MEAD II - AN IMPROVED CAD ENVIRONMENT FOR CONTROLS ENGINEERING

James H. Taylor, Magnus Rimvall, and Hunt A. Sutherland

Control Systems Laboratory
GE Corporate R & D
PO Box 8, Schenectady, NY 12301

ABSTRACT

Recent and future efforts at GE to develop modern environments for Computer-Aided Control Engineering (CACE) are discussed. The basic elements of these systems are:

- a User Interface which combines a "point-and-click" menu- and forms-driven interface with other access modes for the more experienced user,
- a Data-Base Manager organized in terms of projects, models and corresponding results and other related data elements and including version control,
- an Expert System Shell, which performs routine higher-level CACE tasks, and
- a data-driven Supervisor that integrates the above elements with existing CACE packages for linear and nonlinear simulation, analysis and design.

As is usually the case, it has been learned that much more can be done to provide a fully supportive environment for controls engineering, and it has also become clear that certain things might better be done differently. This presentation will focus on such areas.

1. INTRODUCTION

Control system performance requirements are continually becoming more stringent. This trend fuels the demand for the use of advanced controls technology, which in turn translates into the current growing need for advanced computer-aided control engineering (CACE) software. This has given impetus to rapid strides made world-wide in CACE software development and usage, and has strongly motivated the GE MEAD Project.

The GE MEAD Controls Environment [1, 2] has been designed to take maximum advantage of existing software modules. This software is the successor to a mature "production" environment prepared for the US Air Force, also called MEAD (USAF MEAD; [3, 4]). The basic elements of MEAD systems are:

- a point-and-click menu- and forms-driven User Interface [5] that supports all basic CACE activity plus other access modes for the more experienced user (command-driven modes and a macro facility),
- a Data-Base Manager [6] which organizes the user's work into Projects which are populated with models, results, and other related data elements,
- an Expert System Shell, which is programmed to perform routine higher-level CACE tasks that are beyond the capabilities of standard packages and require a level of heuristic decision-making or iteration [7] (this is only working in USAF MEAD), and
- a data-driven Supervisor [8] that provides a shell for existing CACE packages for linear and nonlinear simulation, analysis and design, and interfaces with the Data-Base Manager and Expert System.

The resulting software architecture is depicted in Fig. 1. The CACE tools ("core packages") now include the PRO-MATLAB™ package for linear analysis and design, and the SIMNON™ package for nonlinear simulation, equilibrium determination, and linearization. Other modules are also based on existing software: the user interface was built using the GE Computer / Human Interface Development Environment (CHIDE [9]) which rests on the ROSE™ data-base manager; the MEAD data-base manager (DBM) uses ROSE and the DEC™ Code Management System (CMS) 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 the Ada™ language.

Substantial progress has been made in supporting the controls engineer in a number of areas previously given little or no attention. In particular, the higher-level user interface, data-base management scheme, and expert-aiding are noteworthy new contributions. The MEAD User Interface is much more "user-friendly" compared with those of the underlying packages which have rigid command-driven interfaces. The data-base management capabilities solve important problems associated with version control of the user's models and tracking related files such as results - in essence, every element in the user's MEAD data base is fully documented as to how it was obtained. The MEAD expert system adds yet another area of support, which at this point has not been used to full advantage.

The focus of this presentation will be on those areas where the MEAD concept and implementations can be modified, extended and improved. These refinements are primarily aimed at broadening the scope of MEAD and increasing the support and effectiveness of this environment.

™ MATLAB is a registered trademark of The MathWorks, South Natick, Massachusetts; SIMNON is a trademark of Lund University, Lund, Sweden; VAX, DEC is a 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.

2. USER INTERFACE IMPROVEMENTS AND EXTENSIONS

The MEAD User Interface (UI) combines the simplicity of modern graphical interfaces, using drop-down menus, forms, and "point-and-click" techniques, with the flexibility of command- and macro-interfaces. This provides both the novice and the expert user a powerful and yet fully manageable access to the packages. This is a particularly critical issue for CACE applications, since many users are quite computer-literate and demanding.

The MEAD UI supports four modes of interaction:

- The primary UI mode, which is a menu- and forms-based point-and-click style interface, is the most suitable for beginning or once-in-awhile users. Moreover, the expert user is also well served by its efficient operation, as many individual point-and-click action buttons launch more powerful commands than the ones available in the underlying packages. For example, a 'connect' action in the MEAD model-building environment may translate into more than a dozen commands which are sent to the underlying linear analysis package. However, the functionality is definitely prescribed in this mode, so the expert user may not be able to achieve all desired results.
- The MEAD command mode allows the user to directly enter supervisor-level commands (see Section 4). This mode is primarily intended for the expert user wishing to use the full power of the Supervisor's command language (which includes conditional statements, loops, etc.). Although most commands on this level are available via the more friendly menu/forms mode as well, the ability to combine and structure commands freely can expedite tasks for the expert.
- "Package Mode" gives the user the option of entering arbitrary commands directly to the underlying packages, using their native command syntax. This requires the user to know how to operate the underlying package in stand-alone fashion; it is intended to be used when the exact desired functionality is not available through the use of MEAD commands. Commands are entered via the supervisor, so full data-base management capabilities are available on this level [1,2].
- Macro Mode facilitates defining sequences of commands for repeated execution. These commands may be captured in script form during normal operation of MEAD, or they may be entered using a regular text editor. Macros may contain both MEAD and package commands, they may be edited for customizing, and they are automatically loaded into a selection form for easy access in menu mode.

The MEAD graphical operating environment allows the user to perform all controls-related operations in a very consistent manner over mouse-operated menus and forms. A menu hierarchy is used to group related operations together into domains familiar to control engineers [2]. The menu tree hierarchy is limited to two or three levels for quick access to all domains. At the bottom of the menu tree, selection and action forms are used to give a highly interactive execution of most operations. An example of a selection form, used to browse and operate upon models in the data base, is portrayed in Fig. 2 (see Section 3 for more information on data-base browsing).

The look and feel of a UI is improved by adopting style guidelines. The following principles evolved in the development of the MEAD UI:

- Present options in menus or forms so that their applicability is understood before selection.
- Provide easy access from all menus and forms to status displays and auxiliary operations (e.g.,data-base browsing).
- Provide constant feedback to the user as lengthy operations are performed.
- Automatically manage the setup and initialization of auxiliary programs, such as plot utilities, text editors, or block diagram editors.
- Provide context-directed helps which key upon the currently active form or menu.

Standards are now available under the X Window System™, e.g. the Motif™ and Open Look™ toolkits and style guidelines, which offer strong support for consistency among all applications found on the user's workstation, as well as within the CACE tool. The MEAD UI was implemented before these standards were available, but we plan to adopt Motif when converting to X Windows.

Extensions can also be made to increase the functional capabilities of the UI. These include:

- a *smart editor* for entry and execution of commands and macros,
- *user customization* of the menus and forms,
- tools for *adding new rule bases,* and
- *interactive manipulation and viewing* of data.

Smart Editor: A smart editor would provide a facility for the interactive entry of MEAD and package commands, and assist with the construction and execution of macros. This would combine a language-sensitive editor with an interactive mode for immediate command execution. The commands could be built from scratch, or taken from a buffer which contains the sequence of commands previously executed. (The present version of the MEAD UI already provides access to the previously executed commands, via a standard text editor.) The smart editor would contain knowledge about the command syntax, providing either menus of available commands to select from, or command completion with prompting for parameters associated with a command. It would be loaded from the same command-definition files used by the data-driven supervisor, thus ensuring consistency.

User Customization: A toolbox of standard "dialog-boxes" and forms would enable the user to customize

™ X Windows System is a trademark of MIT; Motif is a trademark of the Open Software Foundation; and Open Look is a trademark of AT&T Company.

the functionality provided in the menus and forms, by installing custom macros, thus achieving the seamless integration of user-developed macros as additional functions appearing in the menu tree. (The present MEAD UI automatically provides a *selection* form for user-developed macros, but this form is not integrated with the main menus.)

Adding New Rule Bases: To support this, the UI must provide a more general framework for queries and responses, to facilitate the interaction of the user with the expert system. (USAF MEAD supports some simple communication protocols; however more generality is needed.) The required UI extensions should prove to be minor in addition to those outlined in the item above.

Dynamic Access to Data-base Objects: An example of the concept is shown in Fig. 3, and is based on the notion of attaching various windows with configurable views to objects in the data-base. The windows would then provide either "direct manipulation" or a continuously updated view of the object. As shown in this example, the user has developed a system model containing a compensator with some design parameters to be tuned, and chooses to view both a root-locus diagram and the step response of the system as the design parameters are varied. Starting from the block diagram of the system, three additional windows are opened. One window contains a list of design parameters to be varied, perhaps taken from the compensator block contained in the model data-base. The other two windows display time-history and root-locus results. Then, as the user types in new values in the design parameter window, the plot windows would automatically update.

Such windows must have dynamic access to data-base objects rather than the static access now provided by the MEAD UI. As shown in this example, data-base objects must be simultaneously changed and viewed; the present MEAD UI does not permit this. This concept has great potential to improve the productivity of the controls designer and is currently an active research interest [10].

3. DBM IMPROVEMENTS AND EXTENSIONS

Little has been done to support engineering data-base-management for CACE prior to MEAD. A user's models and results simply accumulated in the workspace (e.g., in a subdirectory under VAX VMS), and it was up to the engineer to perform version control, to relate results to specific model instances and conditions, to relate linearized models to the "parent" nonlinear model and operating point, and so on. To rectify this situation, data-base-management requirements from the user's perspective were developed under the USAF MEAD project [6].

The CACE user's data base is traditionally but informally organized in the hierarchy *Projects, Models, Components,* and *Results*. The user often sets up a workspace for each project (e.g., Project = *GE_654*), develops models (e.g., Model = *Turbine*) which are comprised of components (e.g., Component 1 = *Stator*, Component 2 = *Rotor*, Component 3 = *Fuel_injector*, ...), and which are used to generate various results (e.g., simulation time-histories, linearizations). This has been accommodated directly in the MEAD data-base tree, as illustrated in Fig. 4.

Within this framework, the MEAD Data-Base Manager (DBM) was designed to address the problems of maintaining the integrity and documentability of the user's models and analysis and design results. It achieves this as follows:

- Rigorous *version control* exists at the Component level, and *classes* are used at the Model level to define specific instances. For example, **class=7** of *Turbine* includes *Stator;2, Rotor;7, Fuel_injector;3,* ... where the notation *";k"* refers to version *k* of a component. Any class of *Turbine* that has not been purged can be fetched from the data base and used; any results generated with a given class will be stored with other results obtained with the same class, so there is never any question about how a result was created.
- Traceability between derivative models (e.g., linearizations and reduced-order linear models) and their parents (the original nonlinear or high-order model) is maintained. For example, *Lin_Turbine* is a linearization of *Turbine class=7* at the operating point *Power = 10 000 HP;* this information is stored in the data base as a *Reference* and *Condition_Spec.*
- Single-point storage of components is provided for those that may be used in building any number of models. For example, *Turbine* is the "home" of the component *Rotor;* model *TurbCtrl* uses this same component by *Linking* to it.
- An on-line Note Facility permits the user to store information/on-line documentation for any given project, model, component, or result in the data base. Headers are automatically generated to uniquely identify the element to which a note refers, and time-stamps are included whenever a note is added or modified.

Elements of the user's MEAD data base are accessed by a Browsing Facility that allows the user to display, annotate, purge and delete them via a point-and-click "selection form", as portrayed in Fig. 2.

The main deficiencies of the present version of this DBM are that it is somewhat limited in terms of data manipulation, it is not easy to search for data elements, and it is restricted to access by a single user. In addition, there are some aspects of support that are only partially addressed. These shortcomings can be alleviated by:

- making the DBM more open and flexible - e.g., allow the user to rename, move, compare, and search for specific data elements, and permit the interactive display of notes;
- complementing the point-and-click interface by adding other access modes;
- extending the Notes Facility so the on-line documentation of the user's design activity can be better supported, including automatic document generation; and
- adding functionality to permit *safe* and *flexible* multi-user access.

Flexible Data Element Manipulation: Renaming and moving data elements are elementary functionalities that are easy to implement. (This is necessitated by the fact that users typically become dissatisfied with the original name they gave an element or with where it was placed, and are then very frustrated if such a change cannot be done.) Any element can be renamed (as long as name conflicts are avoided), and moves can be permitted with the following limits: Results cannot be moved from one Model to another, Components cannot be moved to a Model where they are not used (in other words, a Component could only be moved from its "home" to another model that uses it via the Link mechanism mentioned above). Models (and associated components and results) can be moved arbitrarily among the user's Projects.

Comparing data elements can be done in several senses. At the simplest level, one would like to compare various time-histories obtained with a model or several models by cross-plotting the results; this is trivial. For higher-level comparisons, it would be helpful to have an object-oriented system, so each element has a *method* associated with the operation of *comparison;* for example:

- Component *Compt1* can be compared with *Compt2* to see how they differ (these elements could be different versions of the same component or merely similar components); this could be done on the data level ($A_{2,3}$ might have different values) or attribute level (e.g., by comparing their Bode plots),
- *Result1* can be compared with *Result2* to see how they differ (again, either at the data level by using an ASCII differs utility or at the attribute level by cross-plotting the results or determining mean square error), and
- *Result1* can be compared with *Result2* to see how they differ in their *definition* (e.g., Result1 might differ from Result2 because Result1 was obtained with gain $K_{23} = 1.5$ and Result2 corresponds to gain $K_{23} = 2.33$).

Improved Data Base Access: Adding other modes of access to the point-and-click interface would do much to open up the MEAD user's data base. At present, the user has a limited "window" into the DB, e.g., a Browsing Facility Screen may display all the models in a given project (Fig. 2), or all the results for a given model class, and that is all. One way to facilitate finding data elements by name would be to incorporate a way to portray the entire user's data-base tree in a graphical form that conforms to Fig. 4. Such a display would allow one to determine which project contains *Turbine* much faster than by searching the project screens in the DB Browser one after another until it is located.

There are many cases where a command-mode interface would be still more effective. For example, a simple query language could be used to find all simulation result(s) for all classes of model *Turbine* with a step input of amplitude *WF = 2.33* much more expeditiously than browsing. Perhaps a limited subset of SQL (Standard Query Language) would be a good choice for this use.

Improved On-Line Documentation: The Note Facility can be made much more accessible if the notes can be displayed or modified *from the current screen* rather than from the Browser (Fig. 2). For example, if a 'Note' button is always available, then the user can:

- click 'Note' immediately after configuring a model to annotate it,
- click 'Note' immediately after saving a result to document it, and
- click 'Note' immediately after "modelizing" a result (installing a result as a model in the data base) to annotate the new model.

Further extensions could be made to create an auto-documenting environment. For example, MEAD presently does not prompt for notes as the user works and produces new data elements. In addition, the Notes Facility makes no attempt to relate individual note files to an overall document for a project or model. If an auto-documenting environment were implemented and AUTODOC were turned on, then a document framework could be created from templates and every user action that results in saving a data element could be recorded in that report and the user could be prompted for comments/text blocks to narrate the course of the effort. Organizations that require standard report formats and design approaches could thereby capture much of the required documentation material on-line.

Multi-User Data Base Access: Multi-user access to a single MEAD data base is the most important and substantial extension of the MEAD DBM. This would allow several engineers to work on the same project without the duplication of data (models etc.) and the corollary problems of maintenance and coordination. The main issues involved in developing multi-user data bases relate to safety: How can users share models and still be confident that they know precisely what they are using (version and class control provide some support here), and how can users update models safely (e.g., modify and create new classes without using stale versions of components); software engineering tools exist to solve this problem.

Preliminary thinking regarding opening the DBM to multi-user access was presented in [6]. The layer Sub-project was proposed in addition to those shown in Fig. 4, to accommodate a project leader (working at the project level) and other controls engineers working in individual workspaces corresponding to each sub-project. With this extended hierarchy, standard software development tools could be used to allow the leader to maintain the integrity of the overall data base and to control access to the various data elements. For example, DEC CMS (which the USAF MEAD DBM uses for model version control) supports fetching elements for modification in the user's workspace, controlling concurrent changes to the same element, and merging such concurrent changes. It also tracks which users are working on various elements from a library, and maintains a historical record of such transactions. In addition, other software engineering tools under either VAX VMS or UNIX™ automate and simplify building software systems based on source code, object libraries, include files, compilers, and compilation and link options, which would further discipline and rigorize the

building of complicated models.

In summary, much can be done to extend and improve the MEAD DBM. The features that might be considered by other CACE tool developers include flexible and convenient DBM access modes; full version and class control and tracking of results, references and links in the context of multi-user data bases; and provision for auto-documentation. These can be implemented using existing software engineering tools or by selectively duplicating the capabilities of such tools, as required.

4. SUPERVISOR IMPROVEMENTS AND EXTENSIONS

All CACE packages, whether command-driven or graphics-oriented, have some central controlling kernel, or supervisor, governing the execution of commands, storage and retrieval of data, error handling, external file handling, and so on. In command-driven packages this kernel primarily consists of a command-language parser accepting and decoding user commands, a command interpreter mapping these commands onto action routines, and a data handler for storing and retrieving the results of the commands. More modern, graphics-oriented packages such as MEAD relieve the user from having to enter detailed and often ideosyncratic commands, yet these packages also have a similar kernel performing the above operations (where the command parser may be replaced with a module interpreting the user's interactive menu or form operations).

The MEAD Supervisor plays such a central role in the MEAD architecture (Fig. 1). In this position, it serves as coordinator and package integrator. The various numerical core packages run as separate processes under the direct control of the Supervisor, which is responsible for combining and controlling these packages as well as reformatting or converting data, when necessary, to ensure compatibility. The Supervisor is also responsible for facilitating communication among the UI/user, the DBM, and the expert system.

In GE MEAD, the data-driven Supervisor [8] allows all MEAD commands to be defined in external definition files, which include all information needed to call the different core packages. Actions to be performed are input in the MEAD command language, which is then automatically translated into the correct syntax for the target package. The GE MEAD interpretive programming language is capable of data-base and expert-system interactions as well as mapping MEAD commands into different core packages.

The open architecture of MEAD allows the system to be extended in several ways:

- allowing the creation of new commands (e.g., new aggregations of core package commands),
- accessing added functionality/commands in upgrades to already supported packages, and
- supporting new core packages.

Creation of New Commands: Individual users should be able to create new functionality by aggregating existing primitives into more powerful commands. In GE MEAD, this can be done by writing command-language macros. The Supervisor command language includes command flow statements such as conditional statements and loops, as well as some 75 control-theoretic and data-base related commands. These can be combined and structured freely to expedite tasks for the user. A detailed example of this is included in [8].

Accessing New Core Package Functionality: As we have seen, one advantage of a general-purpose command language is that new commands can be added in the form of macros and procedures at any time. In MEAD, this capability may also be used to accommodate any upgrades (especially, new operations) made to the core packages.

Supporting New Core Packages: The GE MEAD Supervisor allows new commands to be added without coding. However, to add a *new package* one must:

- extend the existing MEAD macros to define how various MEAD commands are to be mapped into package commands,
- create Ada-coded translation modules to accept MEAD-language commands and translate them into the new core package syntax, and
- create Ada-coded handshaking modules to interact with the package at the lowest level (recognizing prompts, error messages, etc).

The first step may be accomplished by editing non-compiled MEAD command data-files; only the two last tasks presently require any programming in the traditional sense.

To make the remaining steps data-driven, and thus permit the addition of new packages without modifying the Supervisor source code, it would be necessary to provide a means for formalizing the command language of the new package so that an automatic translation from the MEAD command language to the package command language can take place. This translation is in some sense the reverse of the "compiler-compiler" problem, and similar to the problem of generating a code-generating back end for a given compiler.

An even more challenging problem is formalizing the hand-shaking mechanisms and error-detection/error-recovery schemes of a generic command-driven package so that the supervisor knows what state the package is in at all times. Pattern matching and dynamic state transition tools might provide the necessary machinery.

In summary, our experiences with the MEAD supervisor and with integrating kernel programs such as PRO-MATLAB and SIMNON indicate that the openness of the architecture is the key factor in both ease-of-design and ease-of-use. This will become even more important in the future as different groups experience the need to connect or integrate different controls packages, or integrate controls packages with non-controls software.

™ UNIX is a trademark of AT&T Company.

5. OTHER EXTENSIONS

Automatic Code Generation: The MEAD environments are presently limited to control system analysis and design; controller *implementation* is not a part of this integrated environment. First-generation code generators are available; however, they are not without their problems. Despite these shortcomings, they constitute a first step in the right direction - an automatic code generator to translate block diagrams into real-time code is essential for gaining the engineering productivity sought today.

Disciplinary Extensions: Many extensions can be made to support specific disciplines more completely. Nonlinear system modeling is often time-consuming and prohibitively expensive, so replacing or augmenting general-purpose simulators (see section 1) with domain-specific modeling environments would result in substantial improvements in CACE efficacy. A variety of other extensions could also be made to support ancillary analysis or design functions peculiar to a given discipline, e.g., trimming an aircraft (which is not the same as equilibriating [11]). The UI and Supervisor extensions in previous discussions are essential for providing the infrastructure to make this feasible in a fully integrated fashion.

Expert System Improvements and Extensions: The MEAD Expert System (ES) implementation was based on on viewing the ES as a "control engineer's assistant" [7], i.e., the user invokes a rule base to carry out a task as follows: The button for an expert-aided function is clicked, the corresponding rule base is loaded into the ES, and it proceeds to elicit set-up information (e.g., specifications for a design process) and carry out the task (perhaps with intermediate interaction with the user). The ES then prepares a report, which the user can display and either initiate a new or modified task or move on. Again, note that the ES is only operational in USAF MEAD [3, 4].

The present MEAD ES is very limited in scope and behavior. While it takes advantage of its capability to apply heuristic decision-making in the course of executing a clear-cut task, many other high-level benefits of this technology are neglected. The following extensions are applications of ES functionalities so far unused or underused:

- *smart helps* - asking 'why' and 'how' can elicit useful answers based on the underlying rule-base.
- *tutoring* - the ES could continuously display logical and numerical steps to the user.
- *progress reports* - an intermediate level of user information could be provided every time the ES reached a milestone.
- *user influence* - at each "progress report" the experienced user could be allowed to modify problem solution if it is believed faster convergence will be achieved.

A more detailed discussion of these extensions may be found in [12]. Acknowledgement: the last two ideas have been implemented elsewhere (personal communication of D. K. Frederick).

6. SUMMARY AND CONCLUSIONS

The above sections outline areas where the MEAD environments can be improved. We hope that the lessons learned in developing MEAD will be helpful in charting new courses in CADCS, and that the ideas and extensions described in this paper will be of broad applicability and benefit to many package developers.

REFERENCES

[1] Taylor, J. H., Frederick, D. K., Rimvall, C. M., and Sutherland, H. A., "The GE MEAD CACE Environment", *Proc. IEEE Workshop on CACSD*, Tampa FL, December 1989.

[2] Taylor, J. H., Frederick, D. K., Rimvall, C. M., and Sutherland, H. A., "CACE Environments: Architecture, User Interface, Data-base Management, and Expert Aiding" (invited tutorial), *Proc. 11th IFAC World Congress*, Tallinn, USSR, August 1990.

[3] Hummel, T. C. and Taylor, J. H., "Multi-disciplinary Expert-aided Analysis and Design (MEAD)", *Proc. Third Annual Conf. on Aerospace Computational Control*, Oxnard, CA, August 1989.

[4] Taylor, J. H. and McKeehen, P. D., "A CACE Environment for Multi-disciplinary Expert-aided Analysis and Design (MEAD)", *Proc. National Aerospace and Electronics Conf. (NAECON)*, Dayton, OH, May 1989.

[5] Rimvall, C. M., Sutherland, H. A., Taylor, J. H. and Lohr, P. J., "GE's MEAD User Interface - a Flexible Menu- and Forms-driven Interface for Engineering Applications", *Proc. IEEE Workshop on CACSD*, Tampa, FL, December 1989.

[6] Taylor, J. H., Nieh, K-H. and Mroz, P. A., "A Data-base Management Scheme for CACE", *Proc. American Control Conf.*, Atlanta, GA, June 1988.

[7] Taylor, J. H., "Expert-aided Environments for CAE of Control Systems" (plenary lecture), *Proc. 4th IFAC Symp. on CADCS '88*, Beijing, PR China, August 1988.

[8] Rimvall, C. M. and Taylor, J. H., "Data-driven Supervisor Design for CACE Package Integration", *Proc. 5th IFAC Symp. on CADCS*, Swansea, UK., July 1991.

[9] Lohr, P. J., "CHIDE: A Usable UIMS for the Engineering Environment", Tech. Report, GE Corporate R & D, Schenectady, NY 12301, 1989.

[10] Ravn, O., "On User-friendly Interface Construction for CACSD Packages", *Proc. IEEE Workshop on CACSD*, Tampa, FL, December 1989.

[11] Taylor, J. H., James, J. R. and Frederick, D. K., "Expert-aided Control Engineering Environment for Nonlinear Systems", *Proc. 10th IFAC World Congress*, Munich, FRG, July 1987.

[12] Taylor, J. H., Rimvall, C. M. and Sutherland, H. A., "Future Developments in Modern Environments for CADCS" (invited tutorial), *Proc. 5th IFAC Symp. on CADCS*, Swansea, UK., July 1991.

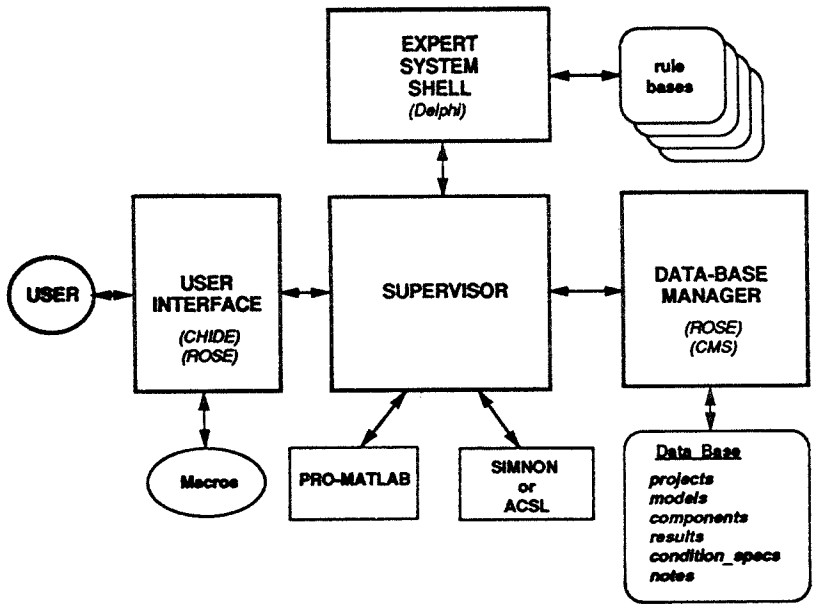

Figure 1. GE MEAD Architecture

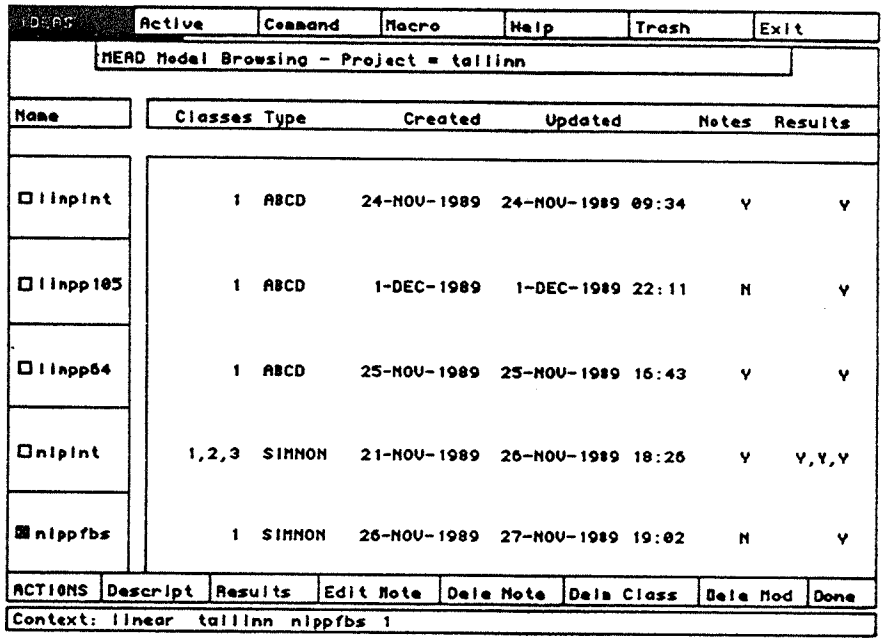

Figure 2. MEAD Model Browsing Screen

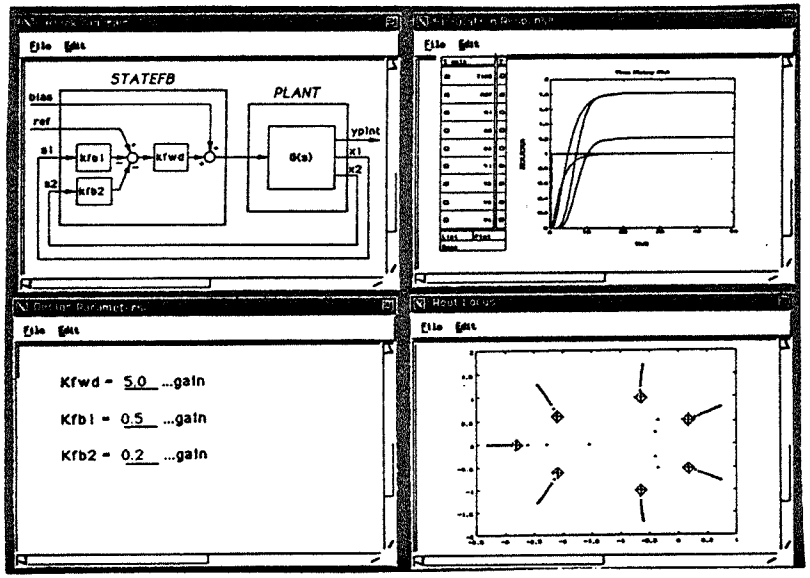

Figure 3. Dynamic Data-Base Viewing

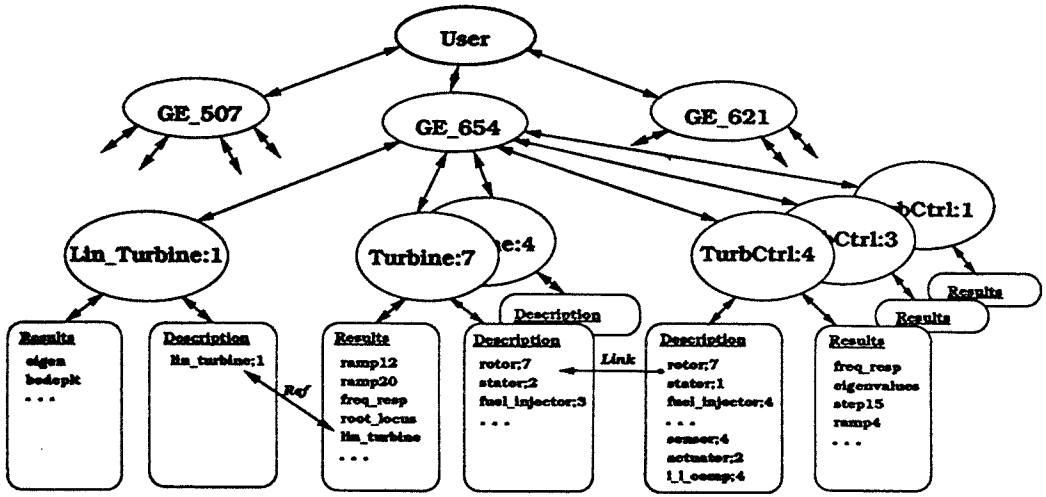

Figure 4. MEAD Data-Base Hierarchy