Data-Driven Supervisor Design for CACE Package Integration

Magnus Rimvall, James H. Taylor
Control Systems Laboratory
General Electric Corporate R&D
P.O. Box 8, Schenectady, NY 12301, USA

Abstract: Large industrial control problems often require the use of several different control packages, each of which has a different user interface and incompatible data formats. The GE-MEAD Computer Aided Control Engineering (CACE) program attempts to alleviate these problems by providing a shell for integrating different commercial CACE packages together with a graphical front end, a data-base manager, and an expert system. The GE-MEAD package as a whole will be presented, with emphasis on the unifying philosophy behind the MEAD data-driven Supervisor. Main features of this Supervisor include a full command language with procedural/macro capabilities, control units for different core packages, and a data-driven translator from generic MEAD commands to specific commands for these core packages.

Keywords: Computer-Aided Control Systems Design, Man-Machine Interfaces, Command Languages, Databases, Formal Language Translation

Introduction

The control engineer of today will find that most of his problems can be solved accurately and efficiently by the use of robust and commercially maintained Computer-Aided Control Engineering (CACE) packages. However, it is quite commonplace in an industrial setting that several CACE packages need to be used, as no single package has the capability of solving all aspects of more complex problems. Unfortunately, as these different CACE packages all have different user interfaces and as few packages give any explicit support to project data-base management and data compatibility, the engineer is burdened with operational and administrative duties which decreases his productivity. Our recent effort in software development for CACE has attempted to alleviate these problems through the integration of standard CACE packages into an environment called GE-MEAD[1] that includes an advanced user interface[2], a supervisor which coordinates the execution of CACE tasks among the included packages, a data-base manager[3], and an expert system[4]. The resulting software architecture is depicted in Figure 1. The focus of this presentation is the overall design and implementation of the package integrator (the "Supervisor"), as seen both from a control engineer's or "user's" perspective and the package designer's or "developer's" perspective.

Recent developments in CACE has been dominated by improvements to the man-machine interface aimed at "enhancing the user friendliness" and "deliver more broadly supportive CAD tools". These were two of the main motivations for developing GE-MEAD. The areas we chose for improvement were the user interface and support facilities for data-base management and expert aiding. In this paper, we first given an overview of the MEAD Environment and its architecture, before concentrating on the Data-Driven Supervisor that is the "heart" of MEAD

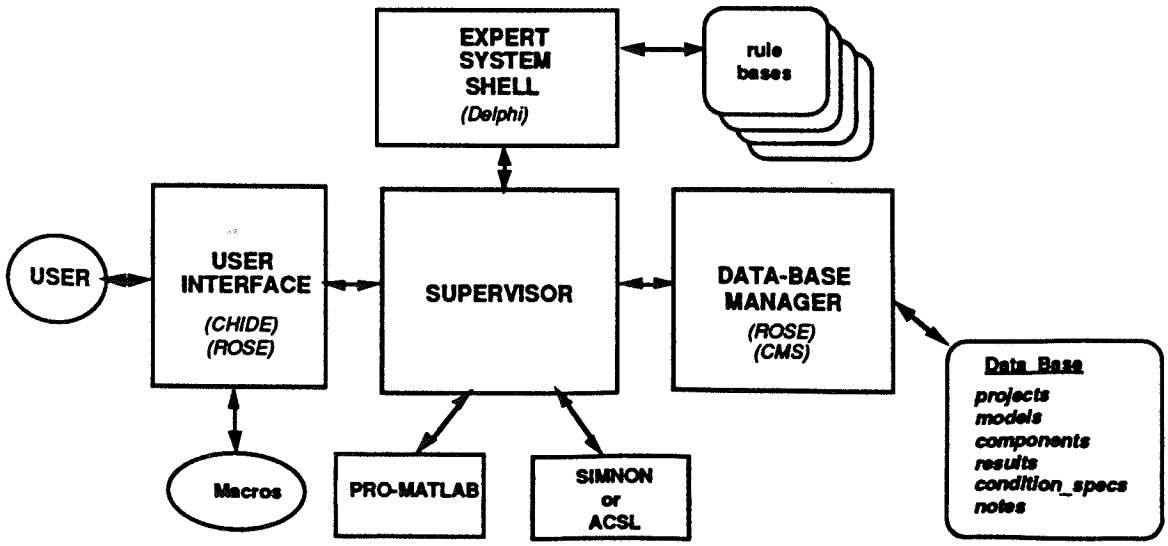

Figure 1. The GE-MEAD architecture. The User-Interface, the Supervisor, the Expert System and the core packages Pro-Matlab and Simnon/ACSL all run in separate processes. The Data-Base is part of the Supervisor process.

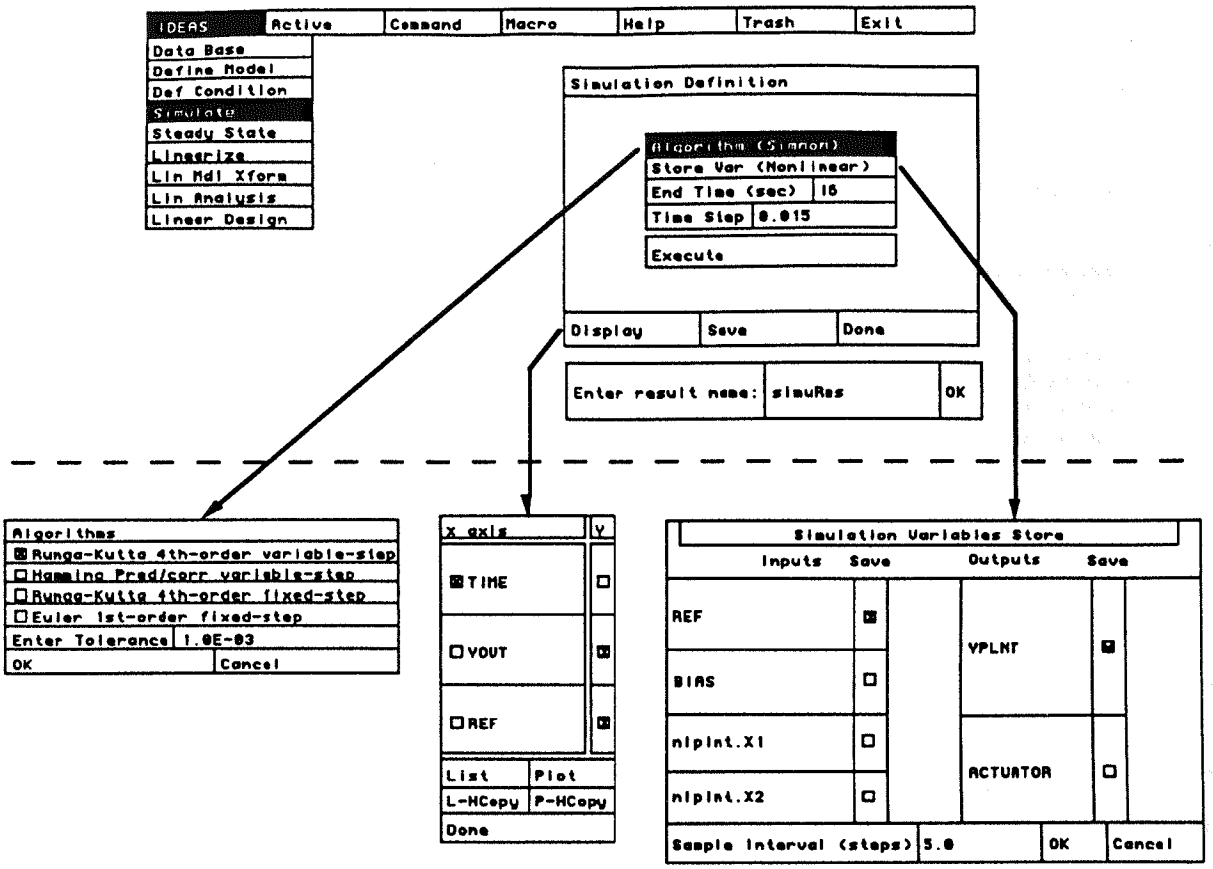

Figure 2. The MEAD simulation screen and affiliated forms. The buttons/entry-fields of the simulation form are positioned in the order they are to be used. Thus, the user would first select integration algorithm, variables to be stored away and the simulation parameters and then execute the simulation. Finally, the user may display or save simulation results. Each of the forms on the bottom is brought up by clicking on the appropriate button.

The GE-MEAD† Components

Powerful CACE packages exist that jointly provide the overall functionality needed in an all-encompassing CACE environment such as MEAD. The use of such packages allows for the decoupling of numerical functionality from the environmental issues of support and user friendliness. The MEAD system can therefore be viewed as a "shell" for existing software rather than yet another package developed from the foundations of numerical analysis and control algorithms. The incorporated packages ("core packages") include the Pro-Matlab™ package for linear analysis and design (GE-MEAD; the Air-Force version of MEAD uses the MATRIX$_x$™ package), and the ACSL™ or Simnon™ nonlinear simulator. Standard, "off-the-tape" versions of these core packages are used to diminish interdependencies between the packages and the supervisor.

Access to the numerical software is given via the User Interface (UI) illustrated in Figure 2. The graphics-based UI gives the user convenient and predictable access to the full functionality of MEAD. The system has been carefully designed to constitute a single, uniform interface to the very different core packages.

The Supervisor, which is situated at the center of the architecture, contains most of the "intelligence" of MEAD. It not only controls the core packages, it also directs the activity of the DBM, and manages the command and data flow between itself, the user interface and the expert system. It tracks the high-level activity of the user, including knowing what model(s) have been activated. Data is converted to ensure compatibility among the packages.

The supervisor is the only module of GE-MEAD which is completely built from scratch. All other modules are partially based on pre-existing software[1]. The supervisor and the front-end of the DBM are coded in the Ada™ language. MEAD runs under VAX/VMS™ with plans to port it to Unix™ Workstations.

† The origin of the acronym MEAD (Multi-disciplinary Expert-Aided Analysis & Design) is the US Air Force MEAD Project[5], 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. Development of the GE-MEAD Supervisor was internally funded at GE.

™ Pro-Matlab is a trademark of The Mathworks; Matrixx is a registered trademark of Integrated Systems, Inc; ACSL is a registered trademark of Mitchell and Gauthier Associates; Simnon is a trademark of SSPA. Ada is a registered trademark of the U.S. Government, Ada Joint Program Office; VAX and VMS are registered trademarks of Digital Equipment Corp.; Unix is a trademark of AT&T Bell Laboratories.

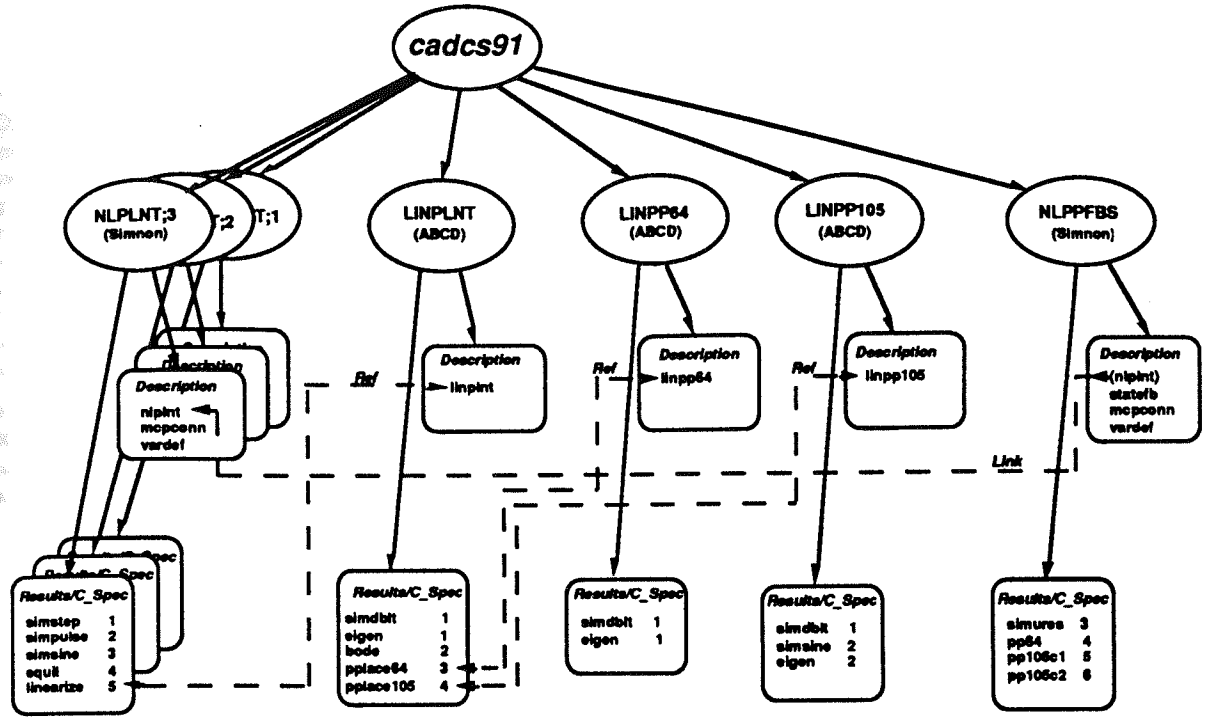

Figure 3. The MEAD data-base architecture. The models `NLPLNT`, `LINPLNT` and so on are grouped into projects such as `cadcs91`. Each model may consist of several interconnected components. `NLPPFBS` consists of the unique component `statefb` and the shared component `nlplnt`, these two components are connected and configured according to the information in `ncpconn` and `vardef`. Results derived from a model are stored under that model.

The GE-MEAD User Interface

The task of producing a single program serving a large, heterogeneous community is non-trivial. In designing the MEAD User Interface (UI)[2], 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 as illustrated in igure 2. 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.

Action forms (such as the simulation form in Fig. 2) vary in size and content, but they all have the same general layout. The top setup buttons enable actions that the user may want or need to perform before executing the function; the `Execute` button triggers the actual operation (e.g., simulate). The bottom row of buttons allows the user to view and save results , and to quit. Whenever the user must enter any alphanumeric information, such as the name of a result, an input form will open up.

The GE-MEAD Data Base

Though largely independent of the underlying CACE packages, the architecture of the MEAD UI is closely coupled with the structure of the MEAD Data Base Manager (DBM). It is used to store away models, and results associated with these models, in a structured, safe and maintainable manner[3]. Various "Browsing Forms" are used to present the user with the available choices, together with any pertinent information about the selectable items. The information within these forms change dynamically as objects are created or deleted. Items presented to the user for selection correspond to the data-base hierarchy, which in turn mimics the way control engineers naturally organize models and results, as shown in Figure 3. 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. Associated with each model there are *results* (e.g. frequency or time response data).

While the CACE database categories are few in number and simple, there are a few more dimensions to the problem: Models tend to change over the life-time of the project. Thus, the DBM must be able to keep track of a series of updated models , so that each analysis or design result can be associated with the correct model instance. Moreover, individual components (e.g. an actuator model) may be used in several models, but for maintenance reasons only one copy of that component is to be kept in the data-base. Finally, results (e.g. linearizations) may be viewed as models themselves, so they must be stored also as models so that they can be used for further analysis and design operations.

The GE-MEAD Supervisor

The MEAD Supervisor plays the role of the hub in MEAD's spoke-and-hub like architecture (see Fig. 1). In this position it serves as execution coordinator and package integrator. The various core packages run as separate processes under the direct control of the supervisor, and the supervisor is responsible for combining and controlling these packages as well as reformatting or converting data, when necessary, to ensure compatibility. The supervisor is also responsible for facilitating communication with the UI/user, the data-base and the expert system.

As mentioned in the previously section, the MEAD User Interface provides a consistent and unified interface to the different core packages and to the data-base. This relieves the user from having to learn the intricacies involved in using each package. However, the core packages are used in their "native" form, and thus each package has a command language of its own, a non-standard error protocol, and unique data formats and file-name conventions. To minimize the complexity of the UI (its finite-state machinery nonetheless has 900+ states) the UI itself is essentially oblivious to the fact that different core packages may be invoked for different commands. The UI understands *only* the MEAD command language, it is the responsibility of the supervisor to translate these commands into the appropriate "package commands".

User modes

The MEAD command language is not only the standard means of communication between the graphical UI and the Supervisor, it is also made available to the user in different ways. This allows the user to alternate the regular point-and-click operations with different kinds of command entries, each suiting different needs and level of user experience. From a users perspective, four operating modes are available:

- *IDEAS Mode* (IDEAS = Integrated Design Environment for All Systems): the menu and forms based, "point-and-click" style graphical interface presented in the previous section. The UI will translate each user operation into one or several MEAD commands, which are then interpreted by the supervisor and either directly executed or passed on to the correct core package.

  IDEAS gives quick and convenient access to basic CACE functionality. The well structured menus and forms, together with context-dependent help, makes this mode suitable for beginning users. Moreover, as each individual action button and form typically 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 core 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.

  As illustration of the conversion from graphical point-and-click operations to the MEAD command language consider Fig 2. A click on the Execute button would result in the command

  ```
  Simulate(16.0,0.25)
  ```

  being sent from the UI to the supervisor. A subsequent click on the Save button and entry of the result name would cause the following command to be transmitted to the supervisor.:

  ```
  Save_Result(simuRes)
  ```

- *MEAD Command Mode:* this mode allows the user to directly enter MEAD commands. It is primarily intended for the experienced user wishing to utilize the full command language of the supervisor (which includes command flow statements such as if-then-else and loops, as well as some 75 control and data-base related commands). The ability to combine and structure commands freely can expedite tasks for the expert user. Thus, the two "UI-commands" above may be combined into one:

  ```
  Simulate(16.0,0.25,result=>simuRes)
  ```

- *MEAD Macro Mode.* This mode allows the user to define and save sequences of commands for repeated execution. Macros may contain MEAD commands, package commands, or a combination thereof. Commands can be captured in script form during normal operation of MEAD, or they may be entered/edited in a regular text editor. Macros may consist of linear sequences of commands or they may be organized into general procedures with parameters which may be given different values each time the procedure is executed. The following example shows a procedure which will make a parameter sweep using ten simulations. The sweep limits, the result name and end-time may be interactively changed (the "&" character means "use the value of", and the `for` loop uses the range expression `from:increment:to.`):

  ```
  procedure Run_Range
            (Par_Lo,
             Steps    : Integer;
             Res_Name : String := "SimuRes";
             End_Time : Scalar := 16.0) is
  begin
   delta := (&Par_Hi-&Par_Lo) / (&Steps - 1);
   for ThisPar in &Par_Lo:&delta:&Par_Hi loop
     Para(controller.k,&ThisPar);
     Simulate(&End_Time,0.25);
     Append_Result(&Res_Name);
   end loop;
  end Run_Range;
  ```

  Macros are dynamically associated with menu fields for convenient execution from Ideas mode. They may also be called from command mode.:

  ```
  Run_Range(0.0,10.0,11,Case2);
  ```

  where the default end-time of 16.0 is used. Alternatively, if the user had selected the macro `Run_Range` from the dynamic macro menu, he would have been presented with a form for entering/changing the different parameter values.

- *Package mode:* allows the user to directly enter commands to a core package using the syntax and command set of that core package. This mode requires the user to know how to directly operate the underlying package; it is intended to be used when the exact desired functionality is not available

through the MEAD commands. Because the commands pass through the supervisor, full data-base management capabilities are still available [3].

Of these four modes, the first three utilize the MEAD command language. Thus, the user can choose between four modes, but the Supervisor needs to distinguish only between MEAD and Package commands. Package mode is basically a transparency mode where the Supervisor passes the commands over to the package without checking or interpretation (the Supervisor *does* look for two special commands signaling that data is to be stored away in the data base, and for the command terminating package mode).

From an implementational standpoint, the reuse of the MEAD command language in three out of the four main user modes has advantages beyond the obvious simplification of the Supervisor's complexity:

- The complexity of the UI can be minimized by relieving it of any knowledge of the particular syntax and command limitation of the individual packages. This advantage should not be underestimated, as the inherit complexity of the graphical UI is considerable.
- The User Interface is made insensitive to upgrades and changes in the core packages, and the introduction/exchange of a core package requires only minor changes to the UI.

The GE-MEAD command language

The MEAD Supervisor is accessed over a direct-execution, interpreted command language. Entered commands are translated to an internal "threaded code" and thereafter immediately interpreted -- the use of threaded code instead of pure interpretation shortens the execution time for all repetitive code segments. The command language follows a syntax similar to that of Ada. Regular control flow statements such as assignments, `if-then-else` and `loop-end loop` constructs are available. Procedures/functions may be interactively defined, they are thereafter available for immediate execution. The regular "programming-language" statements have been augmented with control-oriented data structures such as matrices and system representations using a syntax similar to that found in the Pro-Matlab and IMPACT[6] packages.

The close coupling between the MEAD Supervisor and the data-base provides the Supervisor with the means for taking an active role in defining the context of each command. Thus, the commands entered in command mode can be kept short by relying on the supervisors ability to insert default values (for project, model, and result names; for model, component and result type information; for tolerances and scaling, etc).

The data-driven support of core packages

The Supervisor is responsible for translating the generic MEAD commands to specific commands to be sent to the core packages, data-base and/or operating system, as well as for interpreting the result and status flags echoed back. Typically this involves translating single MEAD commands into a set of more primitive package/data-base commands. Rather than having the individual commands such as `BODE` or `C_to_D` (Continuous_to_Discrete transformation) implemented within the supervisor as Ada procedures, this functionality is again coded in the MEAD command language using more primitive MEAD commands. This approach is similar to the "toolbox" concept used in Pro-Matlab. However, the MEAD supervisor adds a further dimension to this approach by not only equivalencing MEAD commands with toolbox entries, but also transforming primitive MEAD-commands into the correct syntax for the particular packages.

The MEAD supervisor is "data-driven" in the sense that all MEAD commands are transformed into correct package commands by executing macros, i.e. ASCII data files, rather than having such detailed knowledge of the core packages coded in Ada. This allows for an easy and quick extension of the supervisor to support new commands, as no recompilations or links are necessary. Moreover, as the normal user may formulate his own macros, the adaptability of the supervisor to the requirements of individual users is virtually unlimited.

We will illustrate this with the implementation of the command `C_to_D` (note that the code driving $MATRIX_X$ has been included for sake of illustration; GE-MEAD does not presently support this package although it was supported under the version of MEAD sponsored by the US Air-Force[4]):

```
Procedure C_To_D(New_DT    : in Scalar;              -- no default
                 Result    : in Result := null;      -- def: no result saved
                 New_Model: in Model  := null;       -- def: no model created
                 Model     : in Model  := null) is -- def: context model
begin
  XX_Set_Context(&Model,Linear);                    -- >1<(see comments
  case XX_Return_Context_Package is                 -- >2< below)
    when "PRO_MATLAB" =>
      case XX_Return_Context_LinForm is              -- >3<
        when "ABCD" =>
          %Pro_Matlab                                 -- >4<
            (aa,bb,c,d)  := unpack_ss(ss_model);       -- >5<
            new_dt       := &New_DT;
            (a,b)        := c2d(aa,bb,new_dt);
            new_ss_model := pack_ss(a,b,c,d);          -- >5<
          %end;                                       -- >6<
          XX_Modelize_Result(&Model,&New_Model,DABCD,new_ss_model,new_dt);
          XX_Save_Result(&Model,DABCD_Model,&Result,(new_dt,new_ss_model));
        when "DABCD" =>
          XX_Propagate_Error("MEAD_U_ILLEGAL_CONT_ONLY",&Model);
```

```
        when others =>
          XX_Propagate_Error("MEAD_U_ILLEGAL_LINEAR_FORM",&Model);
      end case; -- model class
    when "MATRIX_X" =>
      case XX_Return_Context_LinForm is
        when "ABCD" =>
          %Matrix_x                                    -- >4<
            newdt := &New_DT;
            newss := discretize(ss,ns,newdt);
          %end;                                        -- >6<
          XX_Modelize_Result(&Model,&New_Model,DABCD,(newss,ns),newdt);
          ... more of the same
      end case; -- model class
    when others =>
      XX_Propagate_Error("MEAD_U_LINEAR_PACKAGE_REQUIRED",&Model);
  end case; -- context package
end C_To_D;
```

Comments:

>1 The supervisor always designates the last-used model as the "context"; this is used for all subsequent commands unless another model is specified.

>2 This case/branch determines which core package is to be used.

>3 The `C_to_D` command is available only for continuous models, this branch makes sure an error message is transmitted otherwise.

>4 The statements within `%Pro_Matlab .. %end` and `%Matrix_x ... %end` have these packages as their destination. However, these lines are expressed in MEAD syntax which is then translated by the Supervisor into commands following the Pro-Matlab and MATRIX$_x$ syntax, respectively. For example, assuming that the macro `C_to_D` is called with `New_DT=0.1`, the Pro-Matlab section is translated into the Pro-Matlab statements:

```
[aa,bb,c,d] = unpack_ss(ss_model);
new_dt = 0.1;
[a,b] = c2d(aa,bb,new_dt)
new_ss_model = pack_ss(a,b,c,d)
```

Note how the syntax has changed to

>5 The state-space model representation of ISICLE[7] is used throughout MEAD/Pro-Matlab to express and store away linear models. unpack_ss and pack_ss are ISICLE commands to convert model representations to standard Pro-Matlab matrices.

>6 If the parameter `Result` is given a value, the result of the operation will be stored away in the data-base. Moreover, as the result from this conversion may be interpreted as a model itself, the result will be stored away as a configurable model if the parameter `New_Model` is given a value. This new model may later be activated as the default model for analysis or design.

The workhorses of the package interfaces in the Supervisor are the command translators which accept commands in MEAD syntax as input and produces package commands acceptable to the core packages. These translators have made it possible to implement 90% of the analysis and design commands in a completely data-driven fashion (the data-base related administrative commands are Ada/Rose coded, due to the constraints of Rose). Moreover, new packages can be introduced into MEAD after only a relatively modest effort: a new command translator has to be written (or adapted from an existing one), and code interpreting the error and status messages of the new package has to be inserted. Thereafter, the data-driven files have to be updated for the commands supported by the new package. Compared to rewriting large Ada-packages describing every analysis and design command (which was the case in the Air-Force version of MEAD), this saves a factor 3-5 in implementation time, and simplifies both debugging and maintenance as the total lines-of-code (Ada+Data-driven files) is relatively low.

Conclusions

There has been a growing realization over the last years that existing CACE packages may be reaching a good state in terms of functionality and numerical power, but that the lack of User Interface and data standardization decrease the efficiency at which they can be employed for large industrial problems. In this paper we have shown how the unifying UI and package coordinator integrator can make the of GE-MEAD quality commercial CACE packages more attractive in large and complex projects.

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., Sutherland, H., Taylor, J.H. "GE-MEAD's UI, 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.

[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] 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.

[5] Taylor, J.H., "Expert Aided Environments for CAE of control systems". Proc. 4th IFAC Symposium on CAD in Control Systems, Beijung, PR China, 1988.

[6] Rimvall, M., Schmid, F., and Cellier, F.E., "The different modeling capabilities of IMPACT". In Proceeding IEEE Control System Society 3rd Symposium on Computer-Aided Control System Design (CACSD), Arlington, VA, September 1986.

[7] Minto, K.D., Chow, J.H., and Beseler, J., "Lateral Axis Autopilot Design for large Transport Aircraft: An Explicit Model-Matching Approach", In Proceddings of the ACC, 1989