT-NL-MODEL FNAME EXOREACT	User-asked	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-asked	RB2
CONTROLLER TIME-TYPE CONTINUOUS	User-asked	RB2
CONTROLLER STRUCTURE DIAG-DOMINANT	User-asked	RB2
CONTROLLER CHANNEL1-IN U1	User-asked	RB2
CONTROLLER CHANNEL1-OUT Y1	User-asked	RB2
CONTROLLER CHANNEL2-IN U2	Inferred	RB2
CONTROLLER CHANNEL2-OUT Y2	Inferred	RB2
MAX-STEP-SS-ERR CH1-VALUE 0.25	User-asked	RB2
MAX-REAL-PART CH1-VALUE −1 4	User-asked	RB2
MIN-DAMPING-RATIO CH1-VALUE 1.0	User-asked	RB2
MAX-STEP-SS-ERR CH2-VALUE 0.5	User-asked	RB2
MAX-REAL-PART CH2-VALUE −1 4	User-asked	RB2
MIN-DAMPING-RATIO CH2-VALUE 1.0	User-asked	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

Such a list is illustrated in Table 3, 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 3, 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, It should be observed that the underlying *data*, e g , the numerical results of the nonlinear system diagnosis and the linearized model and its diagnosis, are contained in ancillary files. The expert system 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

D 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 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 [19]–[22] The three most important capabilities we have so far identified are the ability to run external processes, the ability to perform numerical computations and tests, and the ability to be controlled or "steered" by the user

In order for CACE-III to perform the analyses required for diagnosis and design, it is necessary for the inference engine to run external processes that execute tasks such as calculating the frequency response of the plant and determining gain margin and bandwidth. CACE-III must initiate the process with the appropriate input parameters, and must be able to read the results of the analysis in order to update the list of facts. The DELTA inference engine we have used in our research so far does not have this capability; a subsequent version of DELTA will be able to perform this function

The earlier 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 [25]), but in our case it has required the use of numerous additional rules and menus for the entry of data in literal form. To make full use of its potential, DELTA is being extended to allow the use of integers and reals, performing tests for equality, inequality, and inclusion in a range, and executing arithmetic procedures.

The third 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 CACE-III 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) The power to guide the design process can be provided by implementing the capability for the "automatic," "semi-automatic," and "manual" operation of CACE-III.

We have identified the following mechanisms for allowing the designer to interact meaningfully 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 [26]) may be much less desirable.
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; the other ideas (especially items 3 and 4) are specifically targeted for engineering users and may not be as commonplace.

E. Rule Base Writing Support

In developing rules for use in DELTA, we have come to see an urgent need for an environment analogous to the "debug" environment associated with high-level languages such as Fortran. The following capabilities are essential for debugging a rule base effectively:

1. setting break points so that the status of the process can be determined,
2. displaying the complete list of facts at any time in the course of a session,
3. displaying changes in the list of facts as a single rule is being executed or between two breakpoints,
4. tracing the execution of rules, and
5. modifying facts and resuming operation.

A great deal of effort is currently being focused on creating a supportive environment for developing rule bases, at GE and elsewhere (refer to [22], for example).

V. Status

We have thus far developed the following:

1. an architecture for CACE-III, including a specific, detailed outline of the functional characteristics of the expert system and of the rule base;
2. a detailed "transaction" or dialog between CACE-III and a user [18], [26], to provide a more tangible "bottom-up" basis for detailing the functions of the various rule bases shown in Fig. 4;
3. working rule bases for several functions, including an automatic control system design procedure (lead-lag compensator design for a single-input/single-output plant [24]) and some portions of specification development assistance and plant model diagnosis; and
4. analysis routines to implement several nonlinear system diagnostic functions that illustrate the capabilities of RB1 [18], [26].

Only item 1 has been discussed in any detail in this paper. The remaining items were developed primarily as exploratory tools, to sharpen the focus of the concept we have presented here, to determine what artificial intelligence has to offer in the context of CACE, and to ascertain the credibility of that promise.

VI. Summary, Observations and Conclusions

A. Summary

The project we have described in this paper has just emerged from the concept development phase. Basically, the primary outcome of our effort can be considered to be a high-level design specification for a third-generation CACE environment. There are several areas in which we believe progress has been made:

1. elucidating the "CACE problem" (starting with [7] and following through Section I) and identifying an improved environment for dealing with it based on rule-based expert systems;
2. developing a detailed architecture, based on lists of facts (problem and solution frames), rule bases, inference mechanisms, and conventional analysis and design software, that can support the user in the core activities of CACE outlined in Table 1;
3. developing a concept that is flexible enough that it can fulfill the needs and desires of novice and expert users alike; and
4. determining some requirements of an inference engine that would be suitable for implementing CACE-III.

At this point, we have created a tangible top-level description of an expert system for CACE plus a small amount of the underlying rule base and detail [18], [24]. We have placed great emphasis on the issues that translate into *credibility*, namely, the capabilities of expert systems and their relevance to CACE; there is a great deal to be defined and implemented before we have even a prototype "real system." We hope that the concepts and issues set forth here will stimulate further discussion and development.

B. Observations

There are a number of observations (opinions) we have reached in the course of this study:

1. Few (if any) existing CACE packages cover the full scope of the control system design problem and support the range of techniques that most users would like to have available.
2. Critical areas of user support that are not meaningfully provided by existing CACE software environments are:

developing and diagnosing plant models, developing meaningful specifications, selecting design approaches, performing tradeoff studies, validating the design, and implementing it.

3. Most existing broad-scope CACE software is difficult for the less-than-expert and/or infrequent user to master.
4. An expert system for CACE should be conceived with the following primary goals:
   a. removing as much drudgery as possible from the design process,
   b. reducing the probability of error,
   c. allowing the less-than-expert user to obtain better designs than otherwise possible,
   d. adding better discipline and documentation to the design process, and
   e. enhancing rather than replacing engineering skill and judgment.
5. An expert system for CACE should benefit an engineer in another, higher level way, by instilling an understanding and working knowledge of new control system design methods ("technology transfer"). This is only possible if the user can take an active role in the design process, as outlined in Section IV-D.
6. An expert system for CACE should prove to be helpful in engineering education, by elucidating the process to students and allowing them to develop understanding and skill in applying what they are learning.

C. Conclusions

There are a number of conclusions we have reached in the course of this study that directly pertain to CACE:

1. The application of expert system concepts would seem to be a natural choice for providing the inference capabilities, judgment, and heuristics needed for the critical areas of user support we have identified.
2. Expert system concepts show great promise in the development of CACE environments that are flexible enough to meet the needs of a very broad spectrum of control engineers—from novice to expert designers, from infrequent to everyday users.
3. No foreseeable expert system is going to provide the *optimal* solution to *every* control design problem; the case for creating such an environment must be made on the basis the "primary goals" outlined in observation 4 in Section VI-B.
4. Most available design methods are not going to be easy to automate, because they have not been formulated in algorithmic terms.

We have also reached two conclusions which we believe have implications that go well beyond the field of CACE, namely:

1. Expert/knowledge-based systems show great potential for providing the basis for a vastly improved man/machine environment *for engineering design in general*.
2. The "systems approach" (by which we mean the train of thought outlined in the body of this paper, wherein the human expert's design process is modeled and reduced to a structure that can be implemented in a rule base) *provides a powerful methodology for creating the architecture of an expert system*.

ACKNOWLEDGMENT

The research outlined in this paper owes a great deal to the suggestions and efforts of a number of people aside from the authors. The idea of applying artificial intelligence to "the total CACE problem" was an outgrowth of discussions between Taylor and Professor A. G. J. MacFarlane of Cambridge University in the fall of 1982. A major factor in the resulting research was the expertise and expert system software provided by Dr. P. Bonissone of the General Electric Corporate R & D Knowledge Based Systems Program. Other substantial contributions were made by students from Rensselaer Polytechnic Institute, as follows: N. Lassinger modified existing CACE software to make it possible to mechanize an automatic lag/lead compensator design procedure; J. James refined and completed an algorithmic statement of this procedure [24], developed the rule base required for its implementation, and made helpful suggestions for this paper; and R. Quan helped to develop nonlinear system diagnostic software (for the analysis of harmonics and distortion). The ideas presented in this paper have profited substantially from these interactions.

REFERENCES

[1] H. H. Rosenbrock, *Computer-Aided Control System Design*. London: Academic Press, 1974.

[2] Special Issue on Computer-Aided Design of Control Systems, *IEEE Contr. Syst. Mag.*, vol. 2, no. 4, Dec. 1982.

[3] D. K. Frederick, "Computer programs for the simulation and design of control systems," presented at the Arab School on Science and Technology, Bloudan, Syria, Sept. 1981; in *Advances in Control Systems, Theory and Applications*, J. B. Cruz, Jr., Ed. Greenwich, CT: JAI Press, Sept. 1984.

[4] *Proc. Second IFAC Symposium: CAD of Multivariable Technological Systems*, Purdue University, West Lafayette, IN, Sept. 1982.

[5] *CACSD '83 Abstracts*, IEEE Control System Society Symposium on CACSD, Cambridge, MA, Sept. 1983.

[6] H. A. Spang, III, "The federated computer-aided control design system," in *Proc. Second IFAC Symposium: CAD of Multivariable Technological Systems*, Purdue University, West Lafayette, IN, pp. 121–129, Sept. 1982; also this issue, pp. 1725–1731.

[7] J. H. Taylor, "Environment and methods for computer-aided control systems design for nonlinear plants," in *Proc. Second IFAC Symp.: CAD of Multivariable Technological Syst.*, Purdue University, West Lafayette, IN, pp. 361–367, Sept. 1982 (Issued as General Electric Rep. 82CRD193, GE Corporate Research and Development, Aug. 1982.)

[8] J. C. Doyle and G. Stein, "Multivariable feedback design: Concepts for a classical/modern synthesis," *IEEE Trans. Automat. Contr.*, vol. AC-26, no. 1, pp. 4–16, Feb. 1981.

[9] Y. S. Hung and A. G. J. MacFarlane, *Multivariable Feedback: A Quasi-Classical Approach*, Lecture Notes in Control and Information Sciences, vol. 40. New York: Springer-Verlag, 1982.

[10] M. Vidyasagar, *Control System Synthesis. A Factorization Approach*, Fifth Draft, Jan. 1984. (To be published by MIT Press.)

[11] J. K. Hedrick, "The use of statistical describing functions with linear quadratic Gaussian controller design," in *Proc. Joint Automat. Contr. Conf.* (West Lafayette, IN), pp. 390–393, 1976.

[12] J. H. Taylor, "A systematic nonlinear controller design approach based on quasilinear system models," in *Proc. American Contr. Conf.* (San Francisco, CA), pp. 141–145, 1983, and (San Diego, CA), pp. 817–824, 1984.

[13] S. C. Shah, R. A. Walker, and D. B. Varvell, "A control design workstation," presented at the IEEE Contr. Syst. Soc. Symp. on CACSD, Cambridge, MA, Sept. 1983 (see [5]).

[14] R. H. Travassos, "Completing the CACSD process," presented at the American Contr. Conf., San Diego, CA, 1984.

[15] H. A. Sutherland and K. L. Sonin, "A control engineer's workbench—A methodology for microcomputer implementation of controls," *Proc. American Contr. Conf.* San Diego, CA, pp. 115–119, 1984.
[16] J. Wieslander and I. Gustavsson, "IDPAC—An efficient interactive identification program," presented at the 4th IFAC Symp. on Identification and Syst. Parameter Estimation, Tbilisi, USSR, Sept. 1976.
[17] S. Shah, R. Walker, and C. Gregory, Jr., "Matrix$_x$: A data analysis, system identification, control design and simulation program," in *Proc. Second IFAC Symp.: CAD of Multivariable Technological Syst.* (Purdue University, West Lafayette, IN), pp. 131–136, Sept. 1982.
[18] J. H. Taylor and D. K. Frederick, "Expert system concepts for computer-aided control engineering," General Electric Rep. TIS 84CRD127, GE Corporate Research and Development, June 1984.
[19] D. Michie, *Introductory Readings in Expert Systems*. London: Gordon & Breach, 1982.
[20] F. Hayes-Roth, D. Waterman, and D. B. Lenat, *Building Expert Systems*. Reading, MA: Addison Wesley, 1983.
[21] W. B. Gevarter, "Expert systems: Limited but powerful," *IEEE Spectrum*, pp. 39–45, Aug. 1983.
[22] R. Davis and D. B. Lenat, *Knowledge-Based Systems in Artificial Intelligence*. New York: McGraw-Hill, 1982.
[23] K. J. Astrom, private communication to the authors regarding an expert system for IDPAC [16], Sept. 1983.
[24] 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," submitted for presentation to Control '85, Cambridge, UK, July 1985.
[25] H. E. Johnson and P. P. Bonissone, "Expert system for diesel electric locomotive repair," *J. FORTH Appl. and Res.*, vol. 1, no. 1, pp. 7–16, Sept. 1983.
[26] J. H. Taylor, D. K. Frederick, and J. R. James, "An expert system scenario for computer-aided control engineering," in *Proc. American Contr. Conf.* (San Diego, CA), pp. 120–128, 1984.