Appeared in *Proc. National Aerospace Electronics Conference (NAECON)*, Dayton Ohio, May 1989. © IEEE

A COMPUTER-AIDED CONTROL ENGINEERING ENVIRONMENT FOR MULTI-DISCIPLINARY EXPERT-AIDED ANALYSIS AND DESIGN (MEAD[†])

James H. Taylor
Control Systems Lab / KWD-209A
GE Corporate R & D
PO Box 8, Schenectady, NY 12301

Phillip D. McKeehen
Flight Dynamics Laboratory / FIGCA
Wright R & D Center
Wright-Patterson AFB, OH 45433-6523

ABSTRACT - The MEAD Project (MEAD = Multi-disciplinary Expert-aided Analysis and Design) involves integrating computer-aided control engineering (CACE) packages under a supervisor which coordinates the use of these packages with a data-base manager, an expert system, and an advanced user interface. In brief,

- The supervisor is the "shell" or package integrator for the underlying CACE packages and coordinates all activity within the MEAD environment.
- The data-base manager keeps track of system models that evolve over time (e.g., as better modeling information becomes available) and relates each analysis or design result to the right model instance.
- The expert system provides "expert aiding" for clear-cut but complicated procedures that would otherwise involve unnecessary low-level detail.
- The user interface facilitates access to the CACE package capabilities by users with widely different levels of familiarity with the environment, and simplifies the use of the expert system and data-base manager. These goals are achieved by permitting the user to work in several modalities:
  - — a menu/forms style UI for basic CACE activity,
  - — using MEAD commands when this mode expedites CACE work compared with the more user-friendly menu/forms mode,
  - — using the core packages' native commands when the exact desired functionality is not conveniently available via MEAD Commands, or
  - — using the MEAD Macro Facility (which includes both a macro-execute mode and a flexible macro-edit mode).

  The availability of a variety of interaction modes as listed above supports the inexperienced user as conveniently as possible, while providing the more experienced MEAD user with a flexible and effective environment for CACE.

The resulting CACE environment is currently developed for flight control engineering, although the ultimate aim is to support the integration of flight, propulsion, and structural control.

[†] This work was sponsored in part by the Flight Dynamics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), United States Air Force, Wright-Patterson AFB, Ohio 45433-6523, under contract F33615-85-C-3611.

1. INTRODUCTION

1.1 Motivation

Progress in flight vehicle technology requires better performance over larger flight envelopes. These increasingly stringent demands require the application of advanced control technology to keep pace. In particular, this translates into the current trends toward integrating flight, propulsion and structural control (so that beneficial coupling can be exploited and adverse coupling can be reduced or eliminated) and toward using recent advances in control theory to accommodate dynamic variability, uncertainty, component failures, and other effects and phenomena that may degrade performance or limit the flight envelope.

In turn, the demand for advanced integrated control necessitates improvements in CAD software, so better designs can be obtained at less cost in terms of time and effort. In response, the US Air Force initiated the MEAD Project (Multi-disciplinary Expert-aided Analysis and Design) to create a computer-aided control engineering (CACE) environment to facilitate the CAD of modern flight control systems.

1.2 Goal and Approach

The specific goal of the MEAD project is to develop both a "conceptual" and a "real" MEAD Computer Program (MCP) using an interactive data-base manager and an expert system to achieve a user-friendly tool for control system design and analysis. The approach to realizing the MEAD Project goal can be outlined simply by listing the MEAD tasks:

Task 1:	Define methodology for Integrated Flight, Propulsion, and Structural Control (IFPSC)
Task 2:	Determine MEAD software requirements, specifications, and architecture
Task 3:	Define and plan implementating an MCP for flight control systems analysis and design

Task 4:	Implement the MCP
Task 5:	Test, demonstrate, and deliver the MCP

The purpose of Task 1 was to determine IFPSC procedures and functions, the IFPSC models used, the input and output data for each step in IFPSC engineering, and the relations among these entities (steps and data elements). Task 2 dealt with requirements, specifications, and architecture for the MEAD Computer Program, based on the objectives of functionality, functional integration, multidisciplinary integration, flexibility, modularity, data-base management support, expert-aiding, and specific support for IFPSC. The output of this part of the effort was a conceptual definition of the "ideal" MCP. Task 3 more specifically defined an "implementable" MCP, in terms of currently available software and MEAD Project resources. Tasks 4 and 5 encompass implementation through delivery; we are presently at the culmination of Task 5.

1.3 Architecture

The MEAD Project approach to creating the MCP is to take maximum advantage of existing software modules. Implementing the MCP thus entails the integration of several CACE packages under a *Supervisor* which coordinates the execution of these packages with a data-base manager *(DBM)*, an *Expert System*, and an advanced *User Interface*. The resulting CACE environment architecture is depicted in Fig. 1. The underlying CACE tools ("core packages") include MATRIX$_X$^{TM} for linear analysis and design, ALLFIT and AUTOSPEC for flying qualities assessment, and GENESIS for nonlinear simulation and linearization. The last three packages were supplied by Northrop Corp. Aircraft Division (see Acknowledgements).

1.4 Outline

The MEAD Project and MCP development are discussed within the following framework:

Section 2:	overall MCP functionality,
Section 3:	MCP supervisor,
Section 4:	MCP data-base manager,
Section 5:	MCP expert system,
Section 6:	MCP user interface,
Section 7:	hardware and software requirements,
Section 8:	multidisciplinary integration, and
Section 9:	conclusions.

2. OVERALL MCP FUNCTIONALITY

MEAD Task 1 reviewed and systematized the CACE approaches and procedures used in the various disciplines of IFPSC. Control engineers from GE Aircraft Engine and Northrop Corp. Aircraft Division played a pivotal role in this effort (see Acknowledgements). The following list captures the *basic* IFPSC functions that must be performed by the MEAD Computer Program:

1. Modeling: nonlinear and linear airframes, nonlinear and linear engines, linear structural models, and nonlinear and linear controller(s), both discrete- and continuous-time; building arbitrary models from components
2. Simulation: initialize system state variables, set system parameters, define input signals, designate simulation variables for storage, run a simulation, and display (plot or list) simulation variables
3. Trimming (nonlinear airframe) or steady-state determination
4. Linearization of nonlinear models
5. Linear analysis: eigenvalues / eigenvectors, controllability and observability, model reduction, model transformations, root locus, and frequency response (Bode plots, Nyquist plots, gain and phase margins)
6. Linear control system design ("point designs"): frequency-domain methods for single-input / single-output designs (manually, by adding user-specified lead/lag or PID compensation); time-domain methods (pole placement, LQR, LQG, LQ output feedback)
7. Control system validation: frequency-domain analysis of linear models, and time-domain analysis (simulation) of linear and nonlinear models.
8. IFPSC-specific CACE functions: equivalent system fitting (e.g., fitting Bode-plot data for a high-order model of an airframe with stability augmentation system with equivalent low-order transfer functions), and MIL spec verification (checking such low-order equivalent system models for compliance with MIL-F-8785C)

Note that the above list is *not* all-inclusive. The objective of MCP-1.0 is to create a state-of-the-art environment that directly supports basic CACE for flight control systems. Furthermore, the nature of CACE is such that creating an exhaustive catalog of functionality would not be possible (or would at least be the subject of much debate), and new approaches and theories are being added on a continuing basis. The MCP environment has thus been designed to be "open", in the sense of being extensible either by adding built-in functionality or by the higher-level use of MEAD macros or the "Package Mode" access to modern linear analysis and design software. The MEAD Macro Facility and Package Mode are discussed further in Section 6.1.

3. MCP SUPERVISOR

The MCP Supervisor provides the "shell" or package integrator for the underlying CACE packages. Multiple packages can be run under the supervisor, and data formatting is converted, when necessary, to ensure compatibility between packages. The MEAD supervisor accepts "MEAD commands" and translates these into "package commands"; therefore, the user does not have to learn all of the intricacies involved in using each package unless advanced functionality is to be accessed via Package Mode (using a core package under the MCP using its own interface - see Section 6.1).

The use of MEAD Commands in the MCP Supervisor facilitates the unification of the MEAD user interface for dual functions such as nonlinear and linear simulation. The distinction is made by entering either L or NL with commands relating to that activity (e.g., defining system input signals and actually executing the simulation). The UI further simplifies this management of linear and nonlinear systems functionality by keeping the "context" of the user's present work (L or NL) in local memory and inserting L or NL in the user's commands as they are generated via the mouse-driven UI screens.

The supervisor contains most of the "intelligence" of the MCP. It uses its built-in knowledge about how to use the underlying packages, how to direct the activity of the DBM, and how to manage the command and data flow between itself and the user interface. It tracks the high-level activity of the user, including knowing what model(s) have been configured (the user may have one linear model active in the MATRIX$_x$^{TM} workspace and one nonlinear model in use represented by one version of GENESIS running as a subprocess). Finally, the supervisor controls the invocation and use of the expert system by activating it, telling it what rule base to load and execute, serving as the conduit for communications between the user and expert system, and handling the expert system's results when it is finished.

4. MCP DATA-BASE MANAGER (DBM)

Data-base management requirements for CACE were determined as part of Task 1. Data-base elements were catalogued and categorized, and the relations among them were established. In terms of data element categories, there are *models* which are comprised of *components* and a *description* (containing type, connection definition, etc.). Associated with each model there are *results* (e.g., files containing frequency response data or time-history data). Models and results are generally organized according to *Projects* (e.g., project = F18FCS for the analysis and design of a Flight Control System for the F18). These considerations led to the basic DB organization portrayed in Fig. 2; this hierarchy reflects the observation that control engineers naturally think of projects and models as being of paramount importance; all data elements produced during CACE activity are "children" of these entities.

A class of secondary data element is not shown in Fig. 2: the *condition specification*. This element contains information regarding operations performed on a model before a result is obtained; these include actions such as changing a parameter from its nominal value, specifying an initial condition and/or input signal before performing a simulation, defining a frequency list before obtaining Bode plot data, etc. The condition specification also records numerical conditions, such as setting a tolerance for determining controllability or observability, selecting an integration algorithm for simulation, etc. Capturing this data is critical, since it is the combination of model instance *and* condition specification that determines the result and thereby allows the engineer to document or repeat the result. Condition specs are stored in the MEAD data base and may be recovered for any result that has been saved.

While the CACE database categories are few in number and simple, there are several factors that complicate the DBM problem: Models tend to change over the life-time of the project, some results are also models (e.g., linearizations of nonlinear models or transformed linear models), and components tend to be used in several models yet they should be stored in one location to simplify their maintenance. The MCP DBM includes mechanisms to handle all of these situations with little or no burden on the user. This was in accord with the specific design goal of providing DBM support with minimal changes in the way the IFPSC engineer works and minimal added overhead. Further details regarding the MEAD DBM are provided in [1,2].

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

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

One remaining relation that complicates the hierarchical DB organization is that which associates a linearization as a *result* obtained using a nonlinear model with a linearization used as a model *component*. The same situation exists with regard to linear model transforms: For example, one may create the controllable and observable part of a linear model, and desire to save this as both a result and model for further study. These associations are tracked in the MCP DBM using a mechanism called the *reference*. The engineer may inspect a linearization result and check the reference to see if it exists as a component in any model; from the other perspective, a linear model component may be checked to determine if it was obtained as a result generated with a particular nonlinear model and trace that result back to determine how it was obtained (e.g., at what flight regime). The value of a linear model is greatly reduced if component traceability in this sense cannot be assured.

In summary, there is a straightforward hierarchical organization of Projects, Models, Components, and Results + Condition Specifications in the MEAD database. Models are tracked over time via class number and version control. In addition, there are relations called links and references to completely maintain the integrity of the database.

The DBM functionality outlined above is provided by the MCP at virtually no cost to the user. The supervisor and DBM perform all database organization and version control with no effort from the user beyond supplying personally meaningful names for new elements. In fact, accessing the database via the MCP 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, as discussed in Section 6. Finally, we have designed the MCP so that even "Package Mode" activity (using the core interactive packages directly) receives basic data-base management functionality; see also Section 6.

5. MCP EXPERT SYSTEM

The expert system (ES) provides "expert aiding" for a set of clear-cut but complicated procedures that would otherwise involve the user in unnecessary low-level detail. The concept of expert-aided CACE was originally defined in [3]; the primary difference in MEAD involves adopting a less ambitious model for expert aiding that makes the expert system the *user's assistant* [4] rather than putting it in charge of the CACE effort being performed. This change in perspective was motivated by the specific goal of providing support without getting in the IFPSC engineer's way; it is called the "control engineer's assistant" paradigm.

A second noteworthy feature of the integration of the expert system with the MCP is that the ES interfaces with the MCP supervisor in exactly the same fashion as the user working through the user interface. The ES outputs MEAD commands and/or package commands to the supervisor, and gets the same return as the UI, i.e., a result (if little data is involved), a file name (for larger results), or messages (errors or information). This simplifies "knowledge capture", since this is exactly the output and input of the supervisor when the expert user performs the task at hand.

A survey of tasks that might be expert-aided was conducted under Task 1. A number of CACE operations were identified as candidates for expert aiding, and then a pair of indices was established for each candidate according to the *value* of expert aiding (e.g., time savings) and the *feasibility*. The specific functions selected for implementation in MCP-1 were MIL_Spec high-order system Bode-data fitting (frequency-domain fitting according to MIL-F-8785C via ALLFIT), and flying quality assessment using AUTOSPEC combined with control system design iteration to bring the flight control system into compliance with specifications.

6. MCP USER INTERFACE

The MCP user interface (UI) is designed to facilitate access to the CACE package capabilities by users with widely different levels of familiarity with the environment, to unify access to the core packages despite very different package interfaces, and to simplify the use of the data-base management capabilities by taking advantage of the synergism among these functionalities (see [2], for example).

6.1 User Accessibility

The goal of achieving a UI accessible to users of widely different levels of expertise was achieved by permitting the user to work in a number of modes:

- *AIDE Mode* (AIDE = Aircraft Integrated Design Environment), from a menu/forms style UI for basic CACE functionality,
- *M_Command Mode,* i.e., using MEAD commands when this expedites CACE work compared with the more user-friendly menu/forms UI,
- *Package Mode,* i.e., using an interactive core packages' native commands when the exact desired functionality is not available via M_Commands, and use of the
- *MEAD Macro Facility,* which includes both a macro-execute mode and a flexible macro-edit mode and is based on M_Commands, Package Mode commands, or a combination of these.

The opportunity to use a variety of interaction modes as outlined above supports the inexperienced user as conveniently as possible (primarily via AIDE), while providing the more experienced MEAD user with a flexible, effective environment for CACE.

The AIDE Mode was designed to be the most "user friendly". An example of the use of the MCP for frequency-domain analysis of a linear model is illustrated in Fig. 3. The entire menu tree down to the desired functionality is visible, and the operations are defined by mouse operations (point and click) down to the bottom level where command parameters must be defined (e.g., ω_{min}, ω_{max}, and the number of points). The user does not need to know any commands or syntax, and the menu tree hierarchy is designed so that there is a natural path to the desired functionality (Linear Analysis $\rightarrow$ Freq Response $\rightarrow$ Bode Analysis).

The command modes were implemented to expedite the work of users who are familiar with the MCP and/or the underlying packages or to allow the user to perform activity not available in the AIDE Mode. M_Command Mode was designed primarily for use in macros (below); Package Mode was incorporated to provide access to any functionality supported in the interactive core packages (MATRIX$_X^{\text{TM}}$ and GENESIS in MCP-1).

Package Mode is currently available only for MATRIX$_X^{\text{TM}}$. The reason for this is that MATRIX$_X^{\text{TM}}$ is itself more open than GENESIS (i.e., it can support the generation of arbitrary results), and its interface is better suited for DBM support. One can obtain basic data-base management functionality in Package Mode under the following general conditions: The user must configure a linear model within the AIDE Mode before beginning to work within the environment provided by MATRIX$_X^{\text{TM}}$, and the user must designate those elements that are to be managed (saved in the data base as results or as models). The way this works is as follows:

- The user invokes the MCP, selects a Project, and configures a linear model.
- The user may perform any MCP operations on this model, and save results in the DB automatically.
- Whenever the user desires to carry out an operation that is not readily available in the MCP, the 'Package' button on the Resource Bar is clicked to open a Package_Command window (see Fig. 4).
- The user may execute any sequence of MATRIX$_X^{\text{TM}}$ commands and thereby create any data objects (arrays) in the MATRIX$_X^{\text{TM}}$ workspace.
- DBM functionality is obtained as follows:
  - — A data object may be saved in the data base *for the configured model* by entering a simple M_Command that is intercepted, interpreted and executed by the supervisor. For example, if the result is comprised of the arrays Q and K, then the user may enter *MEADSAVE(Q, K, r=thisdata)* and the result will be stored in the DB with the user-supplied name *thisdata*.
  - — A data object may be saved in the Project DB as a *new model* via a second M_Command. For example, if the new system is represented in state-space form by *Anew*, *Bnew*, *Cnew*, *Dnew*, then the model is inserted in the Project DB by first issuing the MATRIX$_X^{\text{TM}}$ command *Snew* = [*Anew* , *Bnew* ; *Cnew* , *Dnew*] and then the M_command *MEADMDL* (*ABCD* , *Snew* , *nS* , *nI* , *nO* , *m* = *thatmdl*). The significance of this notation is that *ABCD* is the type of model, *Snew* is the model in standard packed form as defined above, *nS, nI, nO,* represent the numbers of states, inputs, and outputs, respectively, and *thatmdl* will be the name assigned to the model in the database.

(Note from the above examples that the user has to be an experienced user of the core package in order to use this facility effectively.)

Finally, the Macro Facility provides the "power user" with the capability to streamline CACE by using custom macro procedures. Macros may be set up to initialize the MCP (e.g., to select a project of current interest and configure a key model), to perform a procedure defined by a sequence of M_Commands, to execute a task that may require the use of Package Mode, or to carry out a combination of these activities. Table 1 provides simple illustrations of MCP macros for start-up, for nonlinear simulation, and for evaluating the singular value decomposition of a linear model. Note that macros may be invoked directly, or they may be loaded into the editor, modified for the task at hand, and then executed.

Table 1. MCP Macro Facility Illustrations

MCP Initialization Macro:

```
* initialize in project and configure models:
*   (YF16 is the project name)
projectid YF16
*   '0' → use highest class of model LinFCS
config YF16 LinFCS 0
*   Bring up second class of model YF16mdl
config YF16 YF16mdl 2
* ... now start interactive use of these models
```

MCP Simulation Macro:

```
* config YF16MDL; trim, define input, simulate
*   JH Taylor 16 Feb 1989
config Feb11 yf16mdl 0
*   invoke DCL to place desired trimdef.dat file:
copymytrim_17.datmead work:mytrimdef.dat
trim mytrimdef.dat r=nomtrim
*   ramp: ampl = 15, start at T=1, rise-time=5
input nl PDAF ramp 15.0 1.0 5.0 1
simu nl 15. 0.015 r=pdafsim15
```

MCP Singular Value Decomposition Macro:

```
* config a linear model before using this macro
*   use package mode to get sing. value decomp.
setmode pkge matrixx
[a, b, c, d] = split(S,nS)
[u, s, v ] = svd(a)        // get the SVD
meadsave(u,s,v,r=svdecomp)
// (Result_name = svdecomp in DBM)
setmode aide
* 'setmode aide' -> return to AIDE mode
```

6.2 Data-Base Access

The original MEAD Project goal was to design the UI to provide access to the DB management functionality with minimal user overhead. In fact, we found that it was possible to design the UI so that the DBM is an asset with respect to overhead, rather than a liability. This is due in part to the natural hierarchical data-base system organization, and in part to the use of "object-oriented design" features in the UI.

The first pivotal decision was that a query language would be excessively difficult to implement and use for DB access. This led us to display data element information much as one displays file information in any computer environment via a "Directory" command. For our purposes, we created hierarchical "Browsing Screens" to list the user's projects, the models within each project, and the elements below each model (components and results, see Fig. 2). A sample screen for Result Browsing is depicted in Fig. 5.

The second realization that streamlined the UI in relation to the DBM was that we could use the browsing screens for functionality in addition to display. Thus one may browse the models in a given project and immediately designate a model for use (analysis and design); this is called "configuring a model", and may be done by hitting the CONFIG button shown in Fig. 6. One may also create new models, edit models, purge models, and add, modify, or delete model notes from the same screen.

7. HARDWARE AND SOFTWARE REQUIREMENTS

The official hardware platform for the MCP is the VAX computer under the VMS operating system, with user interface via a Tektronix 4107 terminal (or higher model number) or an IBM PC or PC clone running a Tektronix 4107 (or higher) emulator. This platform is adequate for the type of UI and functionality needed for the MCP, although we recognize the UI could be more powerful and "fancy" (e.g., faster and more flexible in layout) if we had a true workstation environment at our disposal with high-resolution graphics and window management.

MCP software needs were primarily driven by functionality, the hardware platform, and the availability of existing software modules suitable for the tasks at hand. Software incorporated in the MCP may be categorized as core packages and support software.

We performed a survey of core CACE software packages in Task 2, including the following generic CACE packages: MATRIX$_X$™, PRO-MATLAB™, and Ctrl-C™ for linear analysis and design, and EASY-5, ACSL, and SIMNON for nonlinear simulation. Several IFPSC-specific packages were also reviewed, such as IFPCSIM (recently revised to create GENESIS, a nonlinear simulator for nonlinear flight-and-engine control systems), ALLFIT (for equivalent system fitting, i.e., fitting Bode-plot data for an airframe with stability augmentation system with equivalent low-order transfer functions), AUTOSPEC (for MIL spec verification, i.e., checking low-order equivalent system models from ALLFIT for compliance with MIL-F-8785C), COMET (for control mode analysis of aircraft engines), and MFAP (for structural modeling). For a variety of reasons, including functionality, the technical merits of each packages, cost, availability, and site-specific considerations, the core packages presently in the MCP are MATRIX$_X$™, GENESIS, ALLFIT, and AUTOSPEC, as portrayed in Fig. 1.

Support software selected for use in the MCP include: Delphi© (proprietary GE software) for the ES shell, the Computer / Human Interface Development Environment (CHIDE; proprietary GE software) for the UI, the Relational Object System for Engineering (ROSE) [5] for the DBM, and the DEC Code Management System (CMS) for version control.

The MCP Supervisor was created from scratch. The first decision was the selection of a programming language, given the problem of developing a shell for existing software for CACE under the constraints that:

- the analysis and design codes (core packages) are in FORTRAN,
- the expert system shell is in Lisp, and
- the DBM and UI are in C and ROSETALK [5].

The requirement was to achieve a reputable product that is forward-looking, maintainable, supportable, etc. - i.e., to meet standard software engineering goals. The solution: use a "professional" programming environment that encourages (enforces) and supports highly-disciplined software engineering practices.

The above considerations led us to use Ada™ in coding the MCP Supervisor. With Ada, we obtained the following language characteristics: strong typing, in-code specification, structured programming, information hiding, data abstraction, and process abstraction. The results of these characteristics are that compilation catches many more errors than is true with less disciplined languages (so more time can be spent in design and less in implementation and testing), the use of "hacks" in code is discouraged, and the integrity of data objects is protected. These results are especially beneficial in developing an embedded system such as the MEAD Supervisor.

8. MULTIDISCIPLINARY INTEGRATION

Issues pertinent to multidisciplinary integration are manifested in three areas:

1. Integration of models:
   a. nonlinear-to-linear, finite-element-to-linear
   b. airframe + engine + structures + controls

   The MCP accomplishes model integration of both types, by design. The main issue is tracking relationships, as is done by the MEAD DBM.

™ Ada is a registered trademark of the U. S. Government, Ada Joint Program Office.

2. Integration of tools:
   a. nonlinear simulation with linear analysis and design (e.g. MATRIX$_x$^{TM} with GENESIS), finite element modeling with linear analysis and design (e.g. MATRIX$_x$^{TM} with MFAP)
   b. flight-specific + engine-specific + structures-specific tools

   The MCP Supervisor achieves this goal by design. The integration of nonlinear simulation with linear analysis and design is implemented, and the integration of finite element modeling has been anticipated.
3. Integration of the data base:
   a. models maintained with integrity (via links)
   b. maintaining model "lineage" (via references)

   The MCP DBM achieves this by design.

9. CONCLUSIONS

Phase I of the MEAD Project is concluded, as defined by the tasks listed in Section 1.2. The MCP has been in test and evaluation at GE Corporate R & D and the USAF Flight Dynamics Laboratory (FDL) starting in May 1988 (versions "MCP-0.1" to "MCP-0.6"), it has been demonstrated at FDL in June and October 1988, and final debugging, refinement, and robustification is being performed now for the official delivery of MCP-1.0 in March 1989.

MCP-1 represents a new, more supportive environment for computer-aided control engineering (CACE). The most important novel features are an *integrated engineering data-base manager*, a *built-in expert system*, and a *flexible user-friendly user interface* including a "point-and-click' interactive mode, two command modes (MEAD and Package), and a Macro Facility. Another notable attribute is the ability to use the core packages directly without sacrificing the benefits of data-base management. In terms of CACE functionality, MCP-1 is a basic CACE package for flight control system analysis and design, and much functionality is still not particularly "fancy". The higher-level functionality of MATRIX$_x$^{TM} is available through the most user-friendly access mode; all lower-level primitives may be used via Package Mode. A number of extensions and refinements are planned, including the incorporation of a more flexible general-purpose nonlinear simulator, improved UI features, more user-friendly handling of linear models, and additional expert aiding.

The MCP-1 CACE environment is currently most fully developed for flight control engineering, although the ultimate aim is to support the integration of flight, propulsion, and structural control. The MEAD software has been designed to meet this objective with the addition of suitable functionality (new core packages and associated user-interface extensions). The development of the MEAD Computer Program brings to fruition many of the concepts described in [6].

Acknowledgements: A major part of the work described above was performed for the Wright Research and Development Center, United States Air Force, Wright-Patterson AFB, Ohio 45433-6523, under contract F33615-85-C-3611 by GE Corporate Research and Development (GE-CRD) as prime contractor, with primary subcontract support from Northrop Corporation Aircraft Division (NCAD), GE Aircraft Engine (GE-AE), and Aule-Tek Inc. The contributions of the following individuals are most gratefully recognized: *NCAD:* Dinesh Joshi, Pierre Wong, Peter Shaw, Juri Kalviste; *GE AE:* John Polley, Shrider Adibhatla; *Aule-Tek:* David Kassover, James Trojan, Michael Charbonneau, Albert Antoniotti; and *GE-CRD:* Hunt Sutherland and Magnus Rimvall. In addition, Prof. Dean K. Frederick of Renssellaer Polytechnic Institute provided invaluable consulting services over the course of theis project. Finally, numerous personnel at the USAF Flight Dynamics Laboratory have greatly influenced the course of this effort: 1st Lts Mark Schiller and Mike Dunbar, and Mssrs. David Bowser, Stan Lash, Tom Gentry, and Tom Hummel.

REFERENCES

[1] 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.

[2] 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 Control Conference,* Atlanta, GA, June 1988.

[3] Taylor, J. H. and Frederick, D. K., "An Expert System Architecture for Computer-Aided Control Engineering", *IEEE Proceedings, Vol. 72,* 1795-1805, December 1984.

[4] J. H. Taylor, "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.

[5] Hardwick, M, *User Manual for ROSE,* Report no. 86-24, Rensselaer Polytechnic Institute, Troy, New York, 1986. See also Hardwick, M, and Sinha, G, "A Data Management system for Graphical Objects," *Proc. IEEE/Computer Society International Conference on Data Engineering,* Feb. 1986.

[6] Taylor, J. H., "An Expert System for Integrated Aircraft/Engine Controls Design", *Proceedings of the National Aerospace and Electronics Conference (NAECON),* Dayton, OH, May 1985.

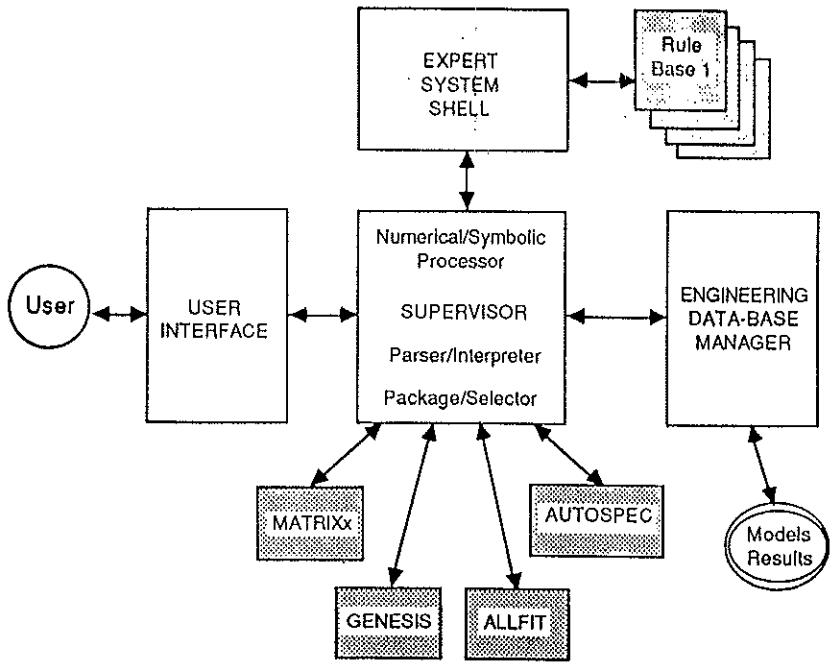


Figure 1.

MCP Architecture

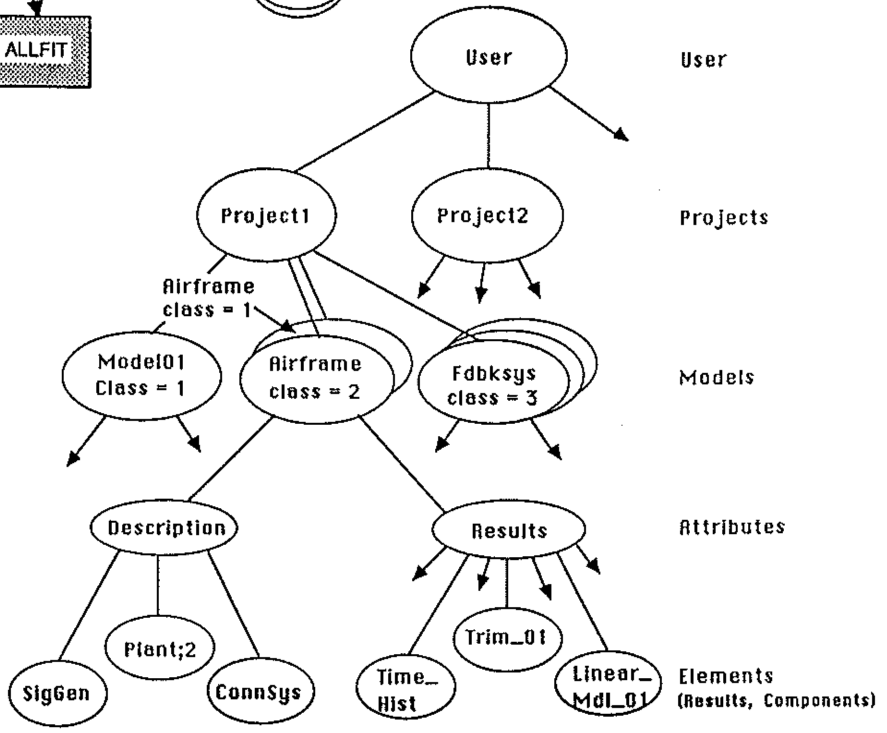
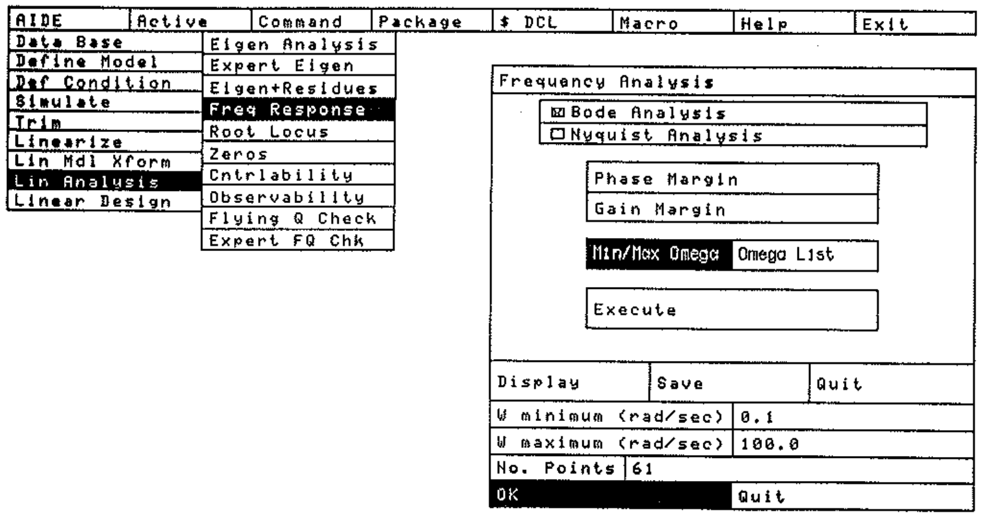


Figure 2.

MCP Data-Base Hierarchy

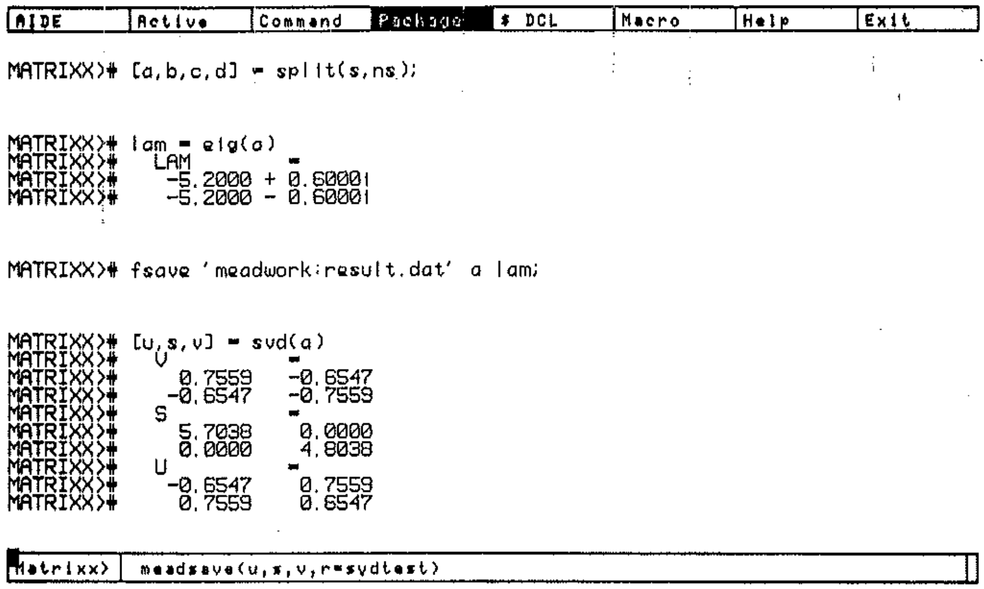


[✓ Bode Analysis] ☐ Nyquist Analysis

W minimum (rad/sec)	0.1
W maximum (rad/sec)	100.0
No. Points	61

Figure 3.

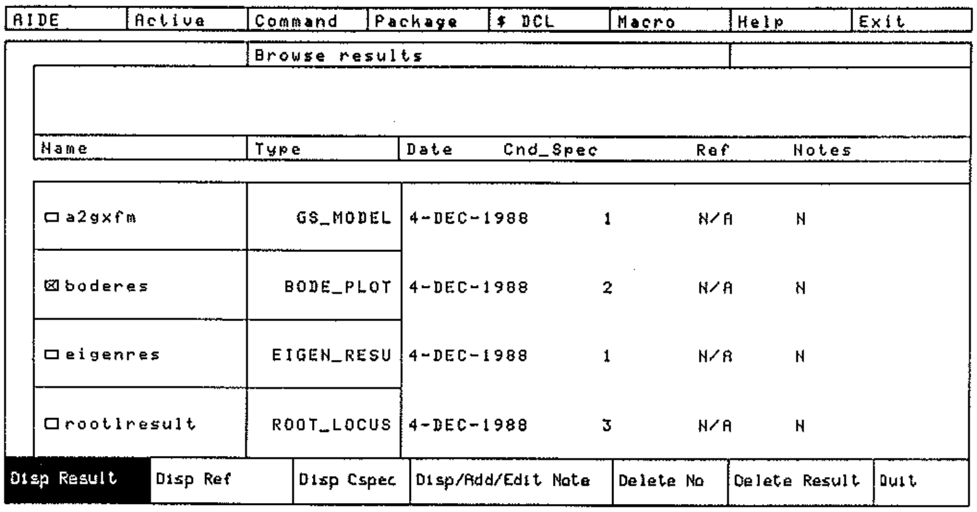
MCP AIDE Screen for Frequency-Domain Analysis



```
MATRIXX># [a,b,c,d] = split(s,ns);


MATRIXX># lam = eig(a)
MATRIXX>#   LAM           =
MATRIXX>#     -5.2000 + 0.6000i
MATRIXX>#     -5.2000 - 0.6000i


MATRIXX># fsave 'meadwork:result.dat'  a lam;


MATRIXX># [u,s,v] = svd(a)
MATRIXX>#   V             =
MATRIXX>#      0.7559   -0.6547
MATRIXX>#     -0.6547   -0.7559
MATRIXX>#   S             =
MATRIXX>#      5.7038    0.0000
MATRIXX>#      0.0000    4.8038
MATRIXX>#   U             =
MATRIXX>#     -0.6547    0.7559
MATRIXX>#      0.7559    0.6547

Matrixx>  meadsave(u,s,v,r=svdtest)
```

Figure 4.

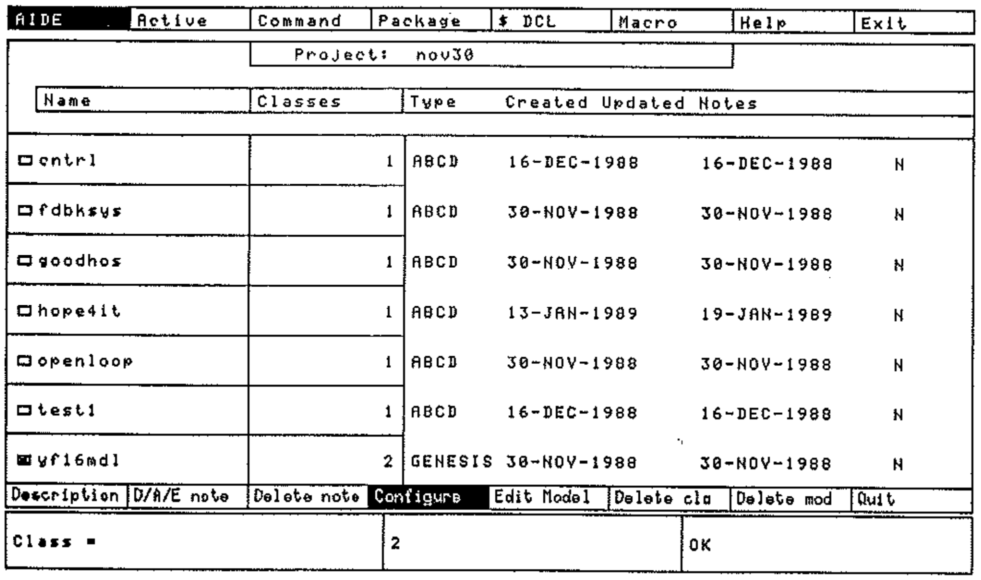
MCP Screen for Package Mode



Name	Type	Date	Cnd_Spec	Ref	Notes
☐ a2gxfm	GS_MODEL	4-DEC-1988	1	N/A	N
☒ boderes	BODE_PLOT	4-DEC-1988	2	N/A	N
☐ eigenres	EIGEN_RESU	4-DEC-1988	1	N/A	N
☐ rootlresult	ROOT_LOCUS	4-DEC-1988	3	N/A	N

Figure 5.

MCP Screen for Result Browsing



Project: nov30

Name	Classes	Type	Created	Updated	Notes
☐ cntrl	1	ABCD	16-DEC-1988	16-DEC-1988	N
☐ fdbksys	1	ABCD	30-NOV-1988	30-NOV-1988	N
☐ goodhos	1	ABCD	30-NOV-1988	30-NOV-1988	N
☐ hope4it	1	ABCD	13-JAN-1989	19-JAN-1989	N
☐ openloop	1	ABCD	30-NOV-1988	30-NOV-1988	N
☐ test1	1	ABCD	16-DEC-1988	16-DEC-1988	N
■ yf16mdl	2	GENESIS	30-NOV-1988	30-NOV-1988	N

Class = 2

Figure 6.

MCP Screen for Model Browsing