CADCS'88 PREPRINTS

4TH IFAC SYMPOSIUM ON COMPUTER AIDED DESIGN IN CONTROL SYSTEMS

SPONSORS

International Federation of Automatic Control (IFAC)
- Applications Committee (APCOM)
- Systems Engineering Committee (SECOM)
- Education Committee (EDCOM)
- Computers Committee (COMPUT)
- Theory Committee (THEORY)

Chinese Association of Automation
China Association for Science and Technology
China International Conference Center for Science and Technology
Natinal Natural Science Foundation of China

INTERNATIONAL PROGRAM COMMITTEE

Chairman: A. van Cauwenberghe (B)
Co-Chairman: Chen Zhen-Yu (PRC)
Vice-Chairman: Qin Hua-Shu (PRC)

P. Kopacek	(A)	J.H. Anderson	(AUS)
K.C. Daly	(AUS)	W. Schaufelberger	(CH)
M. Mansour	(CH)	F. Conrad	(DK)
M. Kummel	(DK)	P.M. Larsen	(DK)
A. Titli	(F)	G. Davoust	(F)
H. Unbehauen	(FRG)	R. Isermann	(FRG)
C. Schmid	(FRG)	H. Rake	(FRG)
L. Keviczky	(H)	K Furuta	(J)
A. Tysso	(N)	P van den Bosch	(NL)
B.A. Ogunnaike	(NI)	Wang Ying-Luo	(PRC)
Han Jing-Qing	(PRC)	Wang Zhi--Bao	(PRC)
T. Soderstrom	(S)	K.J. Astrom	(S)
N. Munro	(UK)	J. Colantuoni	(USA)
G. Leininger	(USA)	M.J. Shah	(USA)
C.J. Herget	(USA)	G. Stephanopoulos	(USA)
H.A. Spang III	(USA)	J. Taylor	(USA)
T. McAvoy	(USA)	C. Georgakis	(USA)

NATIONAL ORGANIZING COMMITTEE

Chairman: Yu Jing-Yuan
Vice-Chairman: Wang Zheng-Zhong

Zhang Zhen-Hua	Wang Zi-Ping	Wu Zhi-Ming
Mao Jian-Qin	Dai Guna-Zhong	Shu Di-Qian
Han Zhi-Gang	Li Shu-Ying	Ye Qing-Kai
Xiong Guang-Leng	Tu Qi-Lie	Mao Xu-Qin

EXPERT-AIDED ENVIRONMENTS FOR CAE OF CONTROL SYSTEMS

James H. Taylor
Control Systems Laboratory
GE Corporate Research & Development
Schenectady, NY 12345 USA

Abstract: - Artificial intelligence in general and expert systems (ESs) in particular have recently gained prominence as effective approaches to computer-aided problem solving. The earliest successes of ESs were in the area of diagnosis or trouble-shooting; computer-aided engineering (CAE) is a much more recent application area of this technology. The use of ESs for computer-aided *control* engineering (CACE) is the specific focus of this presentation. The primary goals are to discuss frankly our group's recent work in this area (see References), to outline various approaches to expert-aided CACE and their implications, and to present a balanced, realistic view of the promise, limitations, and cost of ES technology for CACE. The intent is to provide a personal viewpoint, based on observations and lessons learned in the course of our effort rather than a rigorous survey of the field. It is hoped that this retrospective / tutorial discussion will be beneficial to others working or about to work in this field.

Keywords. Control system design; computer-aided engineering; computer-aided design; expert systems; artificial intelligence; heuristic programming.

1. INTRODUCTION

Substantial interest has developed over the last few years in the perceived benefits of combining conventional CACE software with some sort of artificial intelligence (AI) system. The most common idea has been to add an expert system to an existing CACE environment; this concept is the thrust of the project we call CACE-III (third-generation CACE environment) and is the particular subject of this presentation. The objective is to provide as much insight and realistic information as possible about this approach in terms of what to expect, what to do, and how to do it.

Several questions should be considered before seriously contemplating or pursuing the development of ES-based CACE software:

1. What sort of users are to be supported by the software?
2. What activity can be and is worth expert aiding?
3. What is the best knowledge representation for expertise in performing that activity?
4. In what form should expert aiding be supplied?
5. How should the expert-aided CACE software be assembled (structured)?
6. What is - and is *not* - supported by an ES-based environment?

These considerations are intimately coupled. It is also fair to say that the success or failure of an expert aiding project is likely to hinge on some or all of these issues.

We will explore the above questions in the following framework: Section 2 defines expert-aided CACE, Section 3 covers knowledge representation issues, Section 4 overviews ES software functional requirements, Section 5 titled "Other Implementation Issues" discusses the interrelated topics of the target user group, selection of activity to be expert aided, and the model or "paradigm" for such aid, Section 6 treats architectures as related to these issues, and Section 7 integrates these considerations into an overall assessment of the use of ESs for CACE.

2. WHAT IS EXPERT-AIDED CACE?

First, what is CACE? Under the designation CACE, we include computer-aided analysis and design tools that can perform at least the following catalog of basic activities: nonlinear modeling and simulation, model identification, equilibrium finding, linearization, and linear analysis and design (both in the time and frequency domains). In many disciplines, there exist non-generic CACE functions that add to this list: trim finding for flight control is a good example. Finally, this list could be extended almost indefinitely by adding other nonlinear system analysis and design methods. The fact that control engineering involves such a wide variety of activity - and thus requires a broad and complicated CAE environment for solving realistic problems - provides much of the motivation for considering expert aiding. By the term "realistic"

we usually mean *not small* and *not linear* - many of the tougher challenges of CACE are presented by such problems.

Next, there are various understandings of what is meant by the term "expert" or "expert-aided" as applied to CAE software. This can cover the range from "this software was designed by an expert to help you perform CAE better", a claim that can be made independent of the type of programming that was employed in developing the software, to "this CAE software is or incorporates 'real' AI software (e.g., an expert system shell)" The term *expert-aided CACE* will be used in the second sense in the discussion that follows.

3 KNOWLEDGE REPRESENTIONS

The first major consideration in developing expert-aided CACE software is deciding what CACE functionality to expert aid and selecting a suitable knowledge representation. A wide variety of knowledge representations exist in AI; in general, they have been motivated and shaped by the class of problem(s) being solved. It is, therefore, important to couple the question of knowledge representation with the specific scope of expert aiding.

We have not made an exhaustive study of the full range of possible knowledge representations available in the AI literature; rather we considered the types of problems we wanted to solve and found an appropriate representation for that class. We thus start by discussing types of CACE functionality that we considered for expert aiding, then describe the knowledge representation we have found useful, as mechanized in the expert system shell (ESS) we used in our work, and conclude by mentioning one contrasting problem area and ES framework as an illustration of the range of possibilities.

3.1 Selection of Expert-Aided Functionality Expert or knowledge-based systems are software environments designed to aid in solving problems that require *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 often complicated and broad in scope, and are not amenable to clear-cut well-posed algorithmic solutions. Many control engineering tasks fit these characteristics. The types of expertise that are required for CACE problem solving are: development and diagnosis of realistic plant models, formulating a well-posed controls design problem, selecting appropriate analysis and design methods, performing design tradeoffs, validating and documenting designs, and, in the course of all this, making effective use of conventional CACE software. In many cases, this involves judgments such as "that frequency response curve fit doesn't look good at low frequencies - perhaps changing the weighting will produce a better result" and the corresponding knowledge required to carry out this decision using available tools. Symbolic information to be processed in CACE includes procedural descriptions of methodologies for CACE (especially the decision-making aspects as illustrated above), problem formulation, 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, etc. It may seem surprising that we include low-level detail such as command sets and syntaxes along with higher-level knowledge; however, in many cases the greatest benefit of expert aiding is relieving the user from having to attend to that type of activity.

In the process of deciding where to invest our effort in expert-aided CACE, we developed the following set of questions and rules: First, define a candidate function for expert-aiding; then consider:

1. Does the candidate CACE function involve a substantial amount of heuristic decision making? If the answer is 'no', then conventional programming may be sufficient for the task and less costly to develop.
2. Does the knowledge required to implement this function really exist in a form that can be clearly stated, or is there reasonable hope that this knowledge can be gained? Often a proposed CACE function has to be discarded because the answer is 'no'; we find that there is a natural tendency to propose developing an ES to do tasks that are not well defined. If the answer is 'no', then eliminate this candidate - unfortunately, no programming environment is suited for such endeavors.
3. Is there a user group that wants that function to be expert aided? We have found several instances where the answer is 'no', either because the function is too important to 'trust' to the ES, or because users believe that important knowledge is gained when doing that function manually.

A second consideration in setting our goals has been the observation that conventional CACE packages do provide suitable functional capability (all the required analysis and simulation power to get the CACE effort done), but suffer from a lack of high-level support to make the software as effective and "user friendly" as typically desired. This is discussed in detail in Taylor and Frederick (1984); in summary, most conventional packages require that the user:

- have substantial expertise and knowledge of low-level detail (command sets and syntaxes, etc.) in order to take advantage of the available capabilities;
- adopt a single, usually rigidly-defined, style of interaction;
- make decisions without adequate support mechanisms and the ability to document the basis for judgment; and
- keep track, mentally or manually, of exactly what has been done and where the results are stored.

In addition, most users must use several packages in order to cover the full range of CACE, compounding

these problems. These may not be serious problems for regular users who have invested the time to learn how to use the packages effectively and who keep that knowledge current; less-than-expert and occasional users do not fare so well. We decided that removing or reducing these obstacles to effective CACE should receive high priority

Based on this line of reasoning, the general focus of our work has been the development of CACE software to provide expert-aiding for complicated or decision-intensive procedures, especially those that involve a substantial amount of routine, low-level effort

3 2 Selection of Knowledge Representation: From the above perspective, the type of knowledge that we want to capture is primarily procedural in nature. In particular, expert aiding the class of CACE functions we selected requires having an experienced controls engineer state in detail what steps are to be taken, what decisions are to be made in executing the procedure, how to judge that an acceptable solution has been obtained, and how to use conventional CACE software to carry out the required steps. This generally produces "declarative" knowledge, that is most readily represented as "production rules", having the generic form:

IF (condition_1 & condition_2 & . . .)

THEN (action_1 & action_2 & . . .)

The elements condition_*i* and action_*j* are expressed symbolically; for example, consider

Rule 1234:

IF ((gain_margin desired_value . = GM_spec) &
*(gain_margin actual_value . <=0.9*GM_spec)*
" 10% tolerance

THEN (write (compensator_type needed . lead))

where the variable *GM_spec* must have a value assigned before the rule can be executed

Production rules can be processed according to two basic strategies: *forward chaining*, in which an "inference engine" goes through the set of rules in a systematic fashion (e.g., depth first or breadth first) and executes any rule whose conditions are met, and *backward chaining*, in which a goal is set (" try_to_execute (action_*j*) "), the inference engine seeks a "target rule" that can in fact carry out action_*j*, checks the conditions of that rule, and either (i) executes it if the conditions are met, or (ii) sets unmet conditions of the target rule as sub-goals that are in turn to be satisfied. The first strategy is useful for initializing a knowledge base or whenever a straight-forward procedure is to be executed, while backward chaining is more effective in setting an objective (e.g., try_to_execute *(write (gain_margin specification . satisfied))*), especially when an objective is embedded in a larger framework where forward chaining to completion may be very wasteful. Many CACE problem domains are of sufficient complexity that a mixed strategy is generally most effective (cf James, Taylor, and Frederick, 1985).

Processing production rules involves generating, modifying, or discarding "facts". The conditions and action illustrated in Rule 1234 above provide examples of facts. Note that an action may not directly involve a fact; for example, the immediate action may be to run a conventional CACE package to generate a numerical result, then process that result to produce more knowledge in symbolic form. The second condition of Rule 1234 must be obtained in this fashion.

The ES generally is initialized by loading a fixed set of facts that set up the problem to be solved; more facts accumulate as a problem is being solved by the expert system, in a data structure usually called the *list of facts*. The evolution of the list of facts is achieved by obtaining new information from the user, by running a CACE program to get new results, or by direct inference (if the conditions of a rule are satisfied, the actions often involve writing additional facts, as Rule 1234 illustrates).

The inference capabilities of rule-based ESs are not limited to the basic inference strategies outlined above. There are numerous refinements, including, most notably, reasoning with uncertain or approximate information and non-monotonic reasoning. The first involves the addition of measures of uncertainty or imprecision of information and conclusions, and mechanisms for propagating these measures as inference proceeds (see, e.g., fuzzy reasoning, Mamdani and Gaines, 1981). This is often required in dealing with human operator questions, such as representing and processing information of the form "If the pressure is high and the temperature is low, then . . "; we have not found an application for this reasoning approach in the CACE functions we have treated to date. Non-monotonic reasoning, on the other hand, 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 ES and eliminating the facts developed after that point ("retracting belief"). It may be necessary to save the ES 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 preceding overview outlines the three major elements of a rule-based expert system: rules, facts, and inference engine. The power of this

programming approach arises from this partitioning, in which knowledge is expressed in rules and facts that are handled in a dynamic fashion using a separate processor (inference engine) which may be as simple or sophisticated as necessary. This division achieves a high degree of transparency, so that the rules can be kept lucid (assuming the expert system developer has a clearly-enunciated problem and solution formulation) and the inference mechanisms can be developed and validated separately for use in any number of applications.

We acknowledge that the above "atoms" of knowledge and basic inference mechanisms are not terribly deep, and that an assemblage of such rules could be programmed in a variety of ways that do not require the use of an ESS. We discussed the advantages of programming procedural knowledge in an ES framework as opposed to conventional programming methods in James, Frederick, and Taylor (1987); in summary,

1. The rules are often quite complicated, having perhaps three of four conditions and as many actions, and there may be as many as several hundred or more such productions to solve a problem of moderate complexity; conventional coding of such a mass of logic may be difficult to implement, maintain, and extend.
2. The first problem may be compounded when dealing with a problem whose solution cannot be stated clearly at the beginning of the project. We found, for example, that the design of a lead / lag compensator (James, Frederick, and Taylor, 1985) could not be translated into rules directly from a controls textbook; rather, we iterated many times, making innumerable rule-base modifications until we were (almost!?) satisfied with the results. When this happens in a conventional programming effort, one usually generates code that must be re-implemented and discarded; the ES approach supports the incremental development of understanding much more powerfully.
3. Most ES environments provide 'help' and 'why' facilities that greatly simplify program development and can be used to satisfy the user that the ES "knows what it is doing" (or doesn't know!).

These factors all reduce to the same issue: what is the most effective programming environment for expert-aided CACE? As mentioned at the beginning of this section, the general answer is: an ESS that supports the most suitable knowledge representation and processing power. We decided that the production rule-based ES form of knowledge representation and basic inference strategy can be used to characterize a broad spectrum of CACE activity that might be expert aided. However, it is fair to note that such a conclusion may be colored by the way the problem solver or ES developer thinks, and may not be objectively true for everyone.

There may well be other problems that are difficult to express and solve using production rule-based systems. To illustrate that there are other classes of problems and corresponding knowledge representations, we merely point out as an example that there are many important problems that require detailed classification and pattern-matching (e.g., image processing and understanding or interpreting time-series data in terms of dynamic or other characteristics). Frame-based ESSs have been developed that have built-in mechanisms for setting up classification schemes in terms of parent / child relations, inheritance properties, slots for key attributes and perhaps production rules, etc., facilitating the processing of taxonometric information and approximate pattern matching in a manner that would be difficult to do in a purely rule-based ES.

4. EXPERT SYSTEM SHELLS

To be effective in CACE applications, an ESS must have certain non-symbolic-processing capabilities in addition to the inference power outlined above. Expert aiding, as defined so far, involves the development of rule bases to carry out high-level CACE procedures; the resulting expert system must mimic an expert or highly competent user, using conventional CACE tools to carry out the procedures as programmed in the rule bases whenever they are invoked. In keeping with this model, the ES must be able to issue commands to the conventional CACE software in the same way as the user, accept the results from the package(s), translate the numerical results into symbolic form if necessary, and carry out whatever numerical and logical operations are required to complete the task. (For example, frequency-response data must be processed to obtain the symbolic information *gain_margin actual_value . 12.345_dB* and then analyzed to arrive at the fact *gain_margin actual_value . <= 15.0_dB*). If this complete cycle of activity cannot be done directly, then the ES would have to be used "off-line", telling the user to enter certain commands and data to the packages, and to return information to the ES in symbolic form. The latter approach may be adequate for experimental purposes (exploring the use of an ES for CACE; cf. James, Bonissone, and colleagues, 1985), but it would not be satisfactory for actual use.

Combining this functionality with the knowledge representation and inference capabilities described in Section 3, the following requirements were established for our work:

1. Production rules must be supported, preferably with variables. The use of variables allows one to implement a rule base with fewer actual rules due the ability to eliminate "hard-wired" numerical and symbolic information; Rule 1234 illustrates this.
2. The shell must support, at a minimum, both forward and backward chaining inference strategies.
3. Multiple rule bases (RBs) must be supported, so that the scope of the expert system can be

kept manageable; see Section 6. Some CACE / ES architectures (§§ 6.1, 6.2) require that one must be able to maintain the same list of facts when switching from one RB to another, so that results do not have to be regenerated and reprocessed.

4. Numerical operations must be supported. For example, simple arithmetic, comparisons (inequalities), etc. must be performed in carrying out most CACE tasks.

5. A reasonable 'why' and/or 'help' facility is needed, both for development and user support.

6. The shell must be able to interface with conventional software in a quasi-interactive mode: The ESS must be able to issue commands for the conventional core software to perform the appropriate operations and have the results reported back to the ESS for translation into symbolic form; in most cases the conventional software must be supported interactively through a number of such exchanges, not run as a batch process.

A survey of a large number of commercially available ESSs was conducted and reported in James (1987). There are, to the best of our knowledge, no completely supportive ESSs available today. This necessitates developing a suitable ESS oneself, or finding the best compromise ESS and modifying or extending it as needed. This latter approach can best be accomplished using an ESS based on an interpretive Lisp environment.

We finally note that the required capability of an ESS is generally driven by the complexity of the problems to be solved. It is beneficial to have the correct ESS / problem match to avoid the extremes of "overkill" (using a powerful inference engine to solve simple problems, usually at a high cost in terms of computation 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).

5 OTHER IMPLEMENTATION ISSUES

At the same time one determines the answers to the questions posed in Sections 3 and 4, it is important to consider what type and style of expert aiding is desired. This involves gaging the capabilities and needs of the potential user group, and then deciding how to provide the appropriate support and acquire the necessary knowledge. These issues are discussed in detail below.

5.1 User-Group Characteristics. The first consideration in our work has been that the user of expert-aided CACE software would be a controls engineer (no - we are not yet ready to propose that your job be eliminated!), or at least a student of control engineering. This provides us with a level of technical expertise and computer familiarity such that a "lead the user by the hand down the straight and narrow path" ES paradigm may generally be rejected out of hand. In contrast, controls engineers will insist on knowing what is happening, will want to be able to influence the course of events, and will not want to be lead through an unnecessarily long or tedious process (e.g., dozens of menu selections or question & answer dialogs) to get a job done. In short, no ES user should be patronized, and expert-aided CACE software must set a particularly high standard in this regard or face the reality that most users will stop using the system after a few frustrating episodes.

5.2 Paradigms for Expert-Aiding. The above paragraph indicates that style is important. To be more specific, several distinct "models" or paradigms for expert aiding come to mind when considering how expert aiding can be provided to a user:

- "Expert System as Taskmaster", wherein the ES leads the user "by the hand" through a strictly-defined procedure;
- "Expert System as Tutor", wherein the ES tells an inexperienced user what is involved in carrying out a procedure, how to do it, and what to look for;
- "Expert System as Assistant", wherein the ES carries out complicated but clear-cut tasks and reports the results to the controls engineer for assessment and iteration;
- "Expert System as Coach", wherein the ES monitors the work of the user and offers suggestions when it believes they are required (see Larsson and Åström, 1985), and
- "Expert System as Authority", wherein the ES solves the problem and provides the solution on a take-it-or-leave-it basis with no opportunity for review and iteration.

The selection of such a model has major implications with respect to the integration of symbolic processing (artificial intelligence) with numerical processing (conventional CACE software); see Section 6: Architectures. In addition, this defines the type of environment that is created, and thus the type of user that is well - or poorly - supported, and the cost of development.

We would contend that neither the Taskmaster nor Authority style of expert aiding would be appropriate for CACE, based on the user characteristics outlined previously. The Tutor model (by which we mean an ES providing detailed 'help' and 'why' support and perhaps tutorial examples, not merely a Taskmaster) may be very valuable during learning, but would probably become tedious unless the tutoring can be switched off, transforming the ES into one of the other paradigms. The Coach style ES is very interesting in terms of being completely unobtrusive yet supportive when necessary; however, it would probably be difficult to implement Coach rule bases except for the small subset of CACE activity that fits into clear-cut patterns that can be treated using approximate pattern matching techniques (cf. Larsson and Åström, 1985). This leaves us with the belief that the CACE practitioner will be best served by providing Assistant expert aiding. Perhaps the "role

model" for this approach is the dental hygienist or medical assistant who acts to relieve the professional of routine chores and low-level detail (according to instructions) and reports back with only the necessary high-level information.

We observe that the distinctions made above hinge on differences that may not be black-and-white and which may easily be changed. For example, the difference between the Authority and Assistant paradigms might be that the Assistant starts with user-supplied specifications, generates the corresponding design if possible, and presents the results (e.g., control system simulation results for step response) to the user for review and iteration, while the Authority decides what specifications are "correct" (based on the characteristics of the plant), and generates one design for acceptance or rejection. The complete difference may be in the following alternatives:

Assistant: Recommended specifications are:
bandwidth = 25 Hertz,
gain margin = 18 dB,
position error 5.0 percent;

Enter specifications: bandwidth ?

Authority: Designing controller for:
bandwidth = 25 Hertz,
gain margin = 18 dB,
position error 5.0 percent.

Judge for yourself which mode is preferable!

5.3 Knowledge Capture: Obtaining the procedural knowledge required to expert aid a CACE task may or may not be simple. In some cases, the methodology has been clearly defined in the literature and is thus ready for programming into rule base form; in other cases, there is a lot of "feel" and "instinct" in the minds of experienced practitioners that may be very difficult to elicit (even when, as has often been the case in our work, the mind of the 'expert' and the mind of the rule base developer are one and the same). The generation of scenarios (e.g., Taylor, Frederick, and James, 1984; Taylor, 1985; Taylor, James, and Frederick, 1987) played a large part in this aspect of the development of CACE-III.

We find that the best way to acquire the knowledge that will be captured by any ES is to carry out the given activity in full detail, with realistic problems, using exactly the required tools. In our work, we have generally selected a nonlinear plant model and first go through a "dry run" on paper, then develop command files and macros to execute real exercises using conventional CACE software, and finally write rule bases to capture the process. At this point, we start using the ES to carry out additional scenarios in full, and keep iterating until we believe we have tried enough examples to validate and "de-bug" the system. Taylor, James, and Frederick (1987) provides the best example of this procedure (but note that the nonlinear CACE rule base is still under development).

There is one limitation or drawback to the use of scenarios: We find that there is a tendency to think that an ES that can carry out one scenario or a few scenarios is a "real system". Often this is not true. In many cases, there is too much "branching", "groping", and "exploration of alternatives" involved in CACE to make a one-scenario ES 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 analysis and design problems.

6. ARCHITECTURES

The selection of an appropriate architecture for ES-based CACE software may be quite directly related to the paradigm or style of expert aiding to be provided. The major architectural issues that are considered here are *(de)centralization* and *control*; as an example of the coupling with style, an Assistant ES may merely be a normally inactive ES with sets of rule bases that are loaded and executed at the command of the user, while the Tutor and Coach paradigms clearly involve a higher degree of control and a central "supervisory" capability that is always active. Another architectural consideration, that may or may not be linked with the expert aiding paradigm, might be called 'manner': conventional CACE software (especially most recent commercial offerings) often has a very 'business-like' interface (e.g., command-driven) that can be used rapidly and responds quickly, while many ESs interact with the user in a more chatty and 'supportive' fashion at a substantially slower pace. The question then becomes whether the user will be more comfortable working primarily with a business-like or supportive interface; this preference should be reflected in the architecture.

We will start this discussion by describing the highly centralized (and idealized) architecture that sprang from our earlier work (Taylor and Frederick, 1984). We will then present a more practical simplified version of that framework (James, Taylor, and Frederick, 1985), and finally outline the decentralized architecture we are presently developing. In each case our goal was to create a CACE Assistant; the architectural change from centralized to decentralized was motivated more by considerations of 'manner' and speed than paradigm. We will dwell on the first of these in some detail, because we feel that it still provides a useful *conceptual framework* for expert-aided CACE, and on the decentralized architecture, because of the growing conviction that most CACE practitioners will find it more *congenial and effective*.

6.1 Centralized Architectures for Expert-Aided CACE: Our first effort in expert-aided CACE was to develop the overall framework that can be

represented as in Fig. 1 The basic idea* was that a *complete, meaningful problem formulation* is the first central issue to be addressed 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 so-called *problem frame* was divided into three parts, with the following content:

- *MODEL* - contains the plant model description, e g , its form (ordinary or partial differential equations), type (linear or nonlinear), and characterization (stable or unstable, minimum or nonminimum phase, (un)controllable and/or (un)observable, et cetera)
- *CONSTRAINTS* - contains restrictions with respect to control architecture (e g , centralized or distributed control), implementation (e g., analog or digital; if digital, then limits in sampling time, processing power, and memory), parametric (e g , gains, input signal limits), et cetera
- *SPECIFICATIONS* - contains performance requirements in terms of time- and/or frequency-domain behavior, performance indices, sensitivity, disturbance rejection, robustness, et cetera

This information is clearly a major focal point for CACE

Given a meaningful control design problem, we saw the human expert working in a parallel abstraction, which we called 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 and design results) and the *process* (methodology, overall plan-of-attack, what has been done and what needs to be done, information required for decisions about the selection of design procedures and tradeoff analysis, etc

These structured lists of facts gave us a basis for conceiving *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 the functional architecture of CACE-III that is depicted in Fig. 1 In particular, we 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. As originally conceived, the functions of the rule bases in CACE-III may be summarized as follows:

- RB1 governs interactions involving 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), Taylor (1985), and Taylor, James, and Frederick (1987).
- 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 mechanized and executed by the ES (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 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)

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

* This discussion is taken quite directly from Taylor and Frederick (1984).

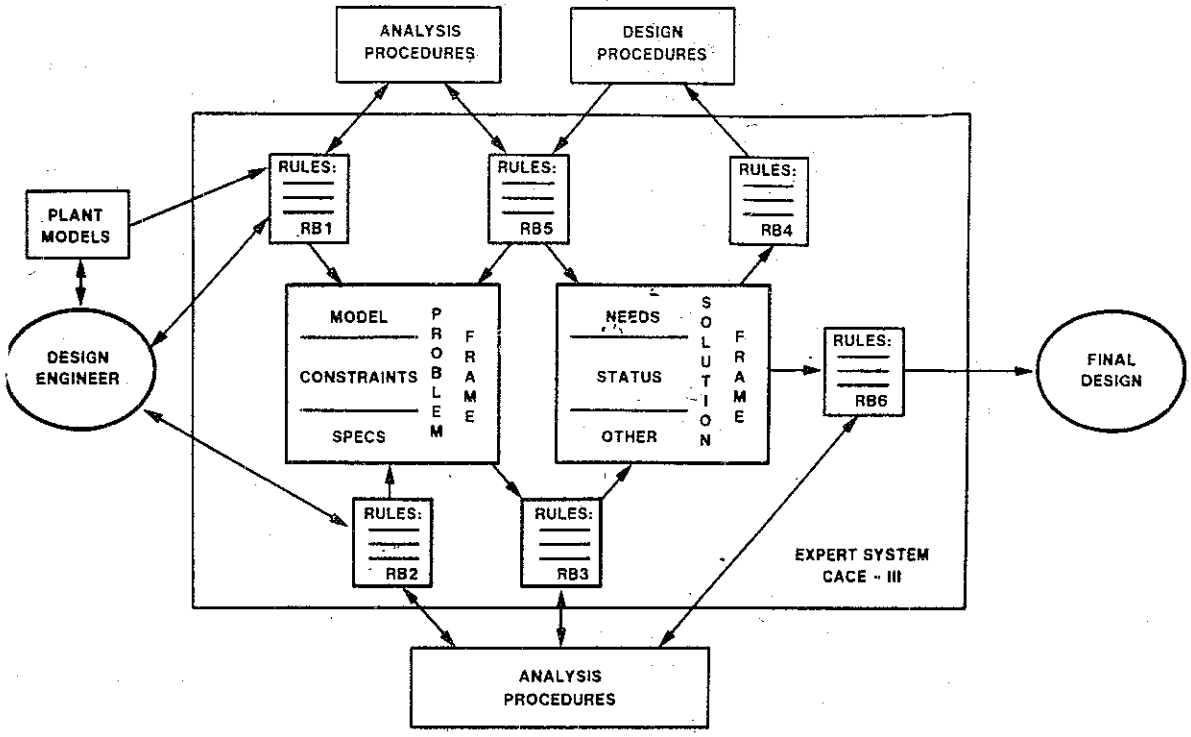

Figure 1. The Conceptual Architecture of CACE-III

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 ES 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 find that the rule base partitioning depicted in Fig. 1 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, in principle, cause the ES 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 should know 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 data-base management (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.

The implementation of CACE-III did not adhere strictly to the idealized framework presented above. The main difference in the actual CACE-III architecture shown in Fig. 2 was eliminating the problem and solution frames. This was done in part because the ESS available to us did not readily support organizing the facts in this fashion, and in part because we didn't see any immediate advantage to building such a system. The main benefit that might arise from adopting such a formalism would be in permitting higher forms of reasoning that require knowledge of the structure of information. Take nonmonotonic reasoning (Section 3.2): the retraction of belief required for back-tracking would be more implementable when facts are organized. Performing a design trade-off study, for example, might simply involve changing one item in the problem frame (a specification or constraint), clearing the solution frame, and starting over. We have not pursued this idea, however.

6.2 Decentralized Architecture for the ES Assistant Paradigm: We have been finding that the Assistant ES paradigm would generally prove to be more effective, and certainly be easier to implement,

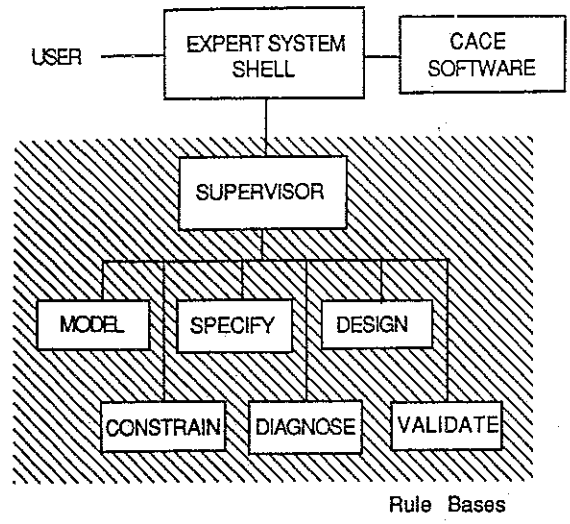

Figure 2. The Actual Architecture of CACE-III

in a decentralized architecture. In this model, portrayed in Fig. 3, the user is not primarily interacting with the expert system; rather, the user is operating all software through a user interface (UI) via a supervisor that manages the operation of both the ES as well as the conventional CACE software. In this model, the UI and supervisor are conventionally programmed, and the ES is inactive until invoked (by the issuance of a command to execute an expert-aided function rather than a conventional algorithm). The ES then takes over the operation of the entire software suite by assuming the functionality of the UI. In other words, the UI is "toggled out", to be replaced by the ES.

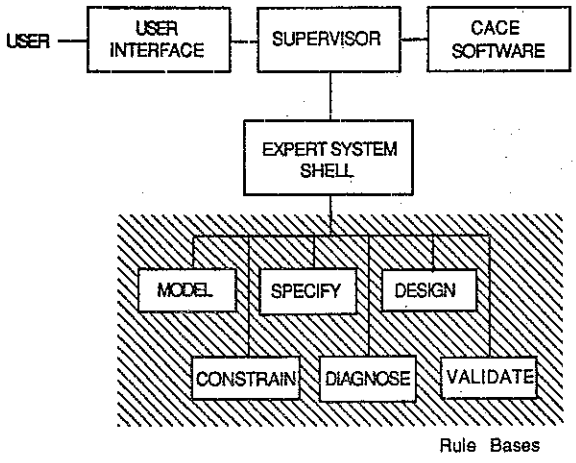

Figure 3. A Decentralized Architecture for Expert-Aided CACE

During operation, the ES may issue any number of commands to the conventional software components via the supervisor, receiving the results and interpreting them, and proceeding with the task until it is completed. At that point, the ES becomes "just another package", and reports the final result to the user via the supervisor and UI.

The scheme outlined above simplifies the structure and operation of the ES (for example, one does not have to worry about keeping the list of facts when bringing in a new RB), and should make the system more responsive. More importantly, the system should be less rigid, since the user performs the higher-level supervision and thus is not limited by the capabilities of a supervisory rule base.

7 OVERALL ASSESSMENT

We have come to a general understanding of the use of AI as support for CACE that might best be characterized as being *"very positive but careful not to get carried away"*. The intent of the preceding sections was to capture what we have learned in the process of developing expert-aided CACE software, so that others may profit by both our successes and mistakes. Some of the lessons may be summarized as follows:

1. Know your users - find out what is needed (functionality) *and* how it should be provided (paradigm).
2. Match the problem being solved with the best knowledge representation and ES software support.
3. Don't expect miracles - you have to have or be able to develop a clear solution to the problem, in every small detail, whether using an ES or a conventional programming environment.
4. Don't underestimate the cost of developing such a system. Unless you have all the functionality outlined in Section 4, writing the rule bases may be a small part of the total effort. In addition, the development of truly general and robust rule bases is no small task, either.

We believe that making CACE software easier to use and more effective may be a better investment of effort than trying to create an ES that, in some high-level sense, "performs like an expert". Eliminating much of the undesirable 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 corresponding functions to be assumed by the ES are invoking packages, executing algorithms, and error handling.

Another excellent application of expert-aided CACE is "technology transfer". Many new methodologies for control system design - especially approaches based on nonlinear analysis and design techniques - involve large amounts of CACE effort to carry out using existing tools and may also entail heuristic judgment that is difficult to convey "on paper". The development of a rule base that can execute the methodology - even if only for relatively simple applications - may be a powerful vehicle for disseminating the approach and getting people to apply it to their own problems.

Finally, the main potential problem in the use of expert systems is the possibility of unrealistic expectations. As we have stated before (Taylor and Frederick, 1984), 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 an *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. These issues require 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; this is exactly the motivation for adopting the decentralized Assistant paradigm in our recent work.

Acknowledgements: I would like to recognize most gratefully the contributions made by Professors Dean Frederick (RPI) and John James (USMA / West Point), both for the effort involved in realizing CACE-III as a working ES, and for many illuminating conversations. I hasten to caution the reader, however, that many observations in this presentation are my personal opinions, based in part on what I have seen looking over their shoulders and contemplating our individual and collective results.

REFERENCES

James, J. R. (1987). A survey of knowledge-based systems for CACSD. American Control Conf. (not in Proc.; write to USMA, West Point NY 10996-1787 USA),

James, J. R., P. P. Bonissone, D. K. Frederick, and J. H. Taylor (1985). A retrospective view of CACE-III: considerations in coordinating symbolic and numeric computations in a rule-based expert system. *Proc. 2nd Conf. on Artificial Intelligence Applications*, Miami Beach, Florida USA.

James, J. R., D. K. Frederick, and J. H. Taylor (1985 and 1987). On the application of expert systems programming techniques to the design of lead/lag precompensators. *Proc. Control 85*, Cambridge, UK; *IEE Proceedings, Vol 134, Pt D: Control Theory and Applications*, No. 3, 137-144.

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. 3rd IFAC Symposium on CAD in Control and Engineering Systems*, Lyngby, Denmark.

Larsson, J. E. and K. J. Åström (1985). An expert system interface for Idpac. *Proc. 2nd IEEE Symposium on CACSD*, Santa Barbara, California USA.

Mamdani, E. H. and B. R. Gaines (1981). *Fuzzy Reasoning and its Applications*. Academic Press, New York.

Spang, H. A. III (1984). The federated computer-aided control design system, *IEEE Proceedings, Vol 72*, 1724-1731.

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

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

Taylor, J. H. (1985). An expert system for integrated aircraft/engine controls design. *Proc National Aerospace and Electronics Conf. (NAECON)*, Dayton, Ohio USA.

Taylor, J. H., J. R. James, and D. K. Frederick (1987). Expert-aided control engineering environment for nonlinear systems. *Proc Tenth IFAC World Congress*, Munich, FRG.