Chapter in *Recent Advances in Computer-Aided Control Systems Engineering*, Ed. by M. Jamshidi and C. J. Herget, Elsevier, 1992.

Data-Base Management for Computer-Aided Control Engineering

James H. Taylor

ORA Corporation, Ithaca, New York 14850-1313 USA

Abstract: There has been substantial progress made in the past decade in the development of analysis and design software for computer-aided control engineering (CACE). Engineering data-base management (EDBM) for support of CACE has not received much attention until comparatively recently, however. As CACE environments become more comprehensive and powerful, and as CACE comes to play an important role in concurrent engineering practices, the need for keeping track of the models, simulations, analysis results, control system designs, and validation study results over the control system design life-cycle becomes more pressing and the lack of EDBM support becomes more of an impediment to effective controls engineering.

A CACE environment with EDBM as an integral part has been defined and implemented in a project conducted at GE Corporate R&D called MEAD. The data base is organized in a hierarchical framework having the levels ***Project***, ***Model***, ***Attribute***, and ***Element***. The level ***Project*** accommodates the control engineer's natural tendency to group or organize results according to a "project" such as Flight_Control, Power_System, . . . or perhaps more specifically F-16, F-18A, . . . Within a project, ***Models*** (plant models, control system models, etc.) are the main focus. Each model has two attributes, a ***Description*** and a ***Result_set***. Fundamental properties plus component models (representations of a plant, compensator, sensor, etc.) comprise the elements of a Description; elements of a Result_set include any data generated with the model, such as a time-history, frequency response, LQR/LTR design, etc., plus the information required to define the result (parameters, algorithmic options, etc.).

The CACE DBM problem is complicated by the fact that system models usually evolve as a project progresses. It is thus necessary to keep track of which results have been obtained using various instances of a model. In our EDBM system, each model in the hierarchy is maintained using a ***version control scheme*** so that results obtained with each instance of the model can be unambiguously associated with that instance, thus maintaining data-base integrity. Two other complications preclude the use of a simple tree structure for the hierarchy outlined above:

1. Linearization of a nonlinear system produces both a ***Result*** and a ***Component*** that can be treated as a model, analyzed and have a ***Result_set*** of its own; this relation is maintained by use of a ***Reference***.
2. Maintaining a component that may be used in a number of models (e.g., the subsystem **Plant** may be used in the models **Plant_Alone** and **NL_Fdbk_Syst**) without a proliferation of copies requires a second mechanism we call a ***Link***.

All of these situations are managed in the CACE data-base schema presented below.

1. INTRODUCTION

Software environments for computer-aided control engineering (CACE) have been in a stage of rapid development over the past decade. Primary emphasis has been placed on the following aspects:

- improving or extending core CACE capabilities, i.e., nonlinear simulation, identification, and linear analysis and design in the frequency and time domains;
- integrating core CACE functionality (e.g., nonlinear simulation with linear analysis and design);
- improved interactive user interfaces;
- introduction of artificial intelligence support or expert aiding; and
- migration to engineering workstations and personal computers for hosting part or all of a CACE environment.

Given these developments, it is surprising that support for engineering data-base management has lagged so far behind. As CACE environments became more powerful in terms of systems analysis and design capabilities and broader in terms of the variety of CACE activity supported, the need for keeping track of the models, analysis results, control system designs, and validation study results has become more important and obvious. In a real industrial control system design project, the large number of files generated in the complete design life-cycle and the relations among these files can be very difficult to comprehend and manage by manual means. Engineers can successfully manage their data bases, but the effort and discipline required to manually maintain and document the files, listings, hard-copy plots, etc. that arise over the design cycle can be prodigious. It is often the case that, after the passage of a few months, the engineer cannot say exactly how a given result was obtained and thus cannot reproduce, document or defend it.

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 was 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 MATLAB or the '.d', '.t' scheme of SIMNON. One early package (CLADP, [1]) generated filenames by appending characters to a user-supplied 'Run Name'. All such support was very rudimentary and left 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 EDBM functionality with a CACE environment appears to be in Bunz and Gutschow [2]. However, the ideas of the complete hierarchy of projects, models, components, and results and of version control were not discussed.

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.flt_ctrl.harrier]**! In addition, there are support functions such as on-line documentation that can add substantially to the value of the EDBM.

These needs and considerations were factored into the design of the MEAD EDBM, which was developed from first principles. These also motivated us to define and implement the EDBM as an integral part of the CACE environment. Specifically, we felt that a simple "add-on" EDBM would not be able to make the required associations without user-supplied information that would add unnecessary overhead and discourage the use of the EDBM. Note that we made the pragmatic design 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.

Rigorous data-base management requirements for CACE were presented in Taylor [3] and in Taylor, Nieh, and Mroz [4]. 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; the latter was selected. An effective user interface for such a system is described in [5]. 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 properly maintaining model components that are used in more than one model. The complete system is described below.

The remainder of this article is organized as follows: An overview of the entire MEAD CACE environment is provided in Section 2, the magnitude of the CACE problem is defined in more detail in Section 3, Section 4 provides a formal statement of the solution, and Section 5 describes the MEAD user interface as it pertains to EDBM operations. Future EDBM refinements and extensions are discussed in Section 6, and summary and conclusions are presented in Section 7.

2. MEAD CACE ENVIRONMENT OVERVIEW

The GE MEAD Controls Environment (Taylor and colleagues, [6, 7, 8]) has been designed to address the support and environmental issues outlined above while taking 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; Taylor and McKeehen [9]). The basic elements of MEAD systems are:

- a multi-modal User Interface (Rimvall and colleagues [10]) that supports all basic CACE activity within a point-and-click menu- and forms-driven environment (see Figs. 7, 8 later in this presentation) and also provides other access modes for the more experienced user (command-driven modes and a macro facility),
- an EDBM (Taylor, Nieh and Mroz [4]) 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 (Taylor [11]; this is only working in USAF MEAD), and
- a data-driven Supervisor (Rimvall and Taylor [12]) 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") include the MATLAB[TM] package for linear analysis and design, and the SIMNON[TM] 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; Lohr [13]) which rests on the ROSE[TM] data-base manager; the MEAD EDBMS uses ROSE and the DEC[TM] Code Management System (DEC VAX/CMS[TM] [14]) for version control; and the expert system uses the GE Delphi[TM] shell which rests on VAX[TM] Lisp. The supervisor and the front-end of the DBM are coded in the Ada[TM] language. Note that this architecture and philosophy represent an instantiation of the Reference Architecture paradigm common to modern software package design methodology and particularized for CACE by Barker and colleagues [15]. Architectural design considerations and relations among the MEAD environment and other major CACE packages are described in further detail in [16].

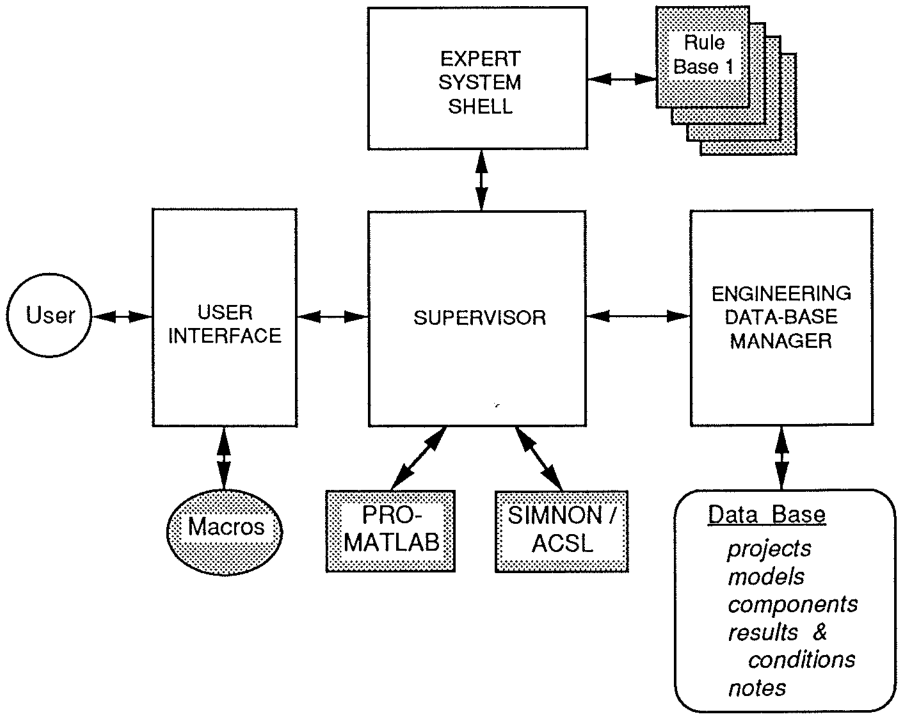


Figure 1. MEAD Architecture for CACE

In the course of developing, testing, and evaluating USAF and GE MEAD, we believe that substantial progress was made in supporting the controls engineer in a number of areas previously given little or no attention. In particular, the higher-level user interface, the data-base management scheme, and expert-aiding were noteworthy new contributions. The MEAD User Interface is much more "user-friendly" compared with those of the

[TM] MATLAB is a registered trademark of The MathWorks, South Natick, MA; SIMNON is a trademark of Lund University, Lund, Sweden; DEC and VAX/CMS are trademarks of Digital Equipment Corp., Maynard, MA; ROSE is a trademark of Martin Hardwick, RPI, Troy, NY; Delphi is a trademark of GE; and Ada is a registered trademark of the U. S. Government, Ada Joint Program Office.

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 and information with rigor - in essence, every element in the user's MEAD data base is fully documented. The MEAD expert system adds yet another area of support, which at this point has not been used to full advantage.

As is usually the case, we have also learned that much more can be done, and that certain things might better be done differently. The focus of the rest of this presentation will be the MEAD EDBM concepts and implementations, and on those areas where it can be modified, extended and improved. A more general discussion of future MEAD enhancements may be found in [7, 8].

3. THE CACE EDBM PROBLEM MAGNITUDE

We indicated above that EDBM is important. The magnitude of the control engineer's DBM problem can best be appreciated by looking at CACE in a larger context than linear analysis and design. In many cases, the user starts with a nonlinear model of the process to be controlled, and progresses through the following range of CACE activity: nonlinear simulation (e.g., model validation and behavioral analysis), trim or equilibrium determination (defining operating points), linearization, linearized system analysis and design, nonlinear control system design, and control system validation using both linear and nonlinear models of high fidelity [17, 18]. Much of this activity is exploratory and iterative in nature. Several systems have been developed that cover most or all of this gamut - Ctrl-C + Model-C, MATLAB + SIMULAB, and MATRIX$_X$ + SystemBuild are well-known commercial examples†. However, none of these managed the resulting data base beyond implementing rudimentary file-naming conventions.

In many applications, control engineering activity develops a substantial data base. In flight control, for example, a typical data base may contain one nonlinear airframe model, 20 linearized models (corresponding to 20 points in the flight envelope), 20 linear control system designs, one or several candidate nonlinear ("full-envelope") control systems, and innumerable time-histories and analysis results (step responses, equilibria, eigenvalues, frequency responses, root-locus data, singular values, . . .).

The above data-base sketch may be multiplied many times over by two additional factors: First, in multi-disciplinary applications such as integrated flight and propulsion control, one can produce an enormous data base by combining 20 flight regimes with 9 engine operating conditions, for example. Second, it is not unusual for the primary nonlinear model of the controlled object to be modified several times over the complete analysis and design cycle. We can thus 'size' the EDBM problem by assuming three instances of each model and perhaps eight results per nonlinear or linear model on the average, giving us the final product 3 (instances) × 8 (results) × (1 + 20 + 20 + 2 + 1 + 9 + 9 + 1) (total models) = 1492 files to manage (here we take the best-case assumption that the 9 engine linearizations can be treated independently of the 20 airframe models - otherwise they would enter multiplicatively, not additively).

As the above cases make clear, CACE project activity can generate hundreds of results and model files, of which the user may wish to retain and manage a substantial percentage. This may not be a "large" data base in terms of commercial DBM systems, but it

† Ctrl-C + Model-C, MATLAB + SIMULAB, and MATRIX$_X$ + SystemBuild are trademarks of Systems Control Technology Inc., Palo Alto, CA; The MathWorks, South Natick, MA; and Integrated Systems Inc., Santa Clara, CA, respectively.

is neither small nor simple, and is thus difficult for most users to manage effectively without support.

4. CACE DATA BASE DEFINITION

As mentioned previously, the CACE user's data base is traditionally but informally organized in terms of ***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 = ***Combustor***, Component 4 = ***Fuel_injector***, ...), and which are used to generate various results (e.g., simulation time-histories, linearizations). This scheme has been accommodated directly in the basic MEAD data-base tree structure, as depicted in Fig. 2 [4]. This represents a refinement of the scheme outlined in [3], and was developed from a careful analysis of the informal organization that is traditional in the field plus knowledge of CACE work-patterns and data element interrelations.

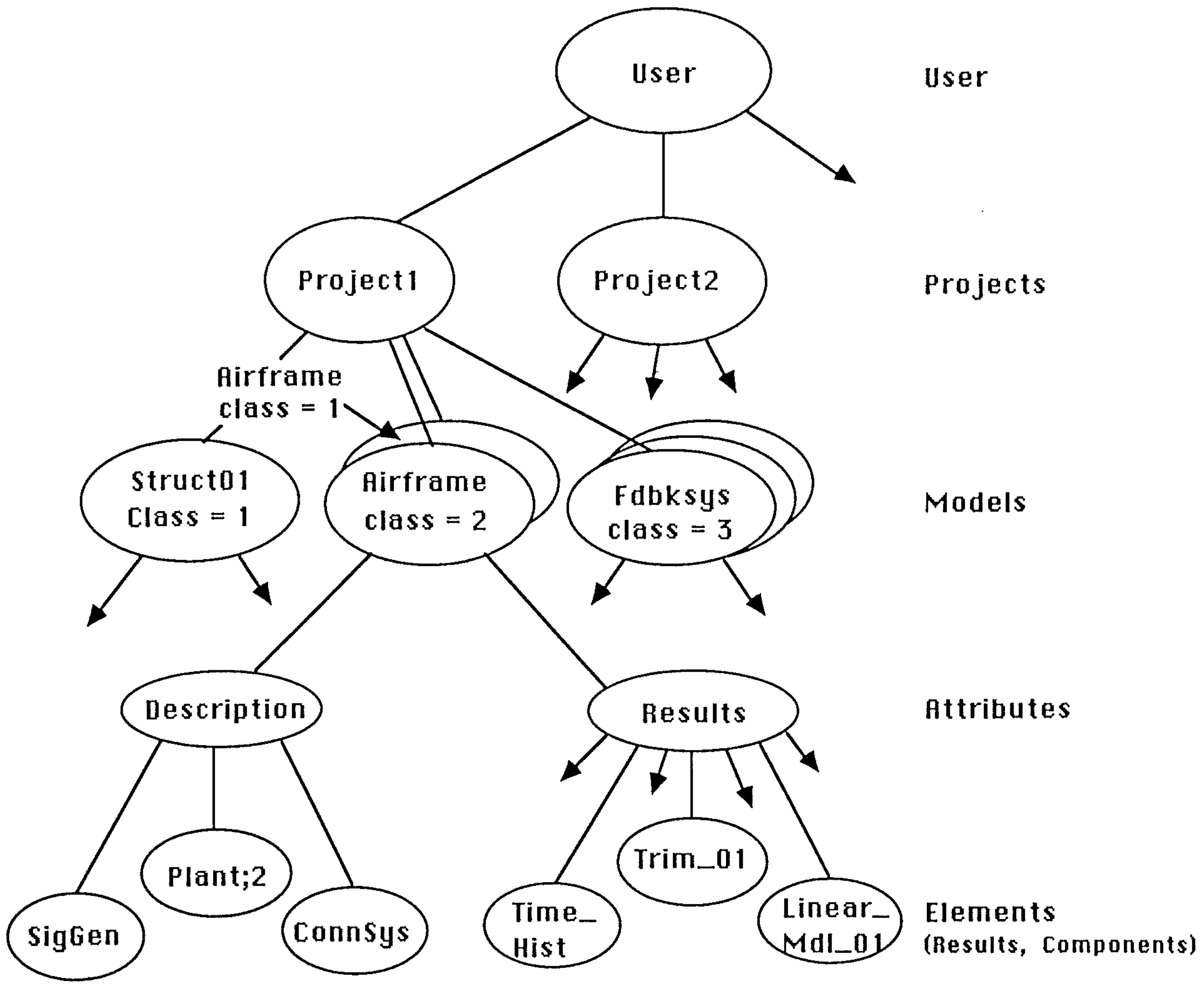


Figure 2. Hierarchical Data-Base Organization for CACE

All CACE activity within a project is carried out in the context of *Models* (plant models, control system models, etc.). Each model has two attributes, a *Description*, i.e., a catalog of the fundamental characteristics of the model (linear or nonlinear, continuous- or discrete-time or mixed, etc.) plus a list of the sub-systems or components that comprise the model and a definition of the sub-system interconnections; and a *Result_set*, i.e., the set of simulation, analysis, and design results obtained using the model. *Elements* of a Description characterize the system in terms of component models, e.g., representations of a plant, compensator, sensor, etc.; elements of a Result_set include any data item associated with the model, such as a time-history, frequency response, root locus, LQR/LTR controller design, etc., plus the information required to define the result (set-up parameters, algorithmic options, etc.). Note that we have not defined the data structures at the element level; in light of recent activity directed at establishing standards in CACE data structures [19-21] we decided, as an interim measure, to accept the data structures and formatting produced by existing CACE packages and incorporate data element "filters" to transform from one format to another where required.

The most difficult aspect of CACE DBM is maintaining the integrity of the data base in situations where system models evolve over the term of the project [3]. It is almost always true that plant models created early in a project have to be modified, either to reflect additional information (e.g., experimental results), changes in the design of the plant itself, or the discovery that the original model is not valid over the range of operation occurring in the validation and acceptance part of the design cycle. In addition, control system models evolve as the design is perfected. The loss of integrity in the sense of not knowing which analysis and design results were obtained with which instance of the model is the most common fault encountered in documenting the control system design cycle, and the outcome is irreproducible results, unsupportable design decisions, and unproductive reiteration.

The solution to this problem is to be found in the software engineering discipline of *version control*. In our system, each model in the hierarchy is maintained using a version control scheme such as found in the DEC product VAX/CMS [14], so that system models can evolve and results obtained with each instance of the model can be correctly related with that instance. Specifically, each instance of the model is identified by a *name* and *class number:* <model_id> = <model_name> + <class_number>. The Description of each model instance is maintained in relational (tabular) form in terms of the *Version* for each *Component model;* this is illustrated in Fig. 3. Components are maintained in the form of file names pointing to the actual data elements that are maintained in CMS libraries. Any version of a component can be obtained using a CMS Fetch command; therefore, any model instance can be assembled for documentation or further analysis. Note that CMS provides for very efficient storage of the component versions in one file using a 'difference' scheme.

Other problems associated with data-base integrity are addressed by the MEAD EDBM as follows:

- 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, if *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* (which is a bi-directional pointer) and *Condition_Spec* (associated with the result in *Turbine class=7*'s data base).

Description:	NL_Fdbk_Syst
System_Type:	Nonlinear
Time_Type:	Mixed Cont-/Discr-Time
Component_set:	Airframe, Ctrlr, Sensor
Connection_Def:	NL_Fdbk_Conn

Component →	Airframe	Ctrlr	Sensor	Note
Class:	Version	Version	Version	
001	001	001	001	Yes
002	001	002	001	No
003	002	003	001	No
004	002	004	002	Yes
005	003	007	002	No

Figure 3. Model Description in a MEAD Data-Base

- Single-point storage of components is provided for sub-systems 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 the component stored in ***Turbine***. The idea of a data-base link is adopted from UNIX^{TM}; here, we mean that the same CMS Library file location of the component **Plant** is used in any Description of any model containing it. The Link mechanism ensures that sub-system models can be maintained with integrity.

Both of these features are portrayed in the detailed CACE data-base schema in Fig. 4.

The importance of the Reference feature can be illustrated by considering the flight control systems design scenario outlined in Section 2: Assume that **Plant** in Fig. 4 represents a nonlinear model of an aircraft's aerodynamic behavior. In order to achieve a "full-envelope" control system design, the standard practice is to define several dozen flight regimes for different altitudes and Mach numbers, linearize at each flight condition, perform a linear analysis and design at each point, and combine the set of linear designs via gain scheduling. In this scenario, there exist perhaps 20 linear models that are used to generate substantial Result_sets of their own; if it is not possible to trace each linear model back to its ***Parent Result*** (Result_set entry for a particular instance of the nonlinear plant model and corresponding to a particular set of conditions such as operating point), then documenting the design and even performing the gain scheduling part of the design may be impossible. In the general case, the reference mechanism is the key to relating linearized models and their associated results with their origin (<model_id> *and* operating regime) with integrity.

The Link concept is used to solve another integrity problem: model proliferation. It is difficult enough to maintain and track one particular component model as it is refined and edited; if several copies of the model exist and have to be maintained as separate entities by manual means, then the problem is seriously compounded. Our EDBMS solves this problem by maintaining only one copy of each sub-system model. Every

TM UNIX is a trademark of AT&T.

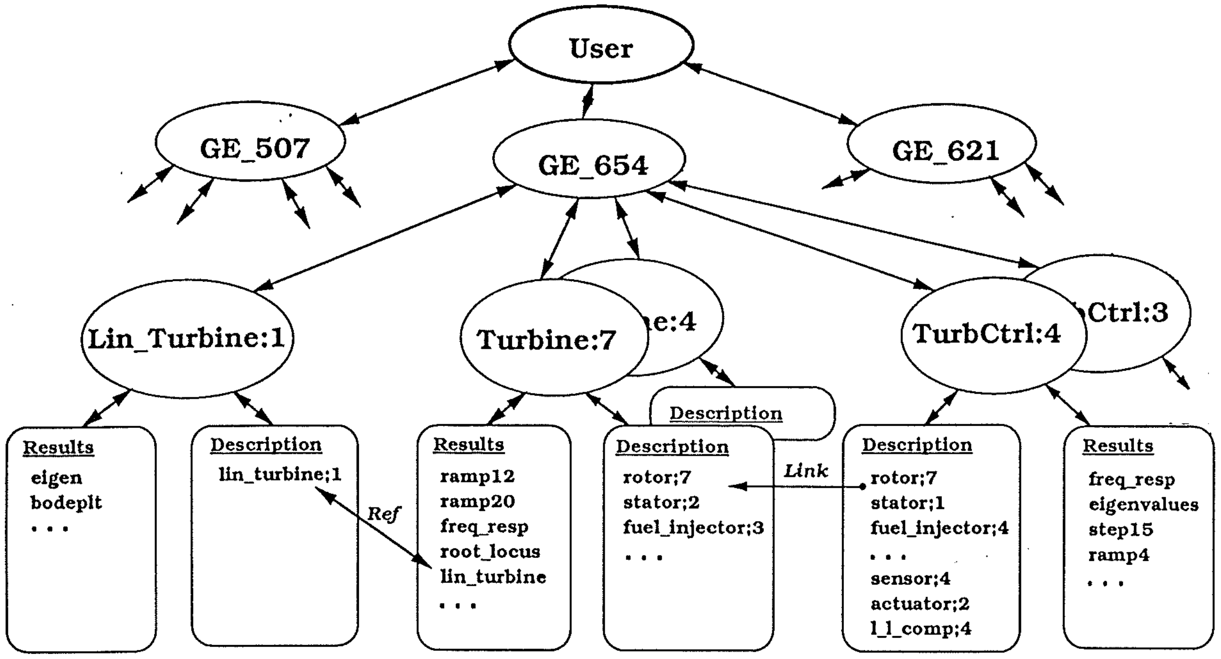


Figure 4. MEAD Data-Base with Reference and Link

time it is modified, a new version is created, as mentioned above; each model's <model_id> is tied to a specific version of each component. Note that each model using a linked component may or may not have an instance corresponding to each sub-system version, as illustrated in Fig. 5. While the component is associated with one model, the fact that there can be sub-system model versions that are unique to any other model provides no loss of generality. The main care that must be taken in treating linked components relates to purging model classes and deleting entire models; for example, if the model "owning" a linked sub-system is deleted, that component must be transferred to another "owner" model. The MEAD EDBMS performs these activities automatically.

The lowest level of the hierarchy below *Description* is directly evident from Fig. 3. The components of a model exist and are managed in CMS library files that contain code appropriate to the modeling language(s) supported in the CACE software for simulation, analysis, and design. Therefore, we have not provided examples of these data elements. The elements under *Results_set* are less traditional in their organization, as shown in the result tabulation in Fig. 6. The result itself (whose name and type are indicated in columns 1 and 2) is clear enough; it corresponds to a file containing time-history data, eigenvalues, etc. The next item is the *Condition_Spec* entry in column 3 of the table. This is required to track the one remaining factor that governs the integrity (reproducibility, etc.) of the CACE data base, i.e., the conditions under which the result was obtained. This is especially important for nonlinear aspects of CACE, such as simulation, equilibrium determination, linearization, etc. where the user may specify arbitrary initial conditions, input amplitudes, and parameter values, and use various algorithms and specifications (e.g., integration algorithms, tolerances and iteration limits). This information is maintained in a separate file that the EDBMS tracks and associates with one or more item in the Result_set, as illustrated. Finally, the date the result was created and the Yes/No flag indicating the presence of notes associated with each result (indicated in last two columns) are also straightforward.

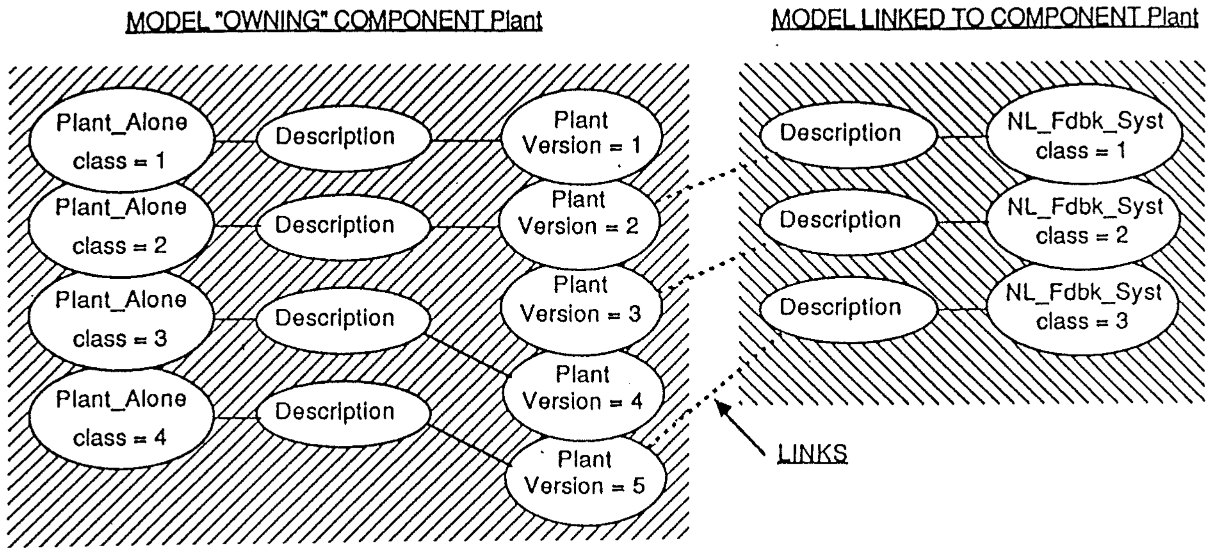


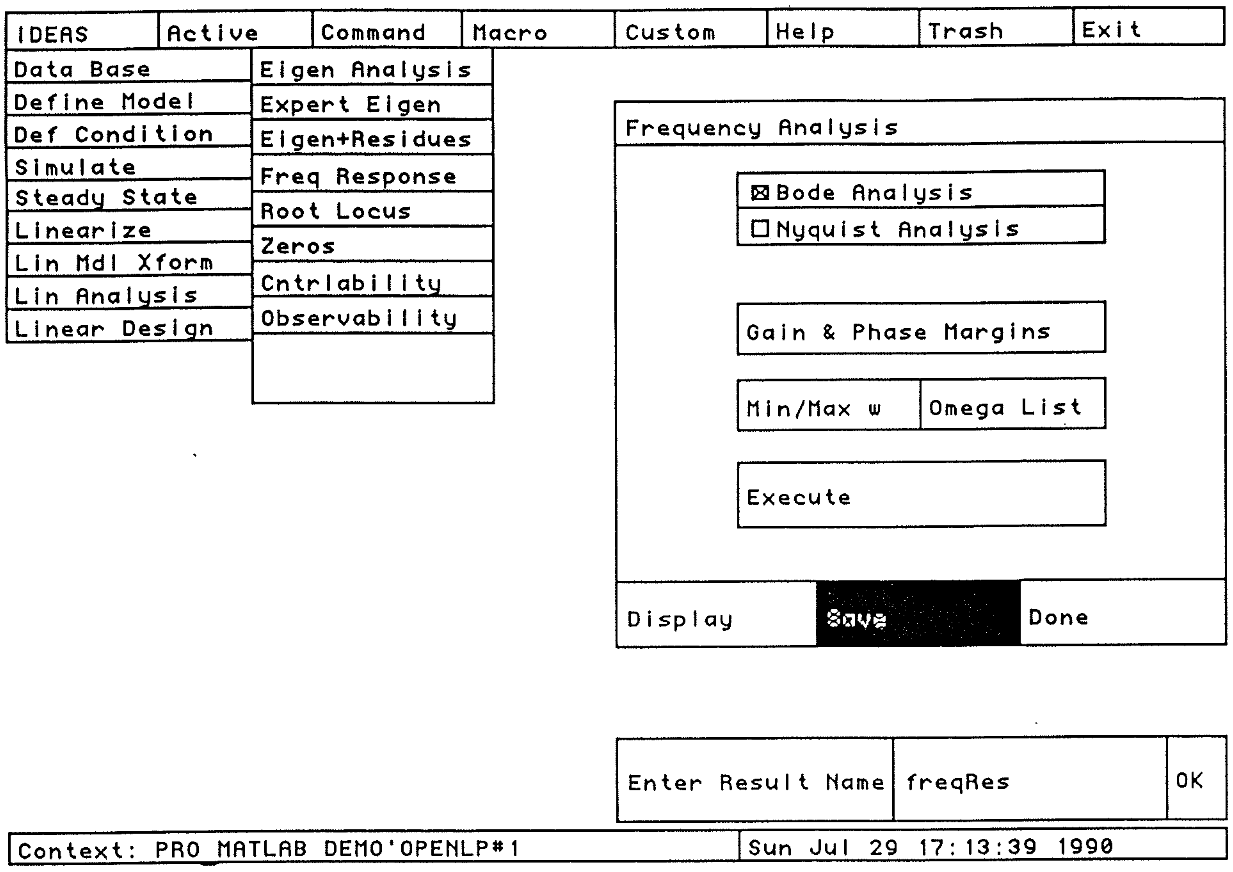
Figure 5. CACE Models with a Linked Component

Result_set:	model_id = NL_Fdbk_Syst + Class03			
Name	Type	Condition_Spec	Create_Date	Note
Cruise_20k_Mp6	Simulation	CS001	26-Mar-1991 09:44	Yes
Cruise_20k_Mp7	Simulation	CS002	27-Mar-1991 11:17	No
Cruise_20k_Mp7	Trim	CS002	27-Mar-1991 11:52	No
Cruise_20k_Mp7	Linearization	CS002	27-Mar-1991 11:57	Yes
Cruise_20k_Mp8	Simulation	CS003	27-Mar-1991 13:17	No
Cruise_20k_Mp8	Trim	CS003	27-Mar-1991 13:29	No

Figure 6. Result_set in a MEAD Data-Base

5. CACE EDBM USER INTERFACE

The motivation and design of the EDBMS are primarily based on considerations of data-base maintainability and integrity. It is important to note, however, that the acceptance of the EDBM is strongly influenced by the mental and operational overhead associated with its use. We have made every effort to minimize such overhead; for example, as shown in Fig. 7 (a literal screen dump), saving a result in the data base is simply accomplished by clicking on 'Save' and supplying a suitable name in the form that appears for this purpose. Furthermore, our experience has shown that a well-designed interface can even enhance the effectiveness of the CACE environment, by making unnecessary detail transparent to the user and by making certain operations more naturally "object-oriented", as outlined below.



☒ Bode Analysis
☐ Nyquist Analysis

Enter Result Name freqRes

Context: PRO MATLAB DEMO'OPENLP#1

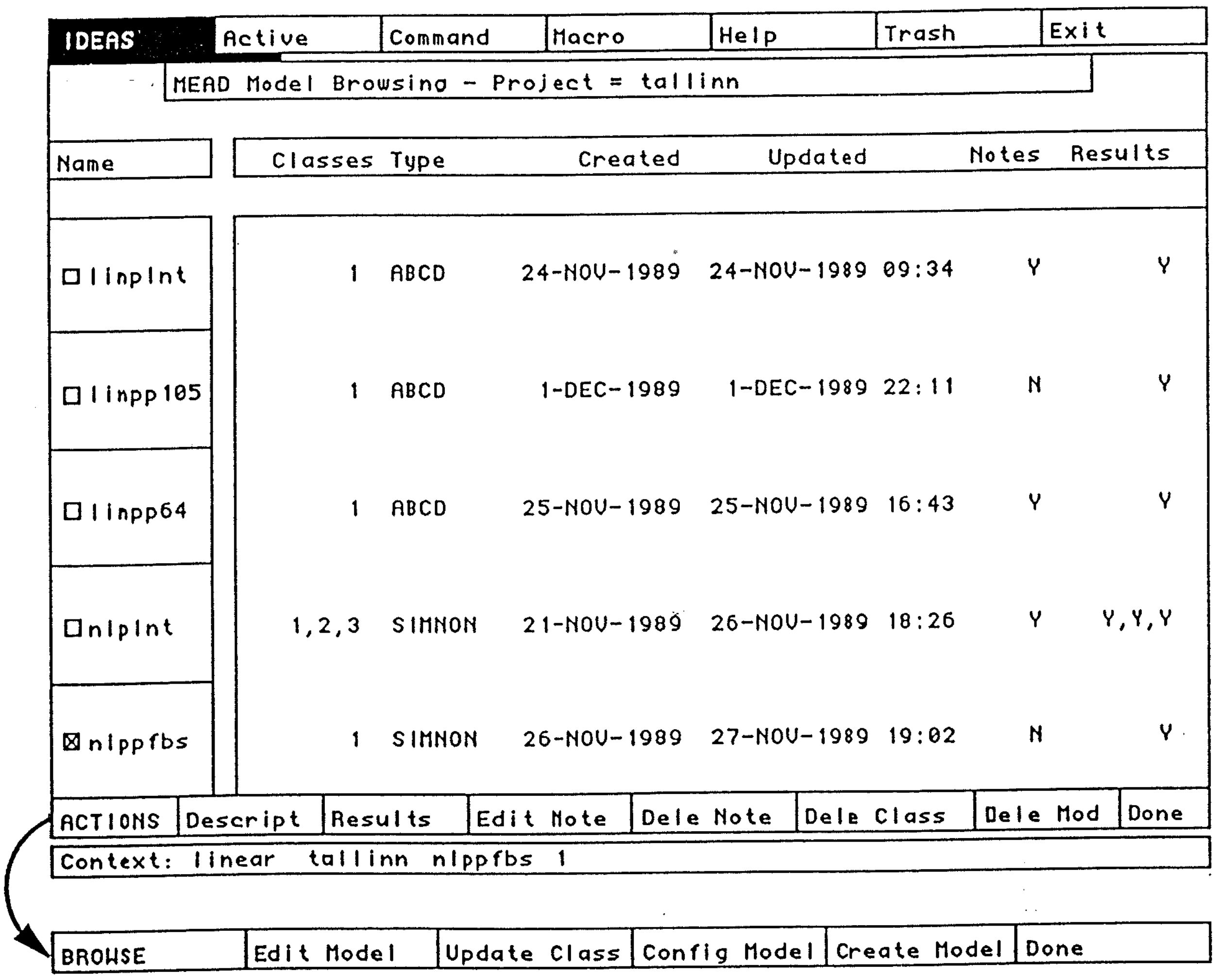
Sun Jul 29 17:13:39 1990

Figure 7. MEAD Simulation Form with 'Save'

Several examples of "information hiding" are implied in Figs. 3 and 6, where it may be observed that names are assumed to be supplied by the user wherever possible (e.g., NL_Fdbk_Syst, Cruise_20k_Mp6), while information such as file physical location (disk, subdirectory, name) is not displayed or needed. The file names are in fact based on file-naming conventions that create unique designations that encode project, model_id, component_id, result_type etc. and thus (as is often true with user-supplied file names as well!) difficult to remember - but only the EDBMS needs to keep track of this. Also, the user only needs to know whether or not a note exists (Yes/No in the last column of Figs. 3, 6); the EDBM can locate it for display if the user requests this action. An example of an objected-oriented action that is enabled under the EDBM's user interface is the configuration of a model (described below).

As mentioned previously, the MEAD data base is accessed by a Browsing Facility. Figure 8 depicts an EDBM Browsing Screen (also a screen dump), which shows that elements of the user's data base are accessed by point-and-click operations on a "selection form". This allows the user to display, annotate, purge and delete items very simply. More specifically, the most powerful browser is supplied for models, as shown in this example; the buttons across the bottom include: 'ACTIONS' (which brings up another row of buttons listed below), 'Descript' (which drops down for browsing at the Component level), 'Results' (which accesses results for a particular model instance), 'Edit

Note' (whereby the user can add or edit notes relating to the model), 'Dele Note' (to delete same), 'Dele Class' (to purge a specific model class), 'Dele Mod' (delete *all* instances of the model), and 'Done' (move back up to the Project level). The ACTION button replaces these with: 'BROWSE' (toggle back), 'Edit Model', 'Update Class' (once the user has edited one or more of the model's components, create a new model class), 'Config Model' (load the model in the appropriate core package for immediate use), 'Create Model', and (again) 'Done'.



MEAD Model Browsing - Project = tallinn

Name	Classes	Type	Created	Updated	Notes	Results
☐ linplnt	1	ABCD	24-NOV-1989	24-NOV-1989 09:34	Y	Y
☐ linpp105	1	ABCD	1-DEC-1989	1-DEC-1989 22:11	N	Y
☐ linpp64	1	ABCD	25-NOV-1989	25-NOV-1989 16:43	Y	Y
☐ nlplnt	1,2,3	SIMNON	21-NOV-1989	26-NOV-1989 18:26	Y	Y,Y,Y
☒ nlppfbs	1	SIMNON	26-NOV-1989	27-NOV-1989 19:02	N	Y

Context: linear tallinn nlppfbs 1

Figure 8. MEAD Model Browsing Form

Every level in the data-base Browser is encompassed by the Note Facility, which 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.

6. DATA-BASE MANAGER IMPROVEMENTS AND EXTENSIONS

As powerful and effective as the MEAD EDBMS may be, the user (and developer) inevitably wants more. The main deficiencies of the present version of the EDBMS 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 EDBMS 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.

6.1. Flexible Data Element Manipulation

Renaming and moving data elements are elementary functionalities that are easy to implement. (This may sound trivial - but users typically do 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 instance to another, Components cannot be moved to a Model where they are not used (in other words, a Component may only be moved from its "home" to another model that uses it via the Link mechanism mentioned above). Models (together with 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 *Compt_01* can be compared with *Compt_02* 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),
- *Result_01* can be compared with *Result_02* 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
- *Result_01* can be compared with *Result_02* 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 - this can be ascertained from the Condition_Spec).

6.2. 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. 8), 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 database tree in a graphical (scrollable) form that conforms to Fig. 5. 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.

6.3. Improved On-Line Documentation

The Note Facility can be made much more accessible if notes could be displayed or modified *from the current screen* rather than from the Browser (Fig. 8). For example, if a context-sensitive 'Note' button were always available, then the user could:

- 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 implemented 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.

6.4. Multi-User Data Base Access

Multi-user access to a single MEAD data base is the most important and substantial extension of the MEAD EDBMS. 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 EDBMS to multi-user access was presented in Taylor, Nieh and Mroz [4]. The layer *Sub-project* was proposed in addition to those shown in Fig. 5, 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 EDBMS uses for model version control) supports the following (DEC, [14]):

> storing elements in a library, fetching elements for modification in the user's workspace, controlling concurrent changes to the same element, merging concurrent changes to an element, creating successive versions of an element, comparing two versions of a library element, relating library elements into groups, defining classes corresponding to versions of a set of elements, tracking which users are working on various elements from a library, and maintaining a historical record of element and library transactions.

Other higher-level functions can be performed on CACE data elements to support multi-user access rigorously. For example, the DEC Module Management System (DEC VAX/MMS™ [22]) automates and simplifies building software systems based on source code, object libraries, include files, compilers, and compilation and link options. This would further discipline and rigorize the building of complicated models. The above examples are based on VAX VMS tools; suitable support software is also available under UNIX, e.g., the Configuration Management Facility (CMF™) provides similar functionality to CMS + MMS.

7. SUMMARY AND CONCLUSIONS

The introductory discussion attempted to demonstrate that rigorous engineering data-base management for computer-aided control engineering is both important and achievable. A hierarchical organization of CACE data base elements was presented, and mechanisms for maintaining data-base integrity were described. The importance of integrating the EDBM with the rest of the CACE environment was also mentioned, and the architecture and interface design that accomplish this were discussed. All of these concepts have been implemented and proven to be effective in the two MEAD environments.

Several areas of refinement and extension were outlined in the preceding section, to round out the concepts actually implemented in MEAD. It is hoped that these contributions will serve as the basis for more supportive CACE environments in the future.

Acknowledgements: The definition and development of the EDBM scheme described in this paper was sponsored in part by:

Flight Dynamics Laboratories
Air Force Wright Aeronautical Laboratories
Aeronautical Systems Division (AFSC)
United States Air Force
Wright-Patterson AFB, Ohio 45433-6553

In addition, the author would like to recognize that this article is based heavily on [4,7,8,16], and to thank his co-authors for their indirect contributions to this exposition.

REFERENCES

1. Edmunds, J. M., "Cambridge Linear Analysis and Design Program", *IFAC Symposium on Computer Aided Design of Control Systems*, Zurich, Switzerland, 1979.
2. Bunz, D. and Gutschow, K., "CATPAC - an Interactive Software Package for Control System Design, *Automatica, 21*, 209-213, 1985.
3. Taylor, J. H., "Conventional and Expert-Aided Data-Base Management for Computer-Aided Control Engineering", *Proc. American Control Conference,* Minneapolis, MN, June 1987.
4. 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, June 1988.
5. Mroz, P. A., McKeehen, P., and Taylor, J. H., "An Interface for Computer-Aided Control Engineering Based on an Engineering Data-Base Manager," *Proc. American*

™ VAX/MMS is a trademark of Digital Equipment Corp., Maynard, MA; CMF is a trademark of EXPERTWARE, Inc.

Control Conference, Atlanta, GA, June 1988.

6. Taylor, J. H., Frederick, D. K., Rimvall C. M., and Sutherland, H. A., "The GE MEAD Computer-Aided Control Engineering Environment", *Proc. IEEE CACSD'89,* Tampa, FL, December 1989.
7. Taylor, J. H., Rimvall C. M., and Sutherland, H. A., "MEAD II - a New CAD Environment for Controls Engineering", *Proc. American Control Conference,* Boston, MA, June 1991.
8. Taylor, J. H., Rimvall C. M., and Sutherland, H. A., "Future Developments in Modern Environments for CADCS", *Proc. 5th IFAC Symp. on CAD in Control Systems '91,* Swansea, Wales, July 1991.
9. Taylor, J. H., and McKeehen, P. D., "A Computer-Aided Control Engineering Environment for Multi-Disciplinary Expert-Aided Analysis and Design (MEAD)", *Proc. National Aerospace and Electronics Conference,* Dayton, OH, May 1989.
10. 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.
11. Taylor, J. H., "Expert-Aided Environments for CAE of Control Systems", Plenary Lecture, *Proc. 4th IFAC Symp. on CAD in Control Systems '88,* Beijing, PR China, August 1988.
12. 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.
13. Lohr, P. J., *CHIDE: a usable UIMS for the engineering environment.* Tech. Report, GE Corp. R & D, Schenectady, NY 12301, 1989.
14. *User's Introduction to VAX DEC/CMS,* DEC, Maynard MA, November 1984.
15. Barker, H. A., Chen, M., Grant, P. W., Jobling, C. P., and Townsend, P., "An Open Architecture for Computer-Assisted Control Engineering", *Proc. IEEE Symposium on CACSD,* Napa, CA, March 1992.
16. Taylor, J. H., Frederick, D. K., Rimvall C. M., and Sutherland, H. A., "Computer-Aided Control Engineering Environments: Architecture, User Interface, Data-Base Management, and Expert Aiding", invited tutorial for *Proc. 11th IFAC World Congress,* Tallinn, USSR, August 1990.
17. Taylor, J. H., "Environment and Methods for Robust Computer-Aided Control Systems Design for Nonlinear Plants", *Proc. Second IFAC Symp. CAD of Multivariable Technological Systems,* West Lafayette, Indiana, September 1982.
18. Taylor, J. H., "Computer-Aided Control Engineering Environment for Nonlinear Systems", *Proc. Third IFAC Symp. CAD in Control and Engineering Systems,* Lyngby, Denmark, August 1985.
19. Maciejowski, J. M., "Data Structures for Control System Design", *Proc. EUROCON '84,* Brighton, UK, 1984.
20. Rimvall, M., "A Structural Approach to CACSD", in *Computer-Aided Control Systems Engineering,* Jamshidi and Herget (Eds.), Elsevier Science Publishers, 1985.
21. Maciejowski, J. M., "Data Structures and Software Tools for CAD of Control Systems: A Survey", Plenary Lecture, *Proc. 4th IFAC Symp. on CAD in Control Systems '88,* Beijing, PR China, August 1988.
22. *User's Guide to VAX DEC/MMS,* Digital Equipment Corp., Maynard MA, 1990.