COMPUTER-AIDED CONTROL ENGINEERING ENVIRONMENTS:
ARCHITECTURE, USER INTERFACE, DATA-BASE MANAGEMENT, AND EXPERT AIDING

James H. Taylor,
Control Systems Laboratory, GE Corporate Research and Development,
PO Box 8, Schenectady, New York 12301 USA

Dean K. Frederick,
Dept. of Electrical, Computer, and Systems Engineering,
Rensselaer Polytechnic Institute, Troy, New York 12180 USA

C. Magnus Rimvall and Hunt A. Sutherland,
Control Systems Laboratory, GE Corporate Research and Development

Abstract. Modern CAD environments for control systems engineering have begun to support substantial functionality beyond modeling, numerical analysis, and simulation. Three major areas that have emerged in the last five years are advanced user interfaces, data-base management, and expert-systems support ("expert aiding"). In addition, there has been a steady thrust to integrate more completely the various functionalities and software that comprise computer-aided control engineering (CACE). These trends have produced CACE environments which have relatively modest improvements in numerical quality but a vastly different "feel" in terms of integration, support, and "user friendliness".

The basic considerations and requirements for a CACE user interface (UI), data-base manager (DBM), and expert aiding are discussed in detail. In many cases, these will be illustrated by examples based on various CACE software suites including the new GE MEAD CACE environment; the basic thrust will be to define needs and show how these can be met. The GE MEAD Project involves the integration of powerful CACE packages under a supervisor which coordinates the execution of these packages with a modern UI, a CACE-oriented DBM, and an expert system. The user interface is designed to facilitate access to the CACE package capabilities by users with widely different levels of familiarity with the environment. The data-base manager keeps track of system models that evolve over time and associates each analysis or design result with the correct model instance. The expert system supplies the machinery for expert aiding complicated or heuristic procedures to free the user from low-level detail and tedium. This system thus exemplifies the trends mentioned above.

Keywords. Computer-aided system design; computer interfaces for CAD; control engineering applications of computers; data-base management systems; expert systems; software tools.

1. INTRODUCTION

1.1. Motivation

Progress in many areas of technology requires better performance from embedded control systems. These controls may be in a manufacturing process where the need is for better control in the sense of more efficient use of resources and materials or tighter quality control; or the controls may be an integral part of a product and the performance of that product must be improved (e.g., the fuel efficiency of a vehicle has to be increased).

More stringent demands on the performance of technological systems require the use of advanced controls. This translates into the current trend toward integrating the control of subsystems so that beneficial subsystem coupling can be exploited and adverse coupling can be reduced or eliminated. It also promotes the use of using recent advances in control theory to accommodate dynamic variability, uncertainty, component failures, and other effects and phenomena that may degrade control system performance in some sense.

In turn, the growing demand for advanced integrated control necessitates improvements in computer-aided control engineering (CACE) software, so better designs can be obtained with less cost in terms of time, effort, and errors in design or implementation. This has given impetus to rapid strides made world-wide in the area of CACE software development and usage, and has specifically motivated the GE MEAD Project.

1.2. Trends in CACE

The development of CACE software started several decades ago with the development of routines to perform specific functions that had previously been done manually (e.g., simulation, root locus, Bode analysis). In the 1970s the emphasis shifted to the

"packaging" of routines to integrate them, share common data structures, and broaden their scope. In this second phase considerable attention was also given to the development of numerically robust algorithms; libraries such as LINPACK and EISPACK began to supplant the "home-brew" algorithms that had been used before. More recently, the focus has moved to further broadening of functionality (e.g., block-diagram interfaces for model building, autocode generation) and to improving the overall environment.

The latter issue - improving the environment - is the specific concern of this presentation. Standard phrases in this regard include "enhanced user friendliness" and "more supportive CAD tools". The areas we chose for improvement are the user interface and support facilities for data-base management and expert aiding. We discuss the architecture of such an improved CACE environment and specific needs, design considerations, and implementation issues. Much of the description of needs and considerations that follows is based on the **GE MEAD CACE Environment** (MEAD = Multi-disciplinary Expert-aided Analysis and Design†) and the experience gained in the course of the GE MEAD Project.

1.3. GE MEAD CACE Environment Overview

The GE MEAD CACE Environment (GMCE) has been designed to address the environmental issues outlined above while taking maximum advantage of existing software modules. Implementing the GMCE thus entailed the integration of commercial CACE packages under a *Supervisor* which coordinates the execution of these packages with an *advanced user interface*, a *data-base manager*, and an *expert system shell*. The resulting software architecture is depicted in Fig. 1. The CACE tools ("core packages") include the PRO-MATLAB® package for linear analysis and design, and the SIMNON® or ACSL® package for nonlinear simulation, equilibrium determination, and linearization. Other modules are also based on pre-existing software: the user interface was built using the GE Computer / Human Interface Development Environment (CHIDE) which rests on the ROSE® data-base manager; the GMCE data-base manager uses ROSE and the DEC® Code Management System (CMS®) software for version control; and the expert system uses the GE Delphi® shell which rests on VAX® Lisp. The supervisor and the front-end of the DBM are coded in Ada®.

---

† The origin of the acronym MEAD is the US Air Force MEAD Project (cf. Taylor and McKeehen, 1989), which is a parallel / synergistic effort to that described here. The USAF MEAD effort was sponsored in part by the Flight Dynamics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), United States Air Force, Wright-Patterson AFB, Ohio 45433-6523, under contract F33615-85-C-3611.

® PRO-MATLAB is a registered trademark of The MathWorks, South Natick, Massachusetts; SIMNON is a trademark of Lund University, Lund, Sweden; ACSL is a registered trademark of Mitchell and Gauthier Associates, Concord, Massachusetts; VAX, DEC and CMS are registered trademarks of Digital Equipment Corp., Maynard, Massachusetts; ROSE is a trademark of Martin Hardwick, RPI, Troy, New York; Delphi is a trademark of GE; and Ada is a registered trademark of the U. S. Government, Ada Joint Program Office.

1.4. Outline

CACE environments are discussed within the following framework: Section 2 itemizes basic functional requirements, Section 3 overviews the integration of CACE packages, Section 4 treats concepts for a modern user interface, Section 5 describes the CACE data-base management problem, Section 6 surveys incorporating expert systems in CACE environments, Section 7 outlines an extended example that illustrates many of the points in Sections 3 - 6, and Section 8 provides concluding remarks.

2. OVERALL CACE FUNCTIONALITY

The following list captures the *basic* (minimal) functions that are required of a modern environment for CACE:

1. Modeling: linear and nonlinear systems, in continuous- and discrete-time; building arbitrarily structured models from components
2. Simulation: initializing system state variables, setting system parameters, defining input signals, designating simulation variables for storage, running a simulation
3. Steady-state (equilibrium) determination
4. Linearization
5. Linear Analysis: eigenvalues/eigenvectors, zeros, controllability and observability, model reduction, model transformations, root locus, frequency response
6. Linear Control System Design: frequency-domain methods (by manually adding lead/lag or PID compensation), pole placement, time-domain methods (LQG, LQR)
7. Control System Validation: frequency-domain analysis of linear models, simulation of linear and nonlinear systems

This list is *not* all-inclusive from a control theoretic point of view or from the standpoint of practical functionality. For example, obvious areas that can be extended are stochastic control (e.g., Monte Carlo analysis capability); nonlinear systems analysis and design (e.g., describing function methods, bifurcation analysis); optimization; discrete-event systems modeling, analysis and design; system model identification; and automatic controller code generation. Furthermore, new approaches and theories are being developed on a continuing basis, making closure impossible.

One objective of any realistic CACE software development project must be to set goals as to functionality. The first phase of the GE MEAD Project was limited to the basic CACE functionality itemized above. Recognizing the need for future extension, the GMCE environment has been designed to be "open", in the sense of being extensible either by adding built-in functionality or by use of MEAD macros or the "Package Mode" access to any CACE functionality of the core software (see Section 4.3).

3. CACE ARCHITECTURES

Powerful CACE packages exist that cover large

parts of the overall functionality outlined in Section 2. This enables decoupling numerical functionality from the environmental issues of support and user friendliness that are the main focus of this presentation. The implementation of a modern CACE environment can then be accomplished by providing a "shell" for existing software rather than starting from the foundations of numerical analysis and algorithms.

There are other advantages to this strategy beyond considerations of development time and cost. Most CACE packages are implemented as monolithic software programs with internally-defined data structures and algorithms that are immutable. Thus even extendable packages, such as the MATLAB-derived packages, are limited by the embedded data structures and core algorithms. Since there will probably never exist a single, "frozen" program capable of accomplishing all tasks performed by a control engineer, it seems preferable to shell the best existing CACE packages in an open software environment rather than committing to a specific set of data structures and algorithms.

The approach taken in the GE MEAD Project was based on these considerations. Implementing the GMCE thus required the integration of a data-base manager (DBM), an expert system shell, and several CACE packages, such as PRO-MATLAB for linear analysis and design and either SIMNON or ACSL for nonlinear analysis and simulation. Access to the software is via a common user interface. The resulting architecture is shown in Fig. 1. This system has been carefully designed to integrate different packages into a single, uniform environment despite diverse package interfaces.

At the heart of such an architecture there must be a module that combines and coordinates the various CACE packages and support modules. The GMCE Supervisor serves as the package integrator for the GE MEAD environment (Fig. 1). Multiple packages are run under the supervisor, and data is reformatted or converted, when necessary, to ensure compatibility among packages. The GMCE supervisor accepts "MEAD commands" and translates these into "package commands"; therefore, the user does not have to learn all of the intricacies involved in using each package unless advanced functionality is to be accessed via Package Mode (using a core package under the GMCE via its own interface - see Section 4.3).

The Supervisor contains most of the "intelligence" of the GMCE. It accesses data files that define how to use the underlying packages (e.g., how to convert MEAD commands into package commands, see Rimvall, 1990), directs the activity of the DBM, and manages the command and data flow between itself and the user interface. It tracks the high-level activity of the user, including knowing what model(s) have been configured. Finally, the supervisor controls the invocation and use of the expert system by activating it, telling it what rule base to load and execute, serving as the conduit for communications between the user and expert system, and handling the expert system's results when it is finished.

The User Interface (UI) of MEAD communicates with all different components of the MEAD system, including the DBM and the expert system shell, through the Supervisor. The Supervisor provides the UI and the expert user with a unified and well-structured command-language interface. This same interface is used by the Expert System when it is invoked to carry out a high-level function. In fact, the GMCE supervisor may be used by itself as an "old-style" command-driven interface to CACE functionality, although this is rarely necessary. The clean separation of the "real" user interface from the supervisor has proven to be very beneficial in terms of development, test, and refinement, and has made it possible to implement the MEAD UI and Expert System independent of the quite diverse collection of underlying modules.

4. USER INTERFACES FOR CACE

A primary goal in designing a "user-friendly" CACE environment is to create a user interface that supports control engineers with widely different levels of expertise in using CACE tools. The design requirements necessary to satisfy inexperienced users are very different from those needed for expert users, and this often results in a fundamental conflict that causes engineers from one or the other end of the experience spectrum to be very dissatisfied with a given CACE environment. This conflict may be resolved in two ways: by making the user-friendly features of the interface fast and non-patronizing, and by allowing the user to work flexibly in a variety of modes.

Despite the importance of the UI to the acceptance and success of a CACE package, user-interface considerations have often played a secondary role in their design. Three periods of UI design philosophy can be distinguished in this field:

- The interactive control packages developed during the 1970's had quite crude user interfaces. Most packages, such as KEDDC (Schmid, 1985), used rigid question-and-answer or low-level menu interactions. This kind of a UI can be made almost self-explanatory for the novice; however, it becomes very tedious for an experienced user.
- With the advent of MATLAB (Moler, 1980) command-driven interfaces came into prominence. Primarily, these were designed to "open up" the underlying programs so that users can extend the functionality of a program by adding interactively defined macros and algorithms. However, flexibility often comes at the expense of complexity, forcing the user to recall and accurately enter many rigidly-defined low-level commands to get a high-level task executed. This cost is quite high for a novice user or for a person who doesn't expect to be a regular user of the package; this factor inhibits many computer-cautious control engineers from using such programs.
- The third generation of UIs, as represented by the GMCE, combines the simplicity of modern graphical interfaces, using drop-down menus, forms, and "point-and-click" techniques, with extendable command- and macro-interfaces as

found in the MATLAB family. This provides both the novice and expert user an adequately powerful and yet fully manageable access to the program.

4.1. Graphical Operating Environments

During the last half decade, the workstation and personal-computing arenas have been revolutionized by the advent of completely graphics-based systems. This is best illustrated by the Apple® Macintosh® computer family, which features a graphical operating system relying on icons, menus, and a mouse, enabling the user to perform virtually all necessary actions with "point-and-click" operations. Computers from other vendors feature similar, albeit less dominant, operating environments. These machines have proven to be particularly well suited for inexperienced and casual computer users. Using the same general paradigm, the main GMCE operating environment is menu- and form-based. This is in distinct contrast to the command-oriented operating environments prevalent in modern MATLAB-based CACE packages and many other programs, e.g., SIMNON and ACSL.

Although "graphics" always played an important role in CACE, it was hitherto mainly used for plotting curves (e.g., frequency and time responses), or for enabling graphical input of models in block diagram form. The control engineer was thought to "need the power" of a command-driven interface, just as software engineers were long thought to need the cryptic details of the UNIX® operating system. In both cases there is a strong risk that users will divide into two distinct groups: expert users and unhappy or non-users. Incorporating a graphical interface for CACE should substantially increase the number of effective users without penalizing the expert, as long as the interface is sufficiently flexible and well designed. Such a UI can be illustrated by overviewing the GMCE interface, where the emphasis is on design considerations and the user's perspective.

The GMCE graphical operating environment allows the user to perform all basic controls-related operations in a very consistent manner over mouse-operated menus and forms. The user does not need to know any commands or syntax, and the menu-tree hierarchy is designed so that there is a natural and easy-to-remember path to each desired functionality. In most cases the menu-tree hierarchy is limited to two or three levels for quick access to all domains. At the last level of the menu tree, selection and action forms are used to permit the highly interactive execution of most operations. The GMCE menu tree is depicted in Fig. 3 (this is not a screen image).

Figure 4 shows a screen-image ensemble illustrating the GMCE graphical operating environment. The top half of this is an actual screen rendering; the bottom half contains forms that are obtained by clicking the buttons as indicated by the arrows.

® Apple and Macintosh are trademarks of Apple Computer Corp., Cupertino, California; UNIX is a trademark of AT&T Bell Laboratories, Holmdel New Jersey.

The top-level horizontal menu or Resource Bar is continually displayed across the upper edge of the screen. When a field in a menu is clicked upon, the corresponding sub-menu or action form appears. Activated fields are displayed in reverse video, and each sub-menu appears in decreasing hierarchical order to the right of it's parent menu.

Action forms vary in size and content, but they are always aligned with the right edge of the screen as shown in Fig. 4. The setup buttons enable actions that the user may want or need to perform before executing the function (such as choosing an integration algorithm or defining the end-time for the simulation); the 'Execute' button triggers the actual operation (e.g., simulate). The bottom row of command buttons is always arranged as follows: the 'Display' button to the far left allows the user to view the result and produce hard copies. The 'Save' button lets the user save the result in the data base. A 'Modelize' button is available on forms where the result may be interpreted as a model (e.g., linearization), in which case the result is reformatted into a model and stored as such in the data base. Finally, the 'Done' button moves back down the menu tree. Whenever the user is requested to enter any alphanumeric information, such as the name of a result or model, an additional input form will open up below the action form, as shown for 'Save'.

The GMCE UI is "object-oriented", in the sense that the user selects items and options by point-and-click operations rather than by typing in names or keywords. Various "Browsing Forms" are used to present the user with the available choices, together with any pertinent information about the selectable items. The information within these forms change dynamically as objects are created or deleted. Items presented to the user for selection correspond to the data-base hierarchy, namely, Projects, Models, Components, and Results (Taylor, Nieh and Mroz, 1988). Each element may be displayed appropriately (text files loaded in an editor, plottable data rendered graphically).

Three Browsing Forms are depicted in Fig. 5. In this example, the projects in a user's data base are listed in Fig. 5 (a), and Project 'Tallinn' is designated by clicking on the corresponding left button. Clicking the 'Models' button on the bottom row brings up Figure 5 (b), which portrays the Browse Model Form for Project 'Tallinn' with a list of the models in that project's data base and their key attributes. The user then selects model 'NLPPFBS' and chooses an action to be taken with that model; in this case, clicking 'Descript' reveals the description and the components of model 'NLPPFBS', as shown in Fig. 5 (c). Clicking on 'Results' would bring up a similar Results Browsing Form for the same model.

Several additional aspects of the GMCE are revealed on these Browse screens. All entities in the data base may be assigned individual notes through the UI by clicking on the appropriate 'Edit Note' button. This feature permits on-line documentation at every level. The UI also provides access to other data associations described in Section 5; for example, component references and links may be displayed via the 'Disp Ref' and

'Disp Link' buttons on Fig. 5 (c). These and other details shown in Fig. 5 are clarified in Section 5 and in the Section 7 example that corresponds to parts (b) and (c) of this figure.

4.2. Implementation of User Interfaces

Building a modern UI without tools and utilities is a huge task. A faster and more economical approach is to employ the tools and methods of a suitable User Interface Management System (UIMS). The need for a UIMS arises especially when constructing a user interface that requires advanced features found on engineering workstations. The use of a UIMS can greatly reduce the effort required to produce a user interface and ensure a consistent and reliable design. In addition, a UI founded on a UIMS can be refined and extended much more readily than a UI built monolithically without a UIMS. Note, however, that the "wrong" UIMS may not be an asset. It is important to specify the UI requirements and to be sure that the UIMS meets them.

The use of a UIMS permits the separation of the design of the user interface from the application program and the display device. The UI can be maintained as a separate component, thus easing the maintenance of the overall environment. Changes can be made relatively independently in the various modules.

A secondary purpose for a UIMS is to provide advanced UI capabilities to the end user which would not be easily gained otherwise. Thus, a UIMS should provide a comprehensive and extensible set of interface tools, e.g. graphical display editor, and support UI capabilities such as user profiles; interactive help facilities; session logging; display graphics; and definition, editing and execution of command scripts.

The GMCE UI has been designed and implemented using a GE-developed, experimental UIMS called CHIDE (Computer / Human Interface Development Environment; see Lohr (1989)). CHIDE supports the UI design paradigm outlined above, and has proven to be very effective.

4.3. User Interface Modes

As mentioned previously, a UI should be designed to facilitate access to CACE package capabilities by engineers with widely different levels of familiarity with the environment. This goal may be achieved by permitting the user to work in a variety of modes. These modes should be available in a flexible interactive framework, so one can always work effectively at the most comfortable level. In the GMCE, there are four modes:

- *IDEAS Mode* (IDEAS = Integrated Design Environment for All Systems), using the graphical menu/forms style UI for basic CACE functionality, as presented in Section 4.2;
- *M_Command Mode*, using GMCE supervisor commands when this expedites CACE work compared with the more user-friendly menu/forms interface;
- *Package Command Mode*, using a core package's native commands when a needed result is not available via M_Commands; and the
- *GMCE Macro Mode*, which includes both direct execution of macro files and a flexible macro-edit facility; macros may contain M_Commands, Package Commands, or a combination thereof.

The availability of a variety of interaction modes supports the inexperienced user as conveniently as possible (primarily via IDEAS), while providing the more experienced GMCE user with an extensible and effective environment for CACE.

The two GMCE command modes are accessed by clicking 'Commands' in the Resource Bar depicted in Fig. 4. The operating system may also be accessed via this button. These three options are included in the menu under 'Commands'; clicking one of these selections opens a command-line area for work to proceed. GMCE supervisor commands are described in Rimvall (1990), and package and operating-system command languages are commercially documented.

Package Mode is currently most extensive for PRO-MATLAB, because this package is more open than SIMNON or ACSL, i.e., it supports the generation of an arbitrary number of result types. This is generally true in comparing nonlinear simulation packages which produce a closed set of results (typically time-history files, equilibrium points, and linearizations) with linear CACE packages like PRO-MATLAB which can produce a wide variety of linear analysis results.

Basic DBM support for PRO-MATLAB is provided in Package Mode by "layering" two M_Commands over the package commands:

- Data objects may be saved in the Result data base for the Model by entering the command 'MEADSAVE' that is intercepted and executed by the supervisor; e.g., if the result is comprised of the arrays Q and K, then one may enter *MEADSAVE (Q, K, result => thisdata)* and the result will be stored in the DB with the user-supplied name *thisdata*.
- A data object may be saved in the Project DB as a *new model* via the 'MEADMDL' command; e.g., if the new system is represented in state-space packed form by *Snew (Snew = [Anew, Bnew; Cnew, Dnew])* then the command *MEADMDL (ABCD, Snew, model => thismdl)* accomplishes the goal, where *ABCD* is the type of model (type must be ABCD for continuous-time state-space models or DABCD for discrete-time) and *thismdl* will be the name assigned to the model in the data base.

DBM functionality for SIMNON or ACSL is presently limited to recording every package command in the Condition Spec so that subsequent results can be documented; saving results in the data base can only be done by returning to IDEAS mode for execution. Since only the standard results and models listed above can be obtained in any event, there is no loss of generality in this implementation. Later versions of the GMCE may incorporate result and model saving as outlined above, to eliminate unnecessary mode switching.

Finally, the Macro Facility allows the user to streamline CACE by using custom macro

procedures. Macros may be set up to initialize a GMCE session (e.g., to select a project and configure a key model), to perform a procedure defined by a sequence of M_Commands, to execute a task that may require the use of Package Mode, or to carry out a combination of these activities. The user interface facilitates macro invocation by providing a 'Macro' button on the Resource Bar (again, see Fig. 4) which when clicked produces a listing of all files with the extension 'MMAC' for the user to designate and use. Macros may be invoked directly from storage, or they may be loaded into the editor, modified for the task at hand, and then executed. The following simple examples are GMCE macros for start-up, for nonlinear simulation, and for evaluating the singular value decomposition of a linear model:

Initialization Macro:

```
Select_project ( Tallinn )
Configure ( LinPlnt )
Configure ( NLPlnt#2 )
```

('Tallinn' is the project name; the highest class of 'LinPlnt' and class 2 of nonlinear model 'NLPlnt' will be configured);

Simulation Macro:

```
Equilibrium ( result => steady_state )
Input ( Ref, Step, {15.0, 1.0} )
Simulate ( 12., 0.01, result => REF_15 )
```

(find equilibrium and save; make input 'Ref' a step of amplitude 15 units starting at t = 1; simulate for 12 time units with dt = 0.01 and save);

SVD Macro:

```
Set_mode ( package, promatlab )
a = unpack_ss(S);
[u, s, v ] = svd(a);
Meadsave ( u, s, v, result => svd )
Set_mode ( UI )
```

(enter Package Mode using PRO-MATLAB; split out the A matrix; obtain the SVD; save result in the data base as 'svd'; reset mode to IDEAS).

4.4. Package Unification

Another GMCE UI goal was to provide a totally unified interface for all core packages, with as little as possible left up to the user's memory. This is also illustrated in Fig. 4, which shows the GMCE framework for simulation which is identical whether using PRO-MATLAB or SIMNON or ACSL. This is facilitated by the fact that the M_Commands are the same wherever possible, despite the very different package interfaces.

4.5. Data-Base Access

A UI should be designed to provide access to data-base management functionality with minimal user overhead. If the user has to do a lot of extra work to use the DBM, then it is unlikely that it will be used. In fact, it was possible to design the UI so that the DBM is an asset with respect to overhead, rather than a liability. This was due in part to the natural hierarchical data-base system organization, and in part to the use of the "object-oriented design" features in the User Interface as outlined in Section 4.1.

The first pivotal decision was that a query language would be excessively difficult to implement and use for DB access. In addition, the nature of the CACE data base did not seem to require the typical capabilities of such systems. For example, "find all males over 2 meters tall" has few natural analogs in CACE data bases. These factors led us to display data-element information via a hierarchical set of "Browsing Screens" as illustrated in Fig. 5.

The second realization that streamlined the UI in relation to the DBM was that the browsing screens can be used for CACE functionality in addition to display. Thus one may browse the models in a given project and immediately designate a model for use. This is called "configuring a model", and is done by hitting the 'Config Model' button in Fig. 5 (b). One may also create new models; edit models; purge models; and add, modify, or delete model notes from the same screen.

4.6. GMCE UI Limitations

The present platform for the GMCE is a VAX® computer running the VMS® operating system, with the user interface displayed via a Tektronix® 4107 terminal (or higher model number) or an IBM® PC or PC clone running a Tektronix 4107 (or higher) emulator. This platform is adequate for the type of UI and functionality needed for the GMCE, although it is recognized that the UI could be faster and more flexible if a true workstation environment with high-resolution graphics and window management was used. We plan to port the GMCE to such a platform during 1990.

5. DATA-BASE MANAGEMENT

The specific issues of data-base management that seem to be the most pressing in CACE are related to maintaining the *integrity* of the data base. Primarily, this involves being sure that the model used to generate a result can be identified with certainty and used again if necessary, being sure that the conditions used to generate each result are documented, and knowing how models were obtained if they were generated numerically, e.g., by linearization. This is a much larger task than simply knowing what is in file refinput_step2.dat in subdirectory [user.tallinn.nlplnt]! In addition, there are support functions such as on-line documentation that can add substantially to the value of the DBM.

Earlier generations of CACE packages provided little or no data-base management support. It was left to the engineer to decide how to organize data and track the relations among them. Often organization is based on storage, e.g., data for a project may be kept in a sub-directory or on a tape separate from data for other projects. Data files for a project may be distinguished by assigning "meaningful" file names. Some packages helped by "tagging" data elements of different types by using different extensions, such as the '.m' convention of PRO-MATLAB. One early package (CLADP) generated filenames by appending characters to a user-supplied 'Run name' (Edmunds,

® VAX and VMS are registered trademarks of Digital Equipment Corp., Maynard, Massachusetts; IBM is a registered trademark of International Business Machines, Armonk, New York; Tektronix is a trademark of Tektronix, Inc., Beaverton, Oregon.

1979). All such support is very rudimentary and leaves it entirely up to the user to shoulder the real burdens associated with maintaining the integrity of the data base. The first effort to track models and results and to integrate DBM functionality with a CACE environment appears to be in Bunz and Gutschow (1985). However, the ideas of the complete hierarchy of projects, models, components, and results and of version control were not discussed.

Rigorous data-base management requirements for CACE were presented in Taylor, Nieh, and Mroz (1988). Data-base elements were catalogued and categorized, and the relations among them were established. Then an organization for these elements was devised. Two approaches for data-base access were considered: query language and browsing, as mentioned above. In each case the CACE software user was the main consideration; this involved determining how the data elements are created and used, how the user perceives their relations, and features that are necessary for "doing the job right". Some of the features in the last area include: version control for models that change over the course of a project, recording the conditions (parameter values, etc.) set up before a result is generated, and maintaining model components that are used in more than one model.

This line of thinking resulted in the design of the GMCE DBM, which was developed from first principles. Note that we made the pragmatic decision not to be concerned with the exact representation of each type of data element; instead, we used the data elements created or used by the core CACE packages as a *de facto* standard and only worried about content and format when required for purposes of inter-package compatibility. Work on this aspect of CACE data base definition may be found in Maciejowski (1988).

5.1. CACE Data Elements

In terms of data element categories, the controls engineer works with *models* that are comprised of *components* and a *description* (type, connection, etc.). Associated with each model there are *results*, e.g., files containing frequency response data or time-history data. Models and results are often organized within *Projects* (e.g., project = 'Tallinn' for the analysis and design example presented in Section 7). These considerations led to the DB hierarchy portrayed in Fig. 2; this reflects the belief that control engineers naturally think of projects and models as being of paramount importance; all data elements produced during CACE activity are "children" of these entities.

The *Condition Specification* (C_Spec), also depicted in Fig. 2, is an important secondary data type directly related to each result. This element contains information regarding operations performed on a model before a specific result is obtained; these include such actions as changing a parameter value, specifying an initial condition and/or input signal before performing a simulation, defining a frequency list used in obtaining Bode plot data, etc. Condition Specs also record algorithmic conditions, such as setting a tolerance for controllability or observability, selecting an integration algorithm for simulation, etc. Capturing this data is critical, since it is the combination of model instance *and* Condition Spec that determines the result and thereby allows the engineer to document or repeat the result. Hence C_Specs must be kept in the CACE data base so they can be recovered for any result that has been saved; this is an integral part of the GMCE data-base definition.

5.2. CACE Data Relations

While the CACE data-base categories outlined above are few in number and simple, there are several factors that complicate the DBM task: Models tend to change over the life-time of the project, some results are also models (e.g., linearizations of nonlinear models or transformed linear models), and components tend to be used in several models yet they should be stored in one location to simplify their maintenance. In terms of full support for the controls engineer, mechanisms to handle all of these situations must be incorporated, and this should be done so that the corresponding DBM activity imposes little or no burden on the user.

5.2.1 CACE Model Version Control - The primary need for "version control" in the conventional software engineering sense exists in the model level of the hierarchy. The DBM must be able to keep track of system models that evolve over time, e.g., as better modeling information becomes available or as preliminary modeling errors are corrected, so that each analysis or design result can be associated with the correct model instance. This requirement motivated the use of a tool that tracks each *version* of a model *component* so that version = 1, 2, 3, ... refers to the original and subsequent variants of this component, and each *class* of a *model* that incorporates the component.

5.2.2 CACE Component Maintenance - The CACE DBM requirement for tracking models also gives rise to the need for *non-redundant* model management, since maintaining the integrity of the Model level of the data base is nearly impossible if copies of various components are separately stored and maintained. The GMCE DBM supports this need via *linking*, which allows the engineer to maintain each component in one model (the "home" model) and use it elsewhere by bringing it out of the home model DB and incorporating it in other models.

5.2.3 CACE Model-to-Result Relations - One remaining relation that complicates the hierarchical DB organization is that which associates a linearization as a *result* obtained using a nonlinear model with a linearization used as a model *component*. The same situation exists with regard to linear model transforms. For example, one may create a reduced-order version of a linear model, and desire to save this as both a result and a distinct model for further study. These associations are tracked in the DBM using a mechanism called the *reference*. The user may inspect a linearization result's reference to see if it exists as a component in any model; from the other perspective, a linear model component may be checked to determine if it was obtained as a result generated with a particular nonlinear model so the user can trace that result back to determine how it was obtained (e.g., at what operating point). The value of a linear model

is greatly reduced if component traceability in this sense cannot be assured.

In summary, there is a natural hierarchical organization of projects, models, components, and results + condition specs in the GMCE database. Models are tracked over time via class number and version control. In addition, there are relations called links and references to completely maintain the integrity of the database.

5.3. Data Base Software

Several important decisions were made in selecting the implementation of the GMCE DBM:

- It was decided not to use a major commercial DBM package. Such a module would be too costly to license widely within GE, most of the functionality would not be useful or suitable in filling the somewhat unique requirements for CACE, and some needed functionality might be difficult to obtain or unavailable.
- We did not, however, develop our own software for version control. One major concern is the efficient storage of multiple versions of models; without a scheme like that in DEC CMS that re-builds each discrete version based on stored differences large amounts of storage would be required.
- The ROSE package was used for low-level data-element relation management, since this package was already a part of the UIMS CHIDE (see Section 4.2) and considerable software development was thus saved.

Based on these considerations, the GMCE DBM consists of an Ada-coded "front end" that calls CMS and ROSE to execute lower-level storage and version control activity.

6. EXPERT SYSTEM SUPPORT

There have been several major studies of augmenting conventional CACE software with AI-based support facilities. For reasons of space, we simply refer to the survey paper by James (1988) for an overview of this activity. It is also important to note that there are a number of widely different styles or paradigms of expert-aided CACE, and that the selection is critically dependent on the needs of the user community. An inappropriate paradigm has the potential to detract from the effectiveness of the entire environment. These considerations are discussed in Taylor (1988).

The GMCE expert system shell (ESS) provides the basis for expert aiding clear-cut but complicated and/or heuristic procedures that involve unnecessary low-level detail. One concept of expert-aided CACE was originally defined in Taylor and Frederick (1984); the primary difference in the GMCE involves adopting a less ambitious model for expert aiding that makes the expert system the *user's assistant* (Taylor, 1988) rather than putting it in charge of the CACE effort being performed. This change in perspective was motivated by the specific goal of providing support without getting in the control engineer's way.

A second noteworthy feature of the integration of the expert system with the GMCE is that the ESS interfaces with the supervisor in exactly the same fashion as the user working through the UI. The ESS outputs MEAD commands and/or package commands to the supervisor, and gets the same return as the UI, i.e., a result (if it is brief), a file name (for larger results), or messages (errors or information). This aids knowledge capture, since this is exactly the output and input of the supervisor if a user were to perform a given task.

The first CACE rule base to be built in the GMCE is a reimplementation of the lead/lag compensator design expert system developed by James, Frederick, and Taylor (1985/87). The function of this rule-based system is to accept specifications for steady-state error coefficient, bandwidth, and gain margin, and to design a lead/lag compensator to meet these specifications if possible. At the end of this task, the expert system performs a simulation of the step response of the designed control system. The user may inspect the result and accept the design if the response is satisfactory or iterate on the input specifications to obtain another trial design. The final compensated linear control system may be modelized if the user wants to use it for further study.

7. A CACE EXAMPLE

The following example illustrates the use of the GMCE on a small sample problem. Most of the salient features of the GE MEAD environment are illustrated in the course of this scenario. Note in particular that the data-base hierarchy portrayed in Fig. 3 exactly represents this example. The following outlines the scenario and relates the steps to the various figures:

- Project 'Tallinn' is created; see Fig. 5 (a) for the corresponding entry in the Projects Browsing Screen. Work begins.
- The nonlinear plant model 'NLPlnt' is created; see Fig. 5 (b) for the corresponding entry in the Models Browsing Screen for project 'Tallinn'. (This screen was captured after the whole scenario was done; hence the additional entries.) The single component 'NLPlnt' is changed twice, to refine it suitably; this creates a total of three classes (see Fig. 3). All work below is done with class 3.
- The nonlinear plant 'NLPlnt' is simulated thrice, equilibriated, and linearized; the results and condition-specs are listed in Fig. 3 and the time-history plot of the input 'Ref' and output 'YPlnt' versus time is depicted in Fig. 6. The linearization is "modelized" to create the linear plant model 'LinPlnt' - see the entries in Figs. 3 and 5 (b); note that the 'Reference' relation *('Ref')* diagrammed in Fig. 3 will permit this model to be traced back to the linearization result for the model 'NLPlnt'.
- Two linear control systems are synthesized from 'LinPlnt' using pole placement with poles at $-6 \pm 4j$ and $-10 \pm 5j$, respectively. These are "modelized" to install the linear closed-loop models 'LinPP64' and 'LinPP105' in the data base - see also the entries in Figs. 3 and 5 (b). Note that two 'Refs' diagrammed in Fig. 3 tie these models to the results of the model

'LinPlnt'.

- The performances of 'LinPP64' and 'LinPP105' are checked via simulation (time histories not shown); simulation and eigenvalue analysis results are catalogued for these models in Fig. 3. The model 'LinPP105' was selected as the final linear design based on these results.
- The nonlinear control system 'NLPPFBS' is created with gains corresponding to 'LinPP105'. Note that this model uses the same representation of the nonlinear plant component by linking back to model 'NLPlnt' as indicated by the *'Link'* relation diagrammed in Fig. 3 and by the 'Linked' designation in Fig. 5 (c). The performance of this nonlinear control system is validated via simulation and the result is shown in Fig. 8.

8. CONCLUSION

There has been a growing realization over the last five years that existing CACE packages may be reaching a good state in terms of functionality and numerical power but that there are other areas of support that are required in order to achieve effective CACE environments for less-than-expert users and for use on large projects. Three pressing needs are for a more user-friendly user interface (to expand the usefulness of CACE software to less expert users), data-base management (to rigorously track the many disparate data elements that are generated during the life of a major project), and expert aiding (to alleviate some of the tedium associated with tasks that presently require a lot of low-level detail and a little heuristic logic). We have assessed and discussed requirements and implementation issues in these areas.

In terms of these features, the GMCE typifies such an advanced, more supportive environment for computer-aided control engineering (CACE). The most important novel features are it's flexible user interface including a "point-and-click' interactive mode, two command modes (MEAD and Package), and a Macro Facility; the integrated data-base manager to rigorously maintain all data elements and relations; and the built-in expert system shell to serve as the user's assistant.

In terms of CACE functionality, the present GMCE is a basic CACE package for control system analysis and design. The higher-level functionality of PRO-MATLAB is available through the most user-friendly access mode; all lower-level primitives may be used via Package Mode. The major areas of nonlinear simulation and analysis are also accessible using the GMCE graphical interface. A number of extensions and refinements are planned, including improved UI features, more user-friendly handling of linear models, additional expert aiding, and porting onto a workstation environment.

Acknowledgements: The work described above has taken advantage of many of the results of the USAF MEAD Project. Invaluable support has been provided by Aule-Tek Inc. personnel; the contributions of David Kassover, James Trojan, Michael Charbonneau, and Alfred Antoniotti are most gratefully recognized.

REFERENCES

Bunz, D. and K. Gutschow. (1985). CATPAC - an interactive software package for control system design. *Automatica, 21*, 209-213.

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

James, J. R. (1988). Expert system shells for combining symbolic and numeric processing in CADCS. *Proc. 4th IFAC Symp. CAD in Ctrl Systems '88*, Beijing, PR China.

James, J. R., D. K. Frederick, and J. H. Taylor. (1985/87). On the application of expert systems programming techniques to the design of lead/lag precompensators. *Proc. Control 85*, Cambridge, UK; also in *IEE Proceedings D: Ctrl Theory and Applications, 134*, 137-144.

Lohr, P. J. (1989). CHIDE: a usable UIMS for the engineering environment. Tech. Report, GE Corp. R & D, Schenectady, NY 12301.

Maciejowski, J. (1988). Data structures and software tools for the computer-aided design of control systems: a survey. (plenary lecture). *Proc. 4th IFAC Symp. CAD in Ctrl Systems '88*, Beijing, PR China.

Moler, C. (1980). *MATLAB User's Guide*. Dept. of Computer Science, University of New Mexico, Albuquerque, NM.

Rimvall, C. M. (1990). A data-driven command interface and translator for CACE applications. In preparation.

Rimvall, C. M., H. A. Sutherland, and J. H. Taylor. (1989). GE's MEAD user interface - a flexible menu- and forms-driven interface for engineering applications. Accepted for *CACSD '89*, Tampa FL.

Schmid, C. (1985). KEDDC - a computer-aided analysis and design package for control systems. in M. Jamshidi and C.J. Herget (eds.), *Advances in Computer-Aided Control Systems Engineering*, North Holland, Elsevier Science Publishers, pp. 159-180.

Taylor, J. H. (1988). Expert-aided environments for CAE of control systems. (plenary lecture). *Proc. 4th IFAC Symp. CAD in Ctrl Systems '88*, Beijing, PR China.

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

Taylor, J. H. and P. D. McKeehen. (1989). A computer-aided control engineering environment for multi-disciplinary expert-aided analysis and design (MEAD). *Proc. National Aerospace and Electronics Conference (NAECON)*, Dayton, OH.

Taylor, J. H., K-H Nieh, and P. A. Mroz. (1988). A data-base management scheme for computer-aided control engineering. *Proc. American Control Conference*, Atlanta, GA.

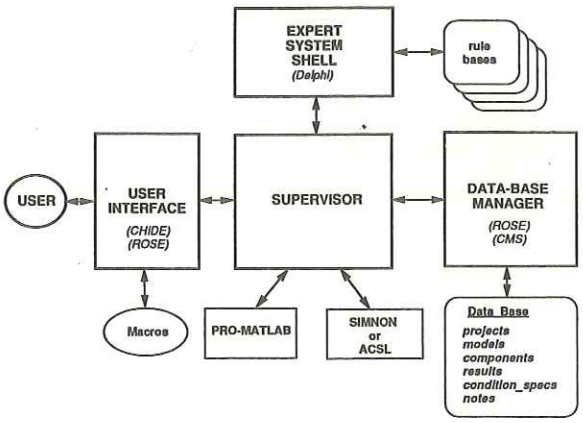


Figure 1. GMCE Architecture

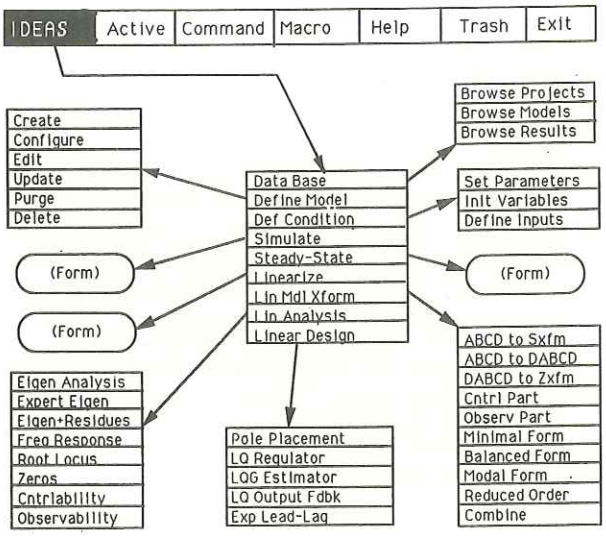


Figure 3. GMCE Menu Tree

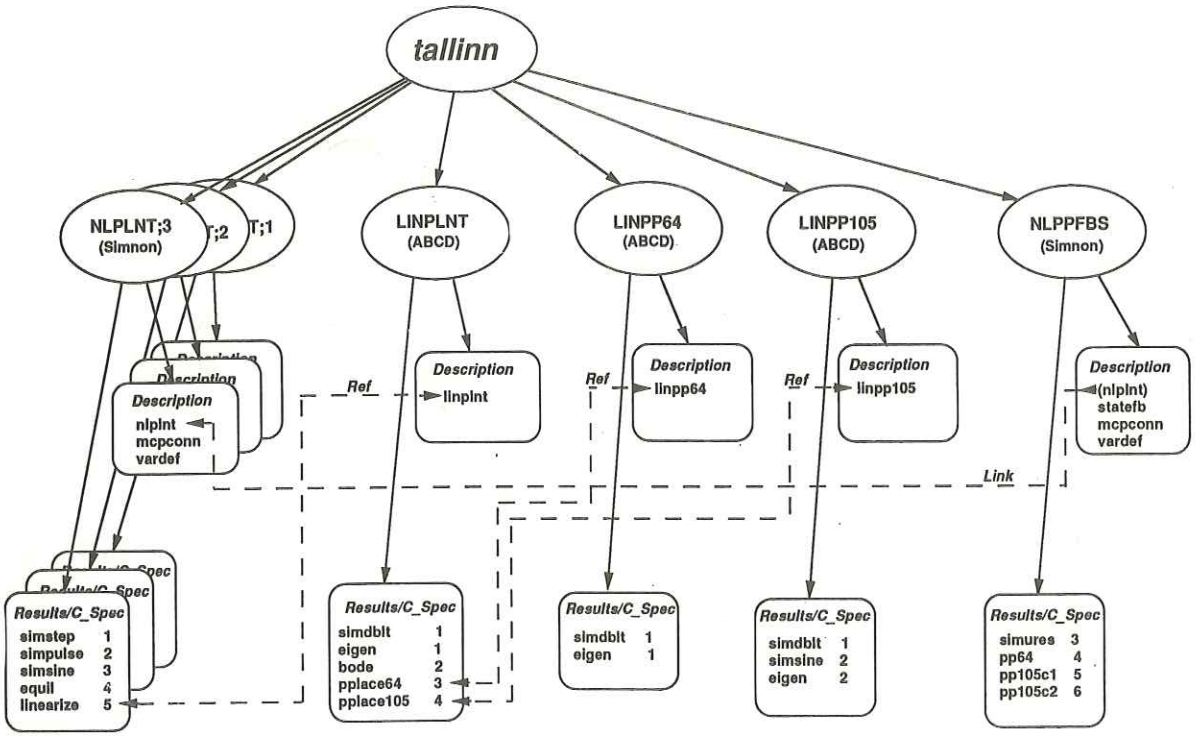


Figure 2. GMCE Data-Base Hierarchy

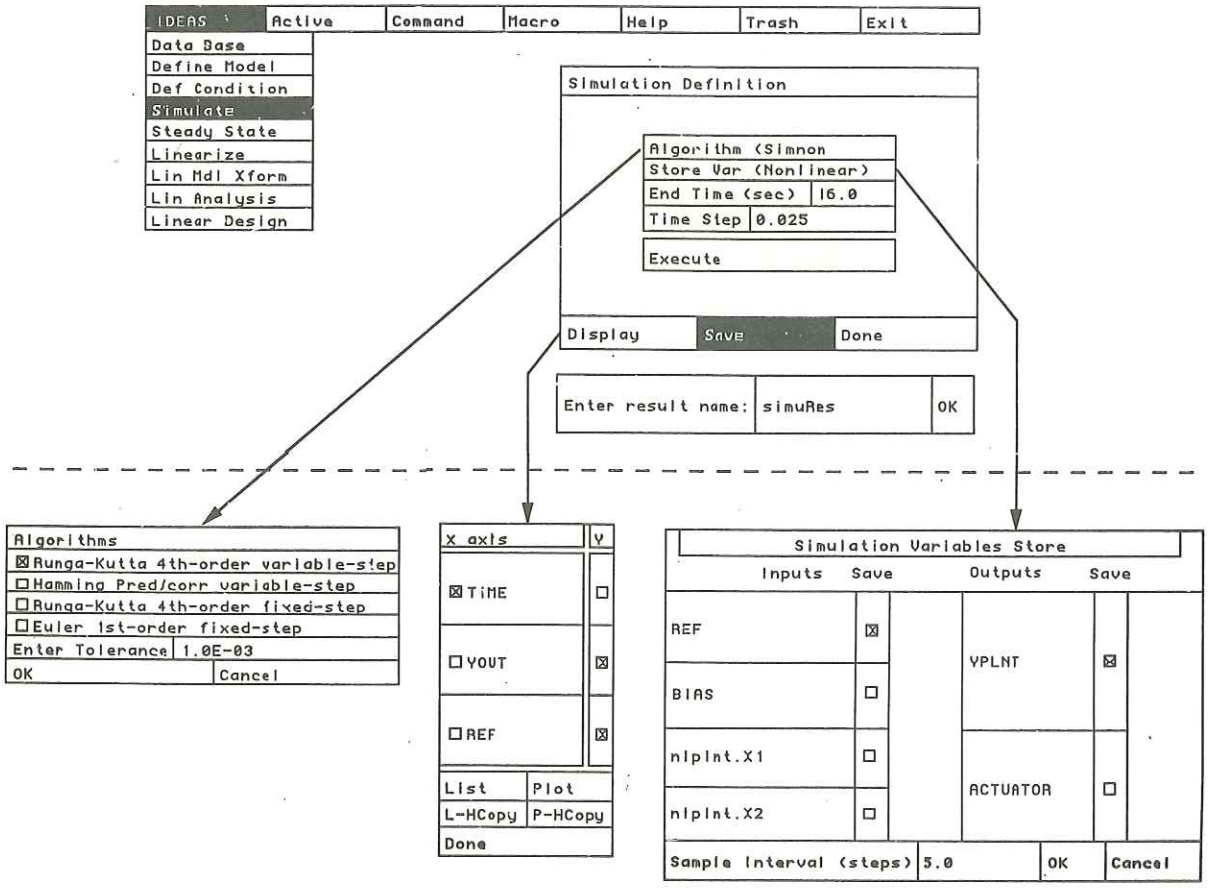


Figure 4. GMCE Simulation Screen and Forms

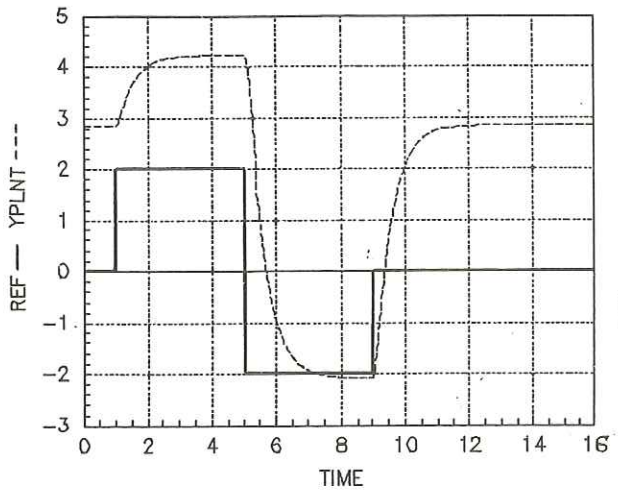


Figure 6. Time History for 'NLPlnt'

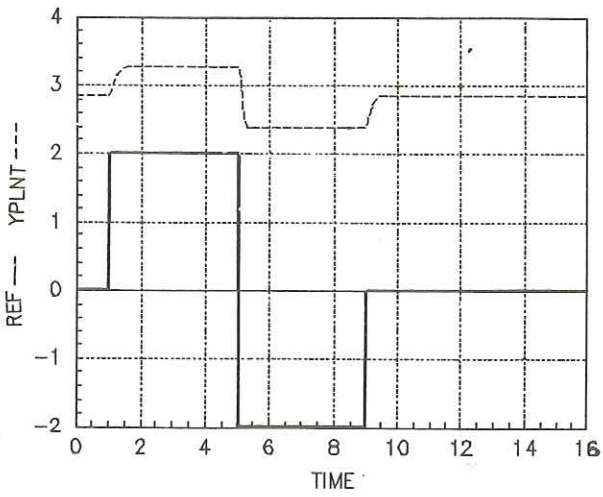
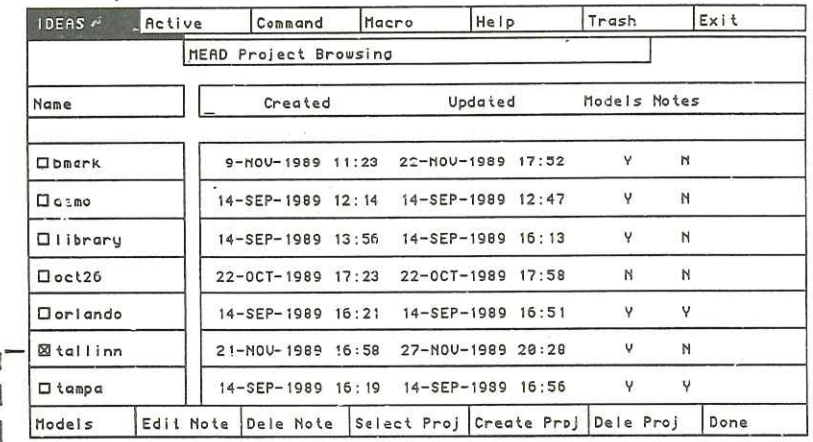
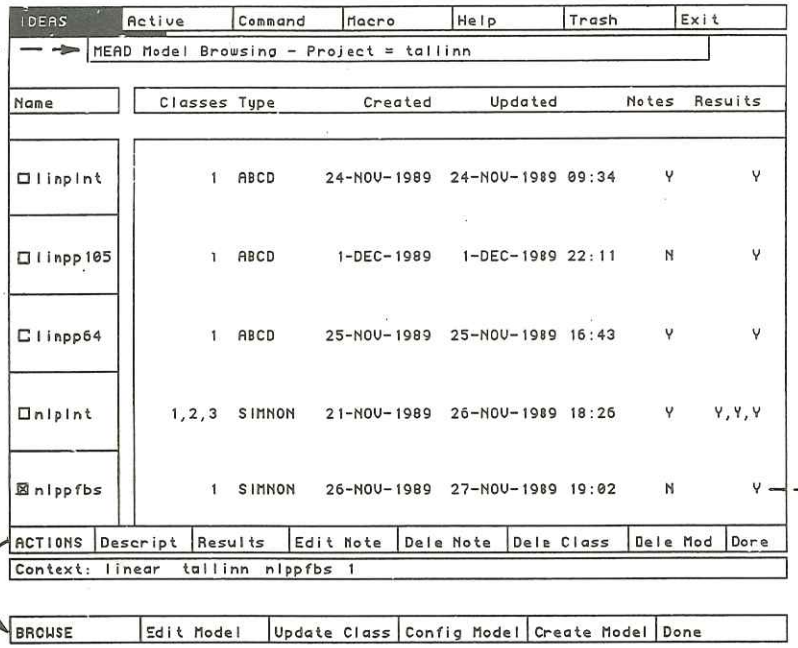


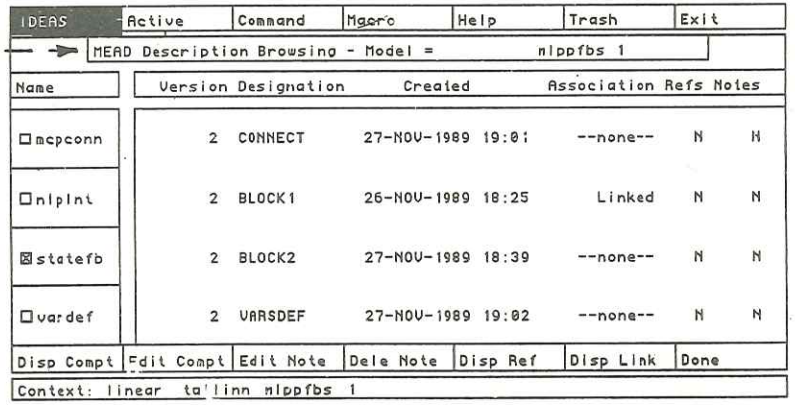
Figure 8. Time History for 'NLPPFBS'



(a) Projects



(b) Models



(c) Description / Components

Figure 5. GMCE Browsing Screens

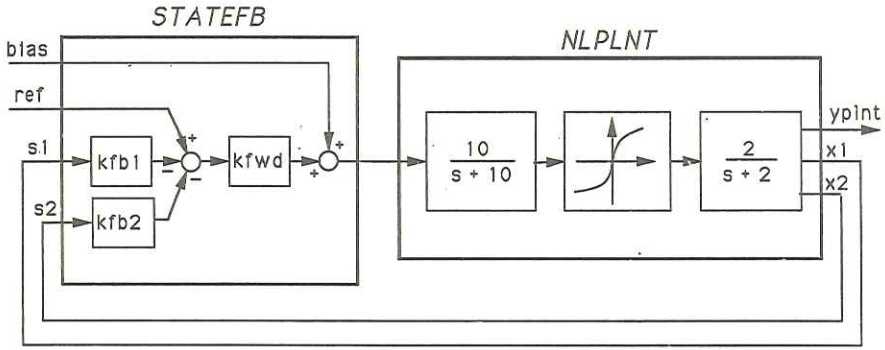


Figure 7. Model 'NLPPFBS'