GE's MEAD User Interface - a flexible menu- and forms-driven interface for engineering applications

Magnus Rimvall, Hunt Sutherland, James H. Taylor, Philip J. Lohr[¶]

Control Systems Laboratory
General Electric Corporate R&D
P.O. Box 8, Schenectady, NY 12301, USA

Abstract

This paper presents the User Interface (UI) of the MEAD[†] Computer Aided Control Engineering (CACE) program. After a brief presentation of the MEAD computer program, the unifying philosophy behind the MEAD UI is discussed. Main features of this UI include a "point-and-click" style interaction, a unifying grouping of similar functionality, and a graphical interface to its CACE data-base management system. A typical modeling, analysis and design scenario illustrates the interface. The use of a User Interface Management System (UIMS) in the design and implementation of the MEAD UI is thereafter discussed. The MEAD user interface is implemented using an experimental UIMS developed at GE which supports both Tektronix® terminal and X window based window systems. Some implementational features of the MEAD user interface are: it is implemented using an object oriented database system, it uses a state tree to specify the dialog control, and it is created entirely by means of a graphical editor thereby avoiding conventional programming. The architecture of the UIMS is described and the implications of creating a user interface with this UIMS are discussed.

Introduction

Our recent effort in software development for Computer-Aided Control Engineering (CACE) has involved the integration of standard CACE packages into an environment called MEAD[1] that includes an advanced user interface, a supervisor[2] which coordinates the execution of CACE tasks using these CACE packages, a data-base manager[3], and an expert system. "MEAD" as used herein refers to GE-MEAD. The GE-MEAD UI was developed entirely with GE funds and consists of GE-owned or liscenced software. The focus of this presentation is the overall design and implementation of the UI, as seen both from a control engineer's "user" perspective and the package designers "developers" perspective.

The design of CACE User Interfaces

Our primary goal in designing a "user-friendly" interface for CACE environments is ambitious: to make the user interface support control engineers with widely different levels of CACE expertise. The design requirements necessary to satisfy inexperienced users are very different from those needed for expert users, and one often finds that there is 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. The interface described here resolves this conflict in two ways: by making the "user friendly" aspects of the interface fast and non-patronizing, and by allowing the user to work flexibly in a variety of modes.

Despite the importance of a good user interface for the acceptance and success of a CACE package, user interface aspects have often played a secondary role in the design of CACE packages. Moreover, even in CACE software developments where much time and effort was devoted to UI design and implementation, the emphasis was often package flexibility and package extendability rather than user friendliness. Three periods of UI design can be distinguished:

- The interactive control packages developed during the 1970's had quite crude user interfaces. Most packages, such as KEDDC[4], 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 to use for an experienced user
- With the advent of MATLAB[5] in 1980, command-driven interfaces came into vogue. Primarily, these interfaces "opened up" the programs. The users can extend the functionality of a program by adding interactively defined macros and algorithms. However, the minimal complexity of such UI's is quite high for a novice user. This prohibits many computer-cautious control engineers from using the program.
- The third and present generation of UI's, as represented by MEAD, 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, to give both the novice and expert user an adequately powerful and yet fully manageable access to the program.

The implementation of User Interfaces

The MEAD User Interface has been designed and implemented using a GE-developed, experimental User Interface Management System (UIMS), CHIDE[6]. A UIMS supplies the tools and methods necessary for building graphical user interfaces. The need for a UIMS arises especially when constructing a user interface that requires many advanced capabilities found on engineering workstations. The use of a UIMS can greatly reduce the effort required to produce a user interface and yet at the same time ensures a consistent and reliable design.

Typically, the use of a UIMS enables the separation of the design of the user interface from the application

---

[¶] Automation Systems Laboratory, GE-CR&D

[†] The origin of the acronym MEAD (Multi-disciplinary Expert-Aided Analysis & Design) is the US Air Force MEAD Project [16], which is a parallel / synergistic effort to the somewhat different "GE version" of MEAD described here.

The USAF MEAD effort was sponsored in part by the Flight Dynamics Laboratory, Wright Research and Development Center, Aeronautical Systems Division, USAF - WPAFB under Contract F33615-85-C-3611.

TH0270-9/89/0000-0024$01.00

program and independent of the display device. In principle the user interface can be maintained as a separate component from the application and thus enhance the ease of overall maintenance of the application, ie. changes can be made relatively independently either in the application or the user interface. As is discussed later, whether the user interface is "loosely" or "tightly" coupled to the application can also affect the ease of maintenance.

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, help facility, session logging, definition, editing and playing of scripts, and display graphics.

MEAD - a Computer-Aided Control Engineering Program

Most CACE programs are implemented as single, monolithic pieces of software containing all necessary data structures and algorithms. Thus, even extendable packages, such as all of the MATLAB-derived packages, are limited by the available data structures and core algorithms. Realizing that there will never exist a single, "frozen" program capable of accomplishing all tasks performed by a control engineer, the approach taken in the MEAD Project has been quite different. MEAD has been created to provide control engineer with a user friendly and yet powerful controls workstation, while taking maximum advantage of existing software modules. Implementing MEAD[1] thus entailed the integration of a data-base, an expert system and several CACE packages, such as the Pro-Matlab® linear analysis and design, and the ACSL® non-linear analysis and simulation package, under a common User Interface. The real advantage of the MEAD architecture thus lies in its ability to integrate different packages into a single, uniform package (despite very different package interfaces).

The User Interface of MEAD communicates with all different components of the MEAD system, including the data-base and the expert system, through a common supervisor[2], which in turn coordinates the execution of the different modules. This supervisor provides the UI, and the expert user, with a unified and well structured command language interface. Thus, it has been possible to implement the MEAD UI independent of the quite diverse underlying modules.

Though largely independent of the underlying CACE packages, the architecture of the MEAD User Interface is closely coupled with the structure of the MEAD Data Base Manager (DBM). The MEAD relational DBM is used to store away models, and results associated with these models, in a well structured, safe and maintainable manner[3]. The hierarchical organization of the MEAD DBM mimics the way control engineers naturally organize models and results, as shown in Figure 1. The control engineer typically analyses models of real or planned systems, and thereafter designs controllers based on these models. Thus, *models* form the main entities of the DBM. Models are grouped into *projects*, a DBM may contain several projects. A model consists of *components* (e.g. plant, sensors, controllers and actuators), connected together according to a model description (ConnSys in Fig. 1). Associated with each model there are *results* (e.g. files containing frequency or time response data).

While the CACE database categories are few in number and simple, there is one more dimension to the problem: Models tend to change over the life-time of the project. Thus, the DBM must be able to keep track of 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. The MEAD DBM includes a version control mechanism to handle this with no burden on the user. Figure 1 shows how each model may have different versions, each with a different *class* number. Different component versions may be organized into these distinct classes, and the results stemming from each model class are kept separated in the data base.

The DBM functionality outlined above is provided at virtually no cost to the user. In fact, accessing the database via the UI Browsing Facility and the ability to make direct use of data elements from that facility makes the DBM an asset rather than a liability in terms of overhead. Moreover, the well-defined structure of the data stored in the DBM is used to simplify user interaction by extracting information from the DBM rather than from the user whenever possible. Also, the DBM can be used as a self-documenting project development log, an important feature in large, industrial projects.

The MEAD User Interface

The MEAD User Interface (UI) is designed to successfully facilitate access to the CACE capabilities of MEAD in a truly heterogeneous industrial environment. As will be further illustrated in the paper, the MEAD UI's multifaceted design allows the system to

- support users with widely different levels of familiarity with the environment,
- provide a single tool both for the engineer confronted with an occasional control problem and the expert control engineer using the most sophisticated control algorithms,
- access different control packages in a uniform fashion,
- provide uniform interactions for similar but disjoint tasks,
- manage the data for very large projects in a reliable manner.

By taking advantage of the synergism among these functionalities, a homogeneous environment within a large industrial setting will be obtained. Thus, MEAD should be seen as one of the first general-purpose but yet fully integrated modeling, controls, simulation and data-management packages.

The task of producing a single program serving a large, multidimensionally heterogeneous community is non-trivial. In designing the MEAD UI, two basic principles have been **consistency** and **agility**. The user is presented with a very predictable menu/form environment operated with "click-and-point" operations. Careful menu/form layout design passively aids the user in his work, and an interactive help facility gives active assistance when necessary. More experienced users are also given the ability to enter complex commands in a fast

and efficient manner. Thus, to accommodate the largest number of control engineers, a user may drive MEAD in different modes, each suiting different needs and level of experience:

- The primary UI mode is a menu and forms based "point-and-click" style graphical interface. This mode gives quick and convenient access to basic CACE functionality. The well structured menus and forms, together with context-dependent helps at each action point on the screen, makes this mode suitable for beginning users. Moreover, as each individual action button and form launches more powerful commands than the ones available in the underlying packages (for example, a connect action in MEAD translates into over a dozen commands which are sent in to the underlying package), the expert user is also well served by its efficient "point-and-click" operation. 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. This mode is primarily intended for the expert user wishing to enter command using the full command language of the supervisor (which includes conditional statements, loops, etc.). Although most of the commands on this level are available through the more friendly user/menu mode as well, the ability to combine and structure commands freely can expedite tasks for the expert user.

- The "package-level" mode gives the user the option of entering arbitrary commands directly to the underlying packages, using the native command syntax of the package. This mode requires the user to know how to operate the underlying package in a stand-alone fashion, it is intended to be used when the exact desired functionality is not available through the MEAD commands. Because the commands are entered through the supervisor, full data-base management capabilities are still available on this level[3].

- A macro mode allows the user to define sequences of commands for repeated execution. These commands may have been captured in script form during normal operation of MEAD, or they may be entered in a regular text editor. Macros may contain both MEAD and package level commands, they may be edited for customization, and they may be dynamically associated with menu fields for easy access when in menu mode.

The graphical operating environment of MEAD

During the last half decade, the workstation and personal computing arena has 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 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 computer "illiterates" and casual computer users. Using the same general paradigm, the main MEAD operating environment is menus and forms based. This is in distinct contrast to the command-oriented operating environments prevalent in modern MATLAB-based CACE packages. Although "graphics" always played an important role in CACE, it was hitherto mainly used for plotting curves (e.g. frequency and time responses), or 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.

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. The menu tree hierarchy is limited to two to 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. All of this is illustrated in the Figures 2, 3 and 4.

Figure 3 shows a screen-dump of the MEAD graphical operating environment. The top-level horizontal menu, or *Resource Bar*, is continually displayed at the upper edge of the screen. When a field in a menu is clicked upon, the corresponding sub-menu or action form pops up. The main resource bar and its drop-down *sub-menus* have been structured to be consistent with other menu-based systems (e.g. that of the Macintosh® and Sun® computers), making the UI uniform in appearance and transparent to use throughout the CACE domains. In the snapshot shown in Fig. 3, the main "IDE" button (Integrated Design Environment) has been selected, followed by the "Lin Mdl Xform" and "ABCD to DABCD" selections, which brought up the continuous to discrete model transform form. Activated fields are always displayed in reverse video and sub-menus appear in decreasing hierarchical order to the right of the original menu. Action forms vary in size and content (also see Fig. 4), but they are always aligned with the right edge of the screen.

In an object-oriented user interface like the MEAD UI, the user selects items and options over point-and-click operations rather than typing in a name or a certain keyword. In MEAD, different *Selection Forms* are used to present the user with the available selections, 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 on selection forms range from projects, models and components as stored in the data base, to simulation output signals to be graphically displayed and other object groups dependent upon the internal structure of the active model. As an example of a selection form, Figure 4 shows the Browse Model form. This form presents the user with a list of all available models in the data base and their key attributes[3]. The user first selects one of these models by clicking on the corresponding left button, and thereafter chooses the action to be taken with the selected model by clicking one of the buttons on the bottom row.

Figure 2 shows the general layout of a MEAD *Action Form*. Though few of the actual forms utilize the full functionality shown in Fig. 2, the form layouts have been standardized for clarity and consistency. The Info and Setup buttons enable actions that the user may want to or have to perform before executing the function (such as setting up the min/max frequency for a Bode operation,

or defining the end-time for a time simulation). The Execute button triggers the actual operation to be performed (e.g. to calculate the Bode frequency data). The Secondary Execute button(s) gives the user access to dependent functions (such as the determination of gain and phase margins after a Bode is done). A bottom row of command buttons is always arranged as follows: a Display button to the far left allows the user to view the results of the operation(s) and produce hard copies. The Save button lets the user save the result(s) in the Data Base. The Model button is available on forms where the result may be interpreted as a model, in which case the result is reformatted into a model and stored as such in the Data Base. Finally, the quit button moves back up the menu tree. Whenever the user is requested to enter any alphanumerical information, such as the name of a new result/model, an additional input form will open up below the action form.

All MEAD screens and forms are variants of one of these three basic screen layouts: the top menu system for accessing functionality, the select forms for object-oriented selections, and the action forms for invoking control functions. All actions are controlled by mouse operations (point-and-click) down to the bottom level (where sometimes names or parameters have to be entered using the keyboard). This makes the UI very agile and easy to use. 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 (as the IDE -> Lin Mod Xform -> ABCD to DABCD).

Operating MEAD over its graphical interface

The general screen layout and forms design of the MEAD UI provides the user with a mechanical operation that is very simple and easy-to-learn. In addition to this, an advanced dynamic state-machine takes the responsibility for a large number of administrative tasks, including retaining, coordinating, and reusing already made selections and specifications. Moreover, all entities in the DBM may be assigned individual notes through the UI. Also, if the user returns at a later time to review results, the UI gives access to the so called *condition specification*, which is a list of changes made to a model or other data-base entity. This list is automatically collected by the supervisor and retained within the DBM. For example, an interactive change of a model parameter is retained throughout all following operations on that model, and the fact that the parameter has been changed is entered by the supervisor into unique condition specifications, one of which is created for each result deriving from that particular instance of the model and stored in the data-base in association with the result.

The overall interaction between a cleanly designed graphical layout and the internal state-driven information management is best perceived through a hands-on use of the MEAD program. Second to that, a case study walk-through should give a good perception of what kind of unique support MEAD offers the control engineer during his system modeling, analysis and design. Thus, in the rest of this section we will illustrate the use of MEAD on a small example problem. In this example, we will create a new linear model of a system and make a simulation to calculate the step-response of the system. Albeit simple enough a task, it will still illustrate most of the salient features of the UI.

Whenever MEAD is started, the user is given the option of entering a tutorial mode, in which case he may walk through an on-line tutorial. If the user is already familiar with MEAD, he would instead click to proceed. MEAD would then scan the selected data-base for valid projects (a user would typically work with only one data-base, he can switch to another one by editing his startup file). If the data-base is brand new and no valid projects can be found, the user is asked to enter the name of a new project. If one or more projects already exist, the user is presented with the Browse Projects form shown in Figure 5. This form illustrates another feature of the MEAD UI, namely a dynamically changing layout of UI screens. Before any project has been selected, the form has only four action buttons (select/create/delete a project, quit). After a project has been selected, four more action buttons specifically operating on the selected project are made available, as shown at the bottom of Figure 5. This scheme prevents the user from selecting certain buttons prematurely, i.e. before they have a valid meaning. In another scheme used in other parts of the UI (in particular on drop-down menus), non-active selections are still displayed in order to indicate their general availability to the user, but if the user selects one of these buttons prematurely, the UI will display an error message of the type "Select a model first".

As soon as a first project has been selected, the login sequence is over and the user is presented with the regular resource bar on the top of the screen. However, before any control-related functions are made available to him, he has to create a new or activate an already existing model on which these functions are to operate. In MEAD terminology, this model is thereafter called the *configured* model. Such a configuration involves collecting all components of that model from the data-base, connecting them together, and loading the complete model into the workspace of the linear or nonlinear package.

A model is activated/created over the Browse Models form shown in Figure 6. The "Configure Model" button on this form is used to activate an already existing model. The "Create Model" button will bring up a sequence of forms, as shown in the Figures 7 through 9. In Fig. 7 the user has elected to create a linear continuous state space (ABCD) model with the name TEST. The model is to contain four component connected in a post-compensator configuration (other predefined configurations are available, an arbitrary general connection of any number of components can be constructed using "General Config"). In Fig. 8 the user has indicated that the first component ("Actuator") is to be created from scratch. This component is to be entered in ABCD format, using a template file as base. The user defines the new component in a regular text-editor, Fig. 9 shows the edited template file. After all components have been entered, the system will automatically configure them into a single, state-space model description and store away that model in the data base. Also, the new model description is kept in the workspace as the active model.

In creating the new model, we note how a relative complex model creation is accomplished with a few keyclicks and the editing of four prepared template files. If the same component definition and component connection was to be performed in one of the popular CACE matrix environments, more than a half dozen complex connection commands would have been entered, including the definition of three opaque interconnection matrices. Moreover, the MEAD UI allows the user to copy or link already existing components into other models (the link command will preserve the version control between

the two components, so that all updates to the original component is reflected in both models; copy will severe the connections and make the two components truly distinct) and combine already existing models into components of other models for a true hierarchical model building.

Once a model has been configured, all actions are automatically directed to that model, and it is not necessary for the user to continuously indicate on which model a function is to operate. Thus, the user is now ready to set up the specifications for a simulation of the created model by selecting "Define Condition" and "Set Input". This brings up the selection form shown in the left portion of Figure 10, in which all input signals to the active model are listed. Note that the number and names of these inputs may vary from model to model. The user picks one of these inputs, selects the kind of input signal (not shown) and enters the parameters of the selected signal form as shown to the right of Fig. 10. The defined input signal is thereafter retained as part of the model condition until the user deletes the signal, configures another linear model or exits MEAD.

To alleviate the user from reselecting and reconfiguring models, both a linear and a nonlinear model may be selected at the same time. This parallel workspace is particular useful when comparing the behavior of a nonlinear model and its linear counterpart, or when a linear controller is to be verified against the complete nonlinear model. Each operation in MEAD is defined to operate only on one of the two models (e.g the linear model for pole-placement , nonlinear model for linearization), or the user is given an explicit choice on the action form. As MEAD supports both linear and nonlinear simulation, the simulation action form in Figure 11 gives such a choice. Note that exactly the same action form is used for both the linear and non-linear simulation, despite the fact that two completely different packages with two different command interfaces are used for the two simulation domains (Pro-Matlab and ACSL, respectively). After a simulation has been invoked, the user may save the results as is also shown in Fig. 11.

In Figure 12, the generic display facility of MEAD is shown. The user may plot or list any output time history against time or another time history (phase plots). The same selection form is used when viewing results retrieved from the data base.

Throughout this short scenario, we have assumed that our imaginary user was quite knowledgeable about the MEAD UI environment. Often, however, this is not the case. Therefore, interactive help is available both for general help information and for information on any particular form or button in the UI (over a dedicated button on the multi-button mouse). Figure 13 shows the top help screen in MEAD. Note how the user may browse up and down within the HELP information tree.

The direct command modes.

The graphical interface to MEAD gives the novice and expert user alike a convenient and powerful interface to CACE functionality. However, for tasks not directly supported by the UI, the direct MEAD and Package command modes are available. Moreover, the MEAD Macro facility allows the user to reuse or modify complex functionality stored in a canned form for later reuse. Unfortunately, it would go well beyond the intended scope of this paper to given any detailed description of these modes[2].

User Interface implementation using CHIDE

The MEAD graphical UI has been implemented using an environment provided by the CHIDE User Interface Management System[6]. The CHIDE (Computer-Human Interface Development Environment) provides a graphically based editor and run-time environment which together enable a developer to completely implement a user interface. The user interface is designed as a "wrapper" over one or more underlying applications as shown in Figure 14. CHIDE also provides the means to design a user interface independent of the target graphics terminal or workstation. CHIDE enabled the MEAD user interface to be implemented with the following features:

- Support migration from a Tektronix® terminal display to the X window system with minor changes to the user interface,
- Provide a "Point and click" , command line, and macro script styles of user interaction,
- Communicate to the MEAD supervisor through a command line interface ,
- Create complex forms from files retrieved from the MEAD database, and
- Support on-line and context determined helps.

The first implementation of the MEAD UI operates with a Tektronix 4107 terminal or a T4107 emulator running on any other terminal, IBM® PC or workstation. The overall design of the UI was made with a generic workstation in mind, and the general layout of the UI is closer to what is normally found on a workstation than on a "dumb" terminal. However, the size and resolution of the T4107 has somewhat limited the complexity of the individual screens. Also, the limited character fonts have precluded the use of advanced workstation features such as graying out of selections. The remainder of this section describes four novel features provided by CHIDE.

"Loosely Coupled" Interface: Most "point-and-click" style interfaces are developed with a "tightly coupled" interface with the application program. The user interface software, which may be implemented by means of a UIMS, is bound into the same executable code as the application which it manages. An alternative approach is a "loosely coupled" interface which is developed at the same time as the command line driven application that it drives. As shown in Figure 14, CHIDE provides a "wrapper" program which is a separate process from the application and which communicates with the application via ASCII messages through a mailbox or socket inter-process communication mechanism. Existing applications can remain unaltered and new applications, such as the MEAD supervisor, can be maintained independently of the user interface. The command line interface, which accepts ACSII commands and returns ASCII prompts, errors, or data, continues to be supported by the application. When connected to the user interface, the input and output channels are redirected to the UI process, and the application can **act** as if it were communicating directly with a user on an ASCII terminal.

State tree dialog control mechanism: The functionality of a UIMS can be partitioned into software components in several different ways[7] and is shown in Figure 15. CHIDE follows to some degree the Seeheim model which divides a UI into three components, a Presentation Manager, a Dialogue Control System and an Application Interface Manager[8]. The three components

can be viewed as the lexical, syntactic and semantic portions of the interface, respectively.

The Presentation Manager is composed of a window system, a UI toolkit and an event handler and deals with the appearance and user interaction with the UI. The Application Interface Manager handles the external interactions with the application. It translates user events into text which is meaningful to an application, and decomposes application responses into information meaningful to the user and Dialogue Control System. Between these two components is the Dialogue Control System which manages the flow of control in the interface. For example, if we consider a drop-down menu, the Presentation Manager allows us to create, display, and receive events from the user, while the Dialogue Control System describes when the menu is displayed and what should happen when each menu item is selected. The Application Interface Manager determines what resulting information, if any, is sent to the application upon menu selection. The Application Interface Manager utilizes event translation to process a response which is parsed in order to update the state of the interface, handle errors and to display the results of an operation.

In CHIDE a state tree model is used in the Dialogue Control System. The state of a UI at a given time defines the context for user and application interaction. The state tree model organizes UI states in a tree where event traps and action routines are attached to the states in the tree. A complete description of this model has been defined by Rumbaugh[9]. A structured state machine such as a state tree, provides many advantages as the basis of the dialog control system as summarized below:

- User state diagrams are a powerful tool for designing user-friendly products and systems[9]. The state tree model provides a way to move a state driven UI design directly and simply into the UI implementation.
- Structure is imposed on the state, events and action routines, making an interface more modular, easier to understand and easier to modify.
- The state tree provides a convenient way of graphically programming the Dialog Control System. The restructuring of an interface can be accomplished simply by repositioning sub-trees in the state tree hierarchy.
- The state tree organizes and reduces the complexity of the Application Interface Manager. The control aspects of an interface are specified in the state tree and not in code. Action routines are more likely to be simple and reusable. Few, if any, existing UIMS models address this issue.

Database Storage of UI specification: CHIDE also differs from other UIMS's in that CHIDE generates and manipulates a UI specification as data in a database, rather than as code in a programming language. To the joy of some, and dismay of others, no code is generated, modified, compiled or maintained when CHIDE is used to create a UI. This means that **all** the code within CHIDE is application independent. Only the database changes from one application interface to the next. This approach has two useful side effects.

First, the UI developer has on-line reconfigurability. That is, any changes or additions made to the interface are immediately executable. No code needs to be changed, recompiled or linked before the changes are tested. This capability has been demonstrated in other UIMS's which incorporate a language interpreter into their environment[10]. An interpreter also allows a UI developer to try out new or modified portions of an interface before compiling and linking, but requires the UI developer to understand the interpreted language which is being used.

The second side effect of storing the interface as data is the ability to modify the interface at run time. Existing UIMS's generally limit the changes which are allowed at run time to the display or "lexical" portions of the UI. CHIDE allows syntactic and semantic portion of a UI to change at run time since all the UI specification is maintained as data[11]. There is also nothing which precludes supplying the end user with the facility to modify an interface.

Interactive Development Tool: GUIDE: CHIDE interfaces are created and modified by editing graphical and tabular representations of the UI. This editing updates the databases which describe the interface, as explained in the previous section. Like some other UIMS systems, CHIDE provides WYSIWYG (what-you-see-is-what-you-get) graphical editing of the display objects, such as menus, panels, dialog boxes etc.[10,12,13]. The user deals only with look and feel of these objects. No knowledge of the underlying presentation and user interaction mechanism are assumed.

Unlike all other UIMS's observed to date, CHIDE also allows a non-programming description of the control flow and application interaction portions of the interface. See the Lohr paper[14] for a more complete description of these facilities. The complete structure and control flow of the interface can be graphically edited because the Dialogue Control System is based on the state tree model. A state tree editor is provided which shows the structural connections between the states and the flow of control which occurs for each user event. States and subtrees may be created, deleted and copied. The UI's structure and control flow can be modified by graphically manipulating the displayed structure and control arcs. In addition, each state may be "opened" to show its contents, i.e., the action routines executed when a state is traversed.

The Application Interaction Manager portion of the interface is defined in a format which avoids use of a programming language. Instead, the approach is to represent the flow of data to and from an application as passing through a series of interchangeable filters. A library of filters are provided which perform standard parsing, arithmetic and string manipulation functions. These filters work from a common input and output steam and thus, may be strung together in different combinations to accomplish complex processing of data. This approach is sufficient for most cases because the state tree dialog control mechanism greatly simplifies the action routines. For the more complicated, non-generic cases, the application developer will have to extend the library to provide a new filter. Simplicity has been gained at the expense of complete flexibility provided by using a programming language.

Development steps of the MEAD User Interface

In this section the steps which were taken by the project team in developing the design specification for the MEAD user interface are described. Through the several stages in the project a series of prototypes of the MEAD system were used to develop a design specification. In the beginning, the style of interface and the needs of our class of user, control engineers in GE, had to be discovered through the use of several prototype programs.

A prototype of the user interface enables an understanding of the actual user needs more so than through a specification document. The initial prototype of MEAD was implemented using lisp programs under GNU Emacs and provided a mock-up of all the major components, including the user interface, to be implemented in the MEAD project. This prototype was intended only as a vehicle to establish the functionality required for MEAD. For the user interface, the structure of the menus and the method for accessing the database through browse screens was identified. At that time we did not attempt to establish the style of interaction. Next, a HyperCard® program mock-up of the user interface was developed which concentrated only on the look and feel of the menus and forms (as opposed to establishing the underlying functionality). The HyperCard mock-up became a living specification which was refined over several months of evaluation by all parties involved.

An alternative to using a HyperCard mock-up might have been immediately to implement the user interface with a UIMS tool set. However in our case, we desired a platform, the Apple® Macintosh® computer, which was freely accessible to all parties involved and which enabled easy experimentation by those people. Also, to simplify building the mock-up the interface to the application (the MEAD supervisor) was only simulated through scenarios. Thus we did not require any of the other functionality of MEAD to be implemented on the Macintosh. It is important to note that by simulating the interaction to the application we were able to concentrate solely upon issues which were important to the users.

From the HyperCard® mock-up two documents were developed: a style guide and an application interface specification. While sometimes a User's Manual is also developed at this time, it was felt that the HyperCard mock-up was equivalent for our purposes. The style guide captured the essential look and feel desired for the MEAD user interface and took into account the constraints of the expected platform - a Tektronix® terminal or IBM® PC. Importantly, the specification of the interface between the MEAD supervisor and user interface interface was developed before implementing the UIMS based user interface. Because of the "loosely coupled" architecture, the specification established the set of commands planned for the supervisor and the kinds of responses which might occur from the supervisor .

The purpose of the style guide is to provide guidelines to the developer of the user interface which establish the appearance and operation of the user interface. The style guide provides a statement of desired attributes of the user interface and is divided into four sections. The first section establishes interface assumptions which would include hardware, software, human and performance characteristics. Next, usability principles are stated which in this document summarize accepted practice[15]. The third section describes the standard of style for the MEAD application. Finally, a strategy for screen layout is identified along with major display components which make up the screen layout. This was a particularly important need since the first release of the MEAD user interface was intended for a Tektronix® terminal display.

[™] HyperCard is a trademark of Apple Computer, Inc.

Conclusions

MEAD represents a new, more supportive environment for computer-aided control engineering (CACE). This paper has shown the flexible user-friendly user interface of MEAD, which is based on a "point-and-click" interactive mode and complimentary direct command modes. It has been shown how a single, consistent user interface may give access to the full range of modeling, analysis and design functionality of a complete CACE environment.

Also, this paper has described the concepts of an experimental UIMS which is based upon the X window architecture. The architecture of the UIMS is further described as is implemented for the MEAD user interface. The user interface has been successfully operating with Tektronix® terminal version for about a year and in 1989 has been moved to function with the X window system under the UNIX® operating system.

Acknowledgment

The authors would like to acknowledge the contribution by Michael Charbonneau who took on the task of using the newly developed GUIDE graphical editor to create the MEAD user interface.

References

[1] Taylor, J. H., Frederick, D. K., Rimvall, C. M. and Sutherland, H., "A Computer Aided Control Engineering Environment with Expert Aiding and Data-Base Management". Proc. IEEE Workshop on Computer-Aided Control System Design, Tampa, FL, 1989.

[2] Rimvall C. M., "A Data-Driven Command Interface and Translator for CACE applications", to be published.

[3] Taylor, J. H., Nieh, K-H, and Mroz, P. A., "A Data-Base Management Scheme for Computer-Aided Control Engineering", Proc. American Control Conference, Atlanta, GA, 1988.

[4] Schmid, C., "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", pp. 159-180, North Holland, Elsevier Science Publishers, 1985.

[5] Moler, C. MATLAB User's Guide. Dept. of Computer Science, University of Albuquerque, NM, 1980.

[6] Lohr, P.J., "CHIDE: A Usable UIMS for the Engineering Environment". Tech. Report, GE Corporate Research & Development, Schenectady, NY, 1989.

[7] Green, M., "A Survey of Three Dialog Models", Trans. on Graphics, ACM, Vol. 5, No. 3, July, 1986.

[8] Green, M., "Report on Dialogue Specification Tools", User Interface Management Systems, edited by Guenther E. Pfaff, Springer-Verlag, Berlin, 1985.

[9] Rumbaugh, J., "State Trees as Structured Finite State Machines for User Interfaces", ACM SIGGRAPH Symposium on User Interface Software. Banff, Alberta, October 1988.

[10] Foody, M., "UIMX: A User Interface Management System for Visualization in Scientific Computing", unpublished paper, Visual Edge Software Ltd., St. Lautent, Que., Canada, 1988.

[11] Hardwick, M., et. al., "Advantages of Using Database Technology in a User Interface Management System", 1989 (submitted for publication).

[12] TAE Plus and TAE Plus Workbench Documentation (Version 3.11), Century Computing, Inc., Laurel, MD, 1988.

[13] Open Dialog UIMS Product Literature, Apollo Computer Inc., Chelmsford, MA, 1988.

[14] Lohr, P.J., "CHIDE: A Usable UIMS for the Engineering Environment", Proc. 10th GE Software Engineering Conference, Daytona, FL., April, 1989.

[15] Shneiderman, B.,"Designing the User Interface: Strategies for Effective Human-Computer Interaction", Addision-Wesley , 1988.

[16] Taylor, J.H., McKeehen, P.D., "A Computer-Aided Control Engineering Environment for Multi-Disciplinary Expert-Aided Analysis and Design (MEAD)". Proc. National Aerospace Electronics Conference (NAECON), Dayton, Ohio, May 1989.

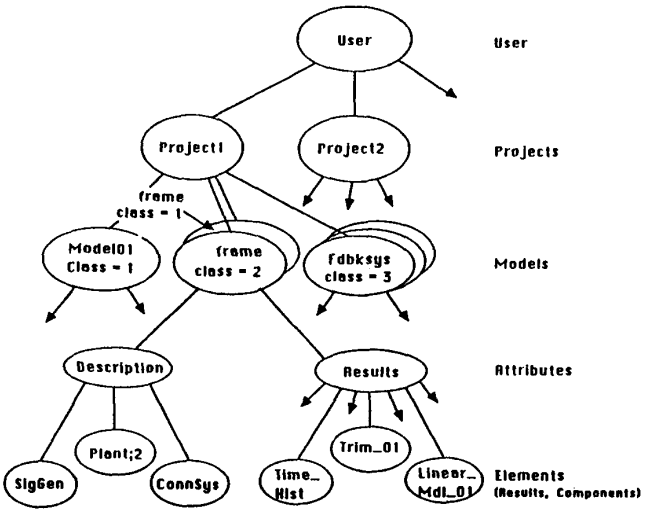

Figure 1. The MEAD Data Base relationships.

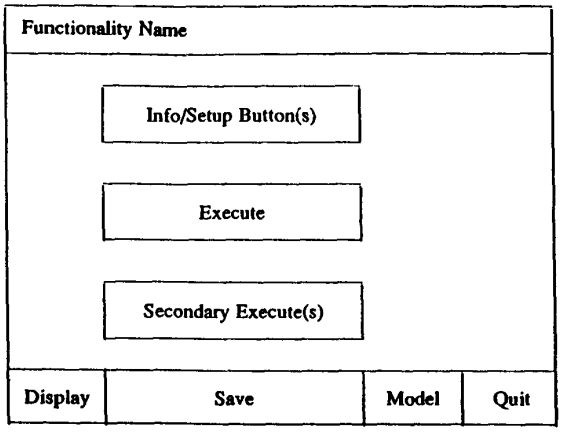

Figure 2. Principal layout of a MEAD *action form*.

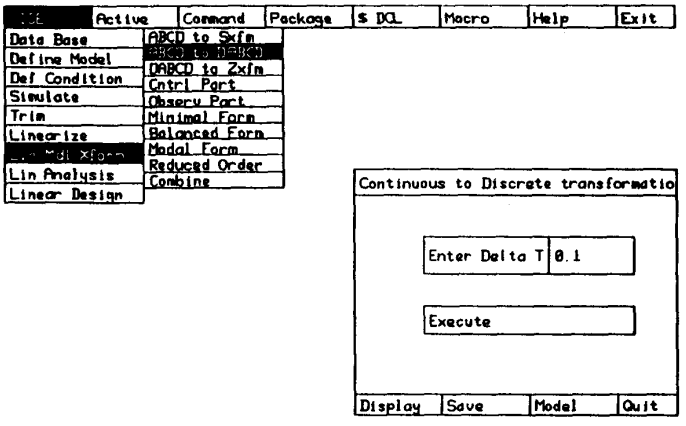

Figure 3. The MEAD *resource bar* and the main IDE menu operating environment. A "continuous-to-discrete" action form is also open.

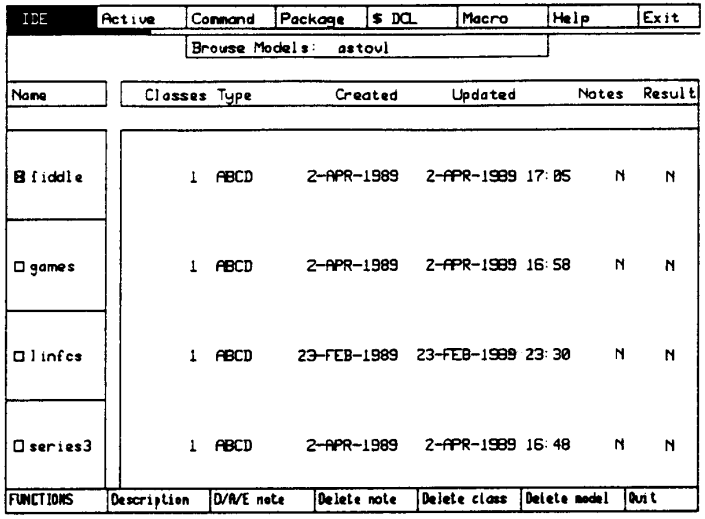

Figure 4. A typical MEAD *selection form* - the Browse Models form (DBM version).

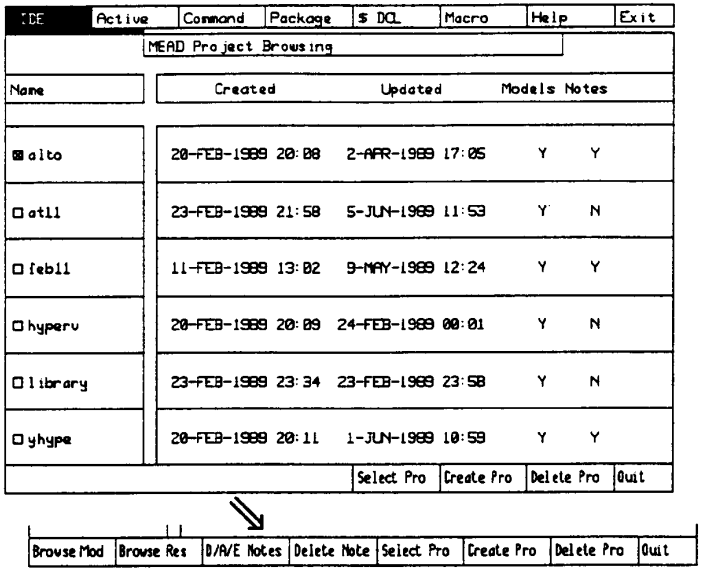

Figure 5. Startup Browse Projects screen with abbreviated menu selections. The alternate full menu selection is shown on the bottom.

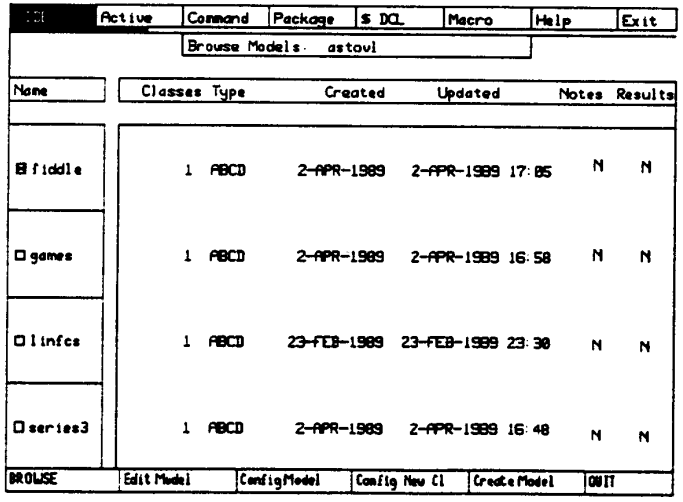

Figure 6. Browse Models form (configure version)

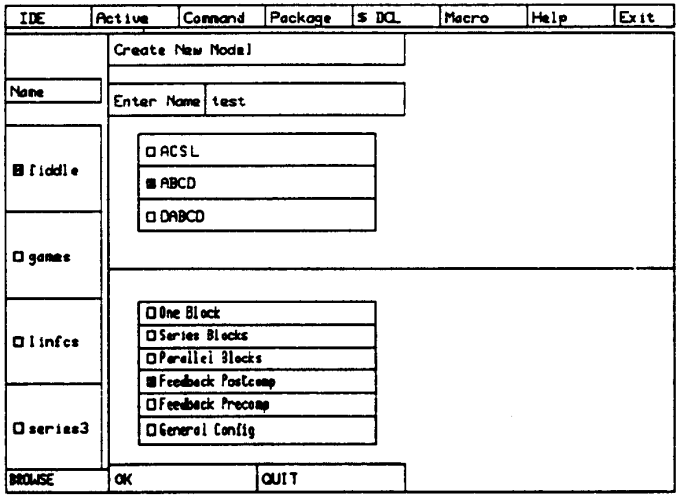

Figure 7. Main forms for creating a new linear model.

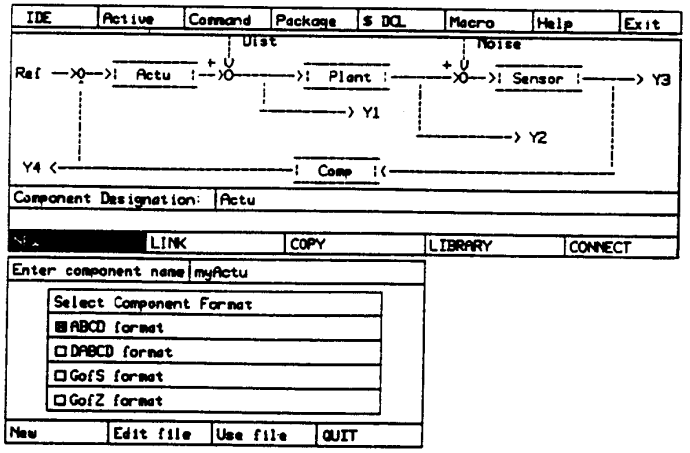

Figure 8. System connection architecture form and form for entering individual components.

```
function [A,B,C,D] = comp
A = [ 0  1  0
      0  0  1
     -1 -2 -3];

B = [ 0;
      0;
      1];

C = [ 1  0  0];

D = [ 0 ];
```

Figure 9. Edited file input representing a linear component in ABCD form

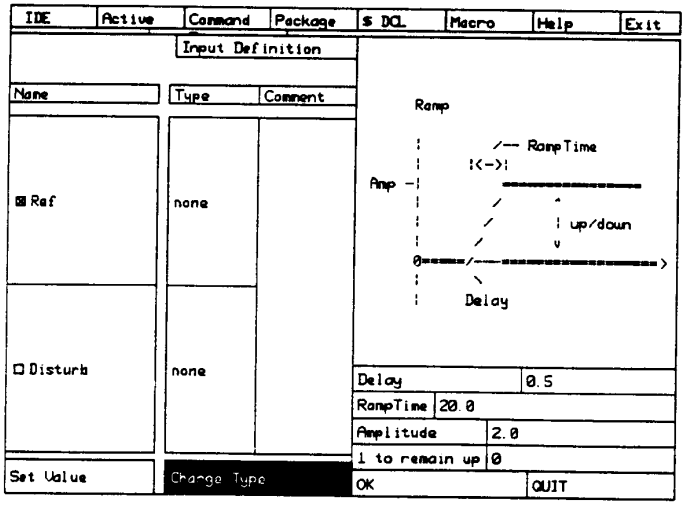

Figure 10. Forms for assigning signals to system input connections.

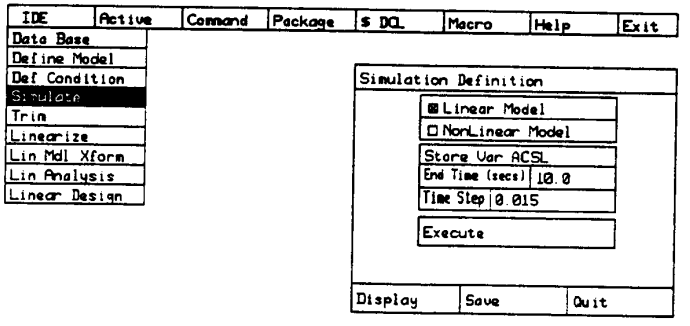

Figure 11. Simulation form

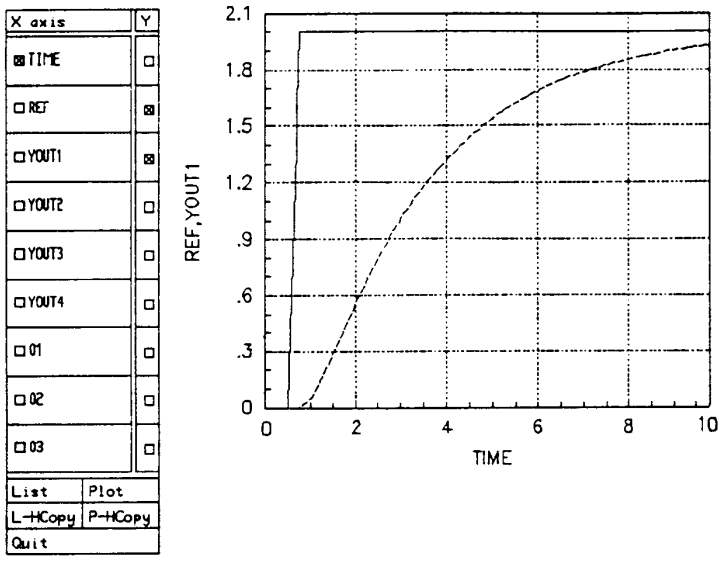

Figure 12. Display form

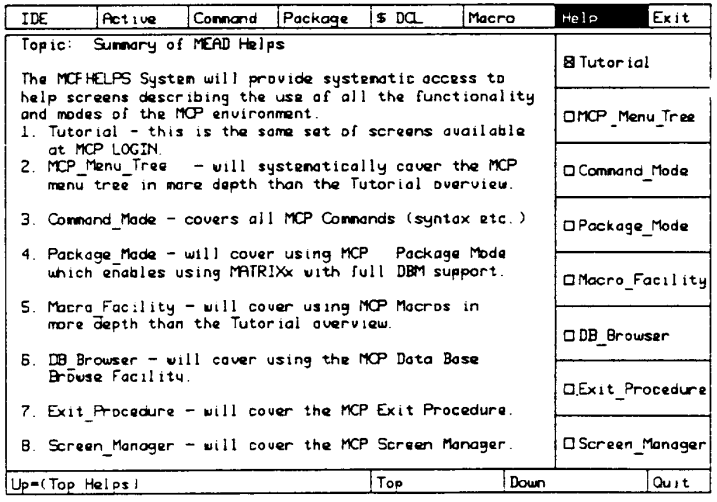

Figure 13. Help form

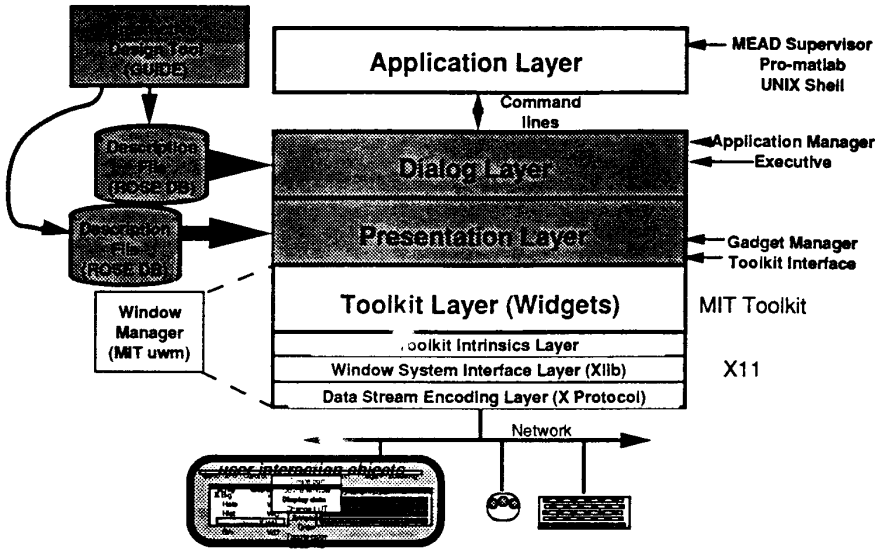

Figure 14. CHIDE wrapper around Applications

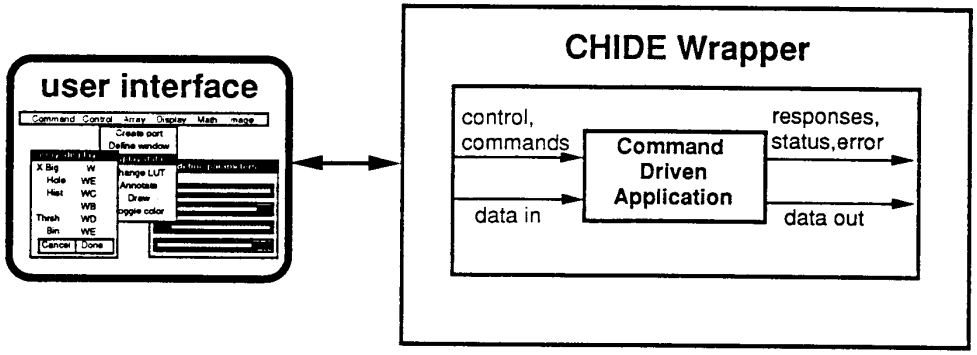

Figure 15. MEAD User Interface Architecture under X Window System