American Control Conference, Minneapolis, MN, 10 June 1987, © AACC.

CONVENTIONAL AND EXPERT-AIDED DATA-BASE MANAGEMENT FOR COMPUTER-AIDED CONTROL ENGINEERING

James H. Taylor
General Electric Corporate Research and Development
Control technology Branch
Schenectady, New York 12345

ABSTRACT

There has been substantial recent progress in the development of software for computer-aided control engineering. This includes both "conventional" software as well as environments based on the use of expert systems. One major area has not received much attention until very recently: data-base management. In this paper, we define a framework for data-base management, describe a set of design assumptions and corresponding functional requirements, and describe several roles that an expert system can play in this activity.

1. INTRODUCTION

The past decade has witnessed a major effort in the development of software for computer-aided control engineering (CACE). This involves both "conventional" software such as CTRL-C and MATRIX$_X$ [1, 2] (to merely name two well known and widely marketed packages) and, more recently, environments based on the use of expert systems (e.g., CACE-III [3,4] and CASCADE [5]). A comprehensive overview of CACE software may be found in the Extended List of Control Software (ELCS, Rimvall [6]). There is one major area, however, that has not received much attention until recently: data-base management (DBM).

As CACE environments become more powerful, the need for keeping track of the models, analysis results, control system designs, and validation study results becomes more pressing. In a real industrial project, the number of files generated in the complete design cycle and the relations among these files can be very difficult to comprehend and manage without support. In addition, an unmanaged data base can become more of a liability than an asset after a short period of time.

In this paper, an expert-aided or expert-system based approach to DBM is developed. Certain concepts underlying this approach are founded, in part, on the recent work of Maciejowski [7] and Rimvall [8], who have contributed to delineating a systematic description of data structures and organizations for CACE. This DBM definition is also based on our own requirements analysis and experience. Once the basic requirements are outlined, we consider the following elements in detail: maintaining the **integrity** of the data base and implementing high-level DBM activities such as **reconstituting** an invalidated DB, **archiving** the DB, **guiding** the conduct of a session or project, **documenting** the outcome of such activity, etc.

The framework for the higher-level DBM functionality is a rule-based expert system. We believe that there is a sufficient amount of heuristic decision making and/or decision support required for a high-capability DBM system to justify this implementation.

2. DATA-BASE MANAGEMENT FOR CACE

2.1 The Problem:

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. We assume that the user starts with nonlinear models of the process to be controlled, and progresses through the following range of CACE activity: nonlinear simulation (e.g., model validation and behavioral analysis), equilibrium determination, linearization, linearized system analysis and design, nonlinear control system design, and control system validation [9,10]. 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 + ACSL, MATRIX$_X$ + SYSTEM_BUILD, the GE Federated System [11] are major examples - but none of these manage the resulting DB beyond implementing rudimentary file-naming conventions.

In many applications, such activity develops a substantial DB. In flight control, for example, a typical DB 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 (eigenvalues, frequency responses, root-locus data, singular values, . . .).

A portion of such a DB is shown in an informal network diagram in Fig. 1. The primary elements of project PROJ_07 provided by the user are shown in the second row, namely, operating point definition data (U0_1, U0_2, . . .), a nonlinear plant model (PLANT), control system specifications (C_S_SPECS), and the definition of a control system configuration (here, a connection specification C_S_CONN and a signal generator or "driver" C_S_DRV). Derived entities include equilibria

(EQ_1, EQ_2), linearized models (L_S_1, L_S_2), analysis results (e.g., frequency response data, L_S_1_FR), compensators based on linearized models and control system specifications (L_S_1_C, L_S_2_C), control systems based on compensator models (C_S_1, C_S_2), time-histories (e.g., step response data; T_H_1_01, . . .), etc. Note that some entities have multiple parentage (e.g., each linearized system model is based on the nonlinear model PLANT and a specific operating point (in this case, equilibria EQ_n), and some elements have multiple offspring (e.g., PLANT); clearly, the DB cannot be represented as a tree structure. Based on this example, it is also clear that multi-disciplinary applications such as integrated flight and propulsion control (IFPC) can produce an even larger and more complicated DB. For a typical IFPC project the DB size and complexity is compounded by the necessity of combining 20 flight regimes with 9 engine operating conditions, for example.

As the above cases make clear, CACE project activity can generate hundreds of results and files, of which the user may wish to retain and manage a substantial percentage. This may not be a "large" DB in terms of commercial DBM systems, but it is beyond the capacity of most human users to manage effectively without support.

2.2 CACE DBM Functions

The primary areas of DBM that provide helpful support for CACE may be informally summarized as follows:

a. *Relation Management* - e.g., given a file name, the DBM should be able to determine what type of data element it is, what model was used in its generation, what conditions were imposed, etc.; or conversely, the DBM should be able to find a data element that was generated using model <model_name> under conditions <condition_spec>, if it exists.

b. *Integrity Checking* - e.g., if the user edits a model file, the DBM should tentatively invalidate all data elements created with that model as a 'parent'; the user should be warned and have the final determination as to the actual validity of the data. (Reformatting the model, adding comment lines, or other cosmetic changes would not actually invalidate all 'offspring'; only the user can make this determination in most cases.)

c. *Reconstituting a DB* - e.g., if the integrity of all or part of a DB has been compromised, the user should be able to command that the DBM system eliminate the invalid elements and regenerate them using the new 'parent' data element.

d. *Archiving a Project DB* - e.g., the DBM should be able to delete all unnecessary or redundant data elements, organize the information in relational form, and store it on tape or other off-line medium for future reinstallation and use.

e. *Annotating a Project DB* - e.g., the DBM system should allow the user to add notes to the information in the relations to enhance the future understanding of the items in the DB.

f. *Documenting a Project* - e.g., the DBM system should be capable of producing a summary report of the transactions that took place in the course of a project to provide at least the raw material for project documentation.

2.3 CACE DBM Design Assumptions

The requirements of control engineers for managing an effort of the sort envisaged in Section 2.1 are somewhat different from those of a commercial application such as personnel record keeping. One difference is that the data is quite disparate, rather than fitting into one or a few straightforward data elements that can be represented by a simple schema or template. Another is that some information is well-suited to maintenance in relations (e.g., the relation between a nonlinear model and its linearized offspring), while other data is better kept in files (e.g., time-history data). Finally, the engineer generally wishes to have relatively free access to the data (e.g., to be able to edit a file containing a model), while typical commercial DBM systems closely control or deny such access.

Before DBM requirements can be specified, it is necessary to establish certain ground rules or design assumptions. The following is proposed as a model for DBM for a project involving several engineers working on a problem:

1. Each project has a central primary model DB:
   - managed by the DBM system under one person's (the Project Manager's) control and
   - accessed by other users having read_only privileges;
   - changes in a primary model cause integrity flags to be set in all users' DBs (project and sub-project DBs) for any data elements based on that model.
2. Each user employs the DBM system to create and manage his/her own sub-project DB.
3. Integrity checking is based on time stamps or file version number (not file contents).
4. Primary storage of engineering data is carried out in a file system:
   - it is *recommended* that each sub-project be organized via subdirectory tree (VAX/VMS) or other standard organization; this would not be forced;
   - most engineering data is **not** in the DB - only relations and file_names are managed;
   - the user can modify file contents:

— using the same file_name → integrity flags are set for all 'children' of that entity;

— using a different file_name → new entity (no children).

These conventions are quite typical of medium-sized multi-disciplinary projects such as those encountered in aerospace control applications, for example. Note that there is a strong trade-off between discipline and 'safety' on one hand *versus* user freedom and risk. The above design assumptions attempt to make reasonable compromises in this regard; clearly, there is no absolute answer to this problem.

2.4 Organization:

A CACE DB such as that informally sketched in section 2.1 needs to be organized in some fashion. We propose the use of a hierarchical framework, which, at the higher levels, is described by the following set of data entities (relations):

- PROJECT - tracks and relates the primary model(s), any CACE activity performed by the Project Manager, and the relation of each Subproject DB to the overall project (see the first template example, Table 1).
- SUBPROJECT - tracks and relates any CACE activity performed by a Project Engineer ('user').
- NONLINEAR_SYSTEM_MODEL* - defines the configuration of a system in terms of the subsystem interconnections and provides DBM access to selected system variables and parameters (see the second template example, Table 2).
- NONLINEAR_SUBSYSTEM_MODEL - provides DBM access to subsystem states, variables, and parameters (see the third template example, Table 3). A subsystem can be continuous- or discrete-time, and may represent a plant, controller, sensor model, or any arbitrary dynamic system or component.
- NONLINEAR_SYSTEM_PROPERTIES - contains or points to results such as equilibrium values, time-histories, points of discontinuity, describing-function based frequency response [10], bifurcation points, etc.
- NONLINEAR_SUBSYSTEM_PROPERTIES - contains or points to results such as those enumerated above pertaining to a nonlinear subsystem.

* We regard *all* system models as being 'configurations', comprised of blocks or subsystems. For example, the system PLANT_0 might be made up of an actual plant model (set of nonlinear ordinary differential equations) and a signal generator; FDBK_SYS might be comprised of a plant model, precompensator, sensor model, and signal generator, assembled in the standard feedback configuration; etc. Allowing nesting (configurations of configurations) is not a necessity (cf. SIMNON [12]); the desirability of this capability depends on both the discipline and the complexity of the system structures supported by the particular selection of analysis and simulation software in the CACE environment.

- LINEARIZED_SYSTEM_MODEL - similar to nonlinear_system_model, except it is permitted to possess properties denied in the nonlinear case (e.g., eigenvalues).
- LINEARIZED_SUBSYSTEM_MODEL - same as the nonlinear case except has an assumed {A, B, C, D} or transfer function form.
- LINEARIZED_SYSTEM_PROPERTIES - contains or points to results such as eigenvalues (poles), zeroes, controllability and observability grammians, frequency response, singular values, root loci, etc. for a particular linear system.
- LINEARIZED_SUBSYSTEM_PROPERTIES - contains or points to results such as those enumerated above pertaining to a linear subsystem.

Each entity in this hierarchy must be characterized by a template or definition in generic terms of the relation content. Tables 1 to 3 portray examples based on Fig. 1 (these are still tentative, pending implementation of the DBM system; also, note that they are influenced by our CACE software package selection [11]). Observe that any element in the model that is not listed in the relation (e.g., a hypothetical parameter 'K79' in relation 'PLANT', Table 3) cannot be tracked by the DBM system. This means, for example, that it would not be possible to distinguish between sets of results for different values of K79, because the required information is not available.

3. CACE DBM FEATURES

The capabilities of the proposed CACE DBM system can be divided into conventional and high-level. The former are those features that are commonly standard in DBM systems; the latter are functions that are specifically related to CACE in their meaning and implementation. Of course, the dividing line is not black-and-white: "conventional" features can be elevated to "high-level" simply by defining the capabilities more ambitiously.

3.1 Conventional Features

1. Integrity checking: the DBM should check file version numbers or time stamps at every data access; the data integrity flag of an element and all of its offspring should be toggled if this information is changed from the previous data access. For example, if a nonlinear model is modified, then the system should check the validity of secondary models derived from it, simulation and analysis results, etc.
2. Query: a simple query language is required to allow user to access information in the relations. Simple examples are:
   - "What is the origin of T_H_0174?" → model used, input definition, initial conditions, parameter values, . . .

- "What time history corresponds to { ALTITUDE = 14k + MACH = 0.7 } ?" → T_H_0091

3. Diary: a complete record of the user's (sub-)project should be maintained; the user can annotate it and eliminate undesirable or unnecessary data elements at will.
4. Recovery: a session journal file should allow an interrupted session to be resumed from the point of interruption with as little repetition as possible.
5. Reconstitution: e.g., if a primary model is modified, an entire sub-project DB can be flushed and re-built.
6. Archive: a sub-project can be put on tape (diary, relations, engineering data) for future reinstallation and use.

These features are well known in the DBM field and thus do not require elaboration here. These more mundane aspects of DBM (e.g., model, relation, and file management) do not seem to require expert aiding - the relations among the various data elements that are generated during a control system analysis and design exercise are not so complicated as to justify or require the addition of a rule-based system.

3.2 High-Level Features

Other functions of a DBM system for CACE are not so readily available or achieved. Based on our requirements analysis, we propose to implement these capabilities as part of an **Expert-aided data-base management (EADBM)** system. As in our previous efforts in expert-aided CACE [3,4], we propose to implement these functions in individual rule bases that are managed by a top-level supervisory rule base. We have tentatively identified the potential usefulness of rule bases to perform DBM activities that can be summarized as follows:

1. *Procedure-oriented user guidance and automation:* certain generic tasks in CACE involve distinctive patterns of activity that reflect a particular procedure being carried out. Whether or not the user is attempting to carry out such a task can be determined by monitoring the series of steps being executed and using approximate pattern-matching to seek correlations with pre-stored templates for such generic activities. This can be mechanized as a direct analog to the "command spy" and "scripts" approach proposed and developed by Åström, Larsson, and Persson [13,14]; here the required information is contained in the Project relation (cf. Table 1), thus eliminating the need for the command spy. Once it has been established that the user is pursuing such an activity, the EADBM system can support the user in several ways:

   - prompt the user if there is a subsequent major deviation from the procedure,
   - automate the rest of the procedure if desired,
   - automate the annotation and documentation of the procedure (below), etc.

   For example, in the case shown in Fig. 1 and Table 1, if the user has carried out the design, configuration, and validation of the control system NL_C_S_1 using a procedure that closely matched a template for such activity, then the EADBM system could annotate the project relation to this effect and ask the user if it is desired to perform the same task using the linearized systems l_s_2 and l_s_3 that are present in the Project DB. The main new element here is the integration of this feature with DBM-specific activity such as Project relation management, annotation, and documentation (see below).

2. *Procedure-oriented data-base annotation:* it is likely that a generic task as outlined above will have a specific DB annotation requirement that parallels the pattern of activity associated with the procedure being carried out. Once it has been determined that the user is involved in such activity, the EADBM rule base can then organize the results in a higher-level framework that reflects the task in-the-large instead of on a piecemeal basis (step-by-step). Examples of this include *sensitivity studies* (where a cycle of functions is carried out repetitively with minor perturbations such as parameter changes), *design validation studies* (where a prescribed set of simulations or analyses need to be performed), *design trade-off analysis*, and any other analysis or design cycle wherein higher-level relations are formed by organizing and/or comparing the results.

3. Custom design-procedure execution and documentation: many engineers and organizations have well-defined systematic schema for carrying out design exercises. These are usually governed by heuristic decision-making procedures that involve carrying out a task, scrutinizing the result, perhaps comparing the result with earlier work, and deciding what to do next. Such an approach can be captured by a rule-based system which, in the context of DBM, includes "expert knowledge" about how results of tasks are to be organized and reported in an engineering design document. This rule-based EADBM system can be developed to work from an Engineering Design Document template, and insert the specific information created as a design is being generated; the document would be ready for final editing at the end of the session.

The preceding ideas are still "conceptual designs", not working rule bases, based on a detailed requirements analysis and on identifying areas where the amount of heuristic logic and decision-making is substantial. The examples presented above include the following such activities:

- approximate template matching,

- applying "rules of thumb" to carry out high-level procedures such as trade-off study, design validation, etc.,
- automating a major task by extrapolating from one exercise (e.g., from one control design and validation, as mentioned above) to a set of exercises,
- organizing and annotating a DB based on an activity pattern, and
- executing and documenting a high-level task defined by a template.

This type of functionality calls for the use of an expert system implementation approach such as the rule-based system programming method; conventional programming approaches would be difficult to apply, maintain, and extend.

4. SUMMARY AND CONCLUSION

The need for a capable data-base management system as an adjunct to modern computer-aided control engineering software is established in Section 1. The definition of such a system in terms of the design assumptions and functional requirements is set forth in Section 2. The fact that control engineering has unique needs for high-level functionality that are not readily available in standard commercial DBM software and that are not amenable to a straightforward solution using conventional programming methods has motivated the use of an ancillary expert system to achieve the desired results. The primary reason for the use of this approach is the extent to which DBM for control engineering can be effectively supported by heuristic reasoning and decision aiding.

These factors have caused us to reject the approach of simply coupling a commercial DBM system with our CACE environment and accepting the resulting limitations and deficiencies. A small conventional DBM system integrated with an expert system for high-level functionality appears to provide the best solution with respect to capability, price, and performance.

The status of the work described in this paper is that the high-level DBM requirements definition is complete, and we are beginning to implement the conventional DBM aspects. Implementation is being done under contract F33615-85-C-3611 for the Air Force Wright Aeronautical Laboratories (Aeronautical Systems Division, Flight Dynamics Laboratories, Wright-Patterson AFB, Ohio); this contract effort will be reported separately. The development of expert-aided features as outlined above will be pursued after the conventional system has been completed and proven.

REFERENCES

[1] Little, J. N., Emami-Naeini, A., and Bangert, S. N., "CTRL-C and Matrix Environments for the Computer-Aided Design of Control Systems", *Proc. 6th International Conf. on Analysis and Optimization of Systems* (published in *Lecture Notes in Control and Information Sciences, Vol. 63,* Springer-Verlag), Nice, France, 1984.

[2] Walker, R. A., Gregory, C. Z. Jr., and Shah, S., "MATRIX$_X$: A Data Analysis, System Identification, Control Design and Simulation Package", IEEE *Control Systems Magazine, Vol. 2,* pp. 30-37, December, 1982.

[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] James, J. R., D. K. Frederick, and J. H. Taylor, "On the Application of Expert Systems Programming Techniques to the Design of Lead/lag Precompensators", *Proc. Control '85,* Cambridge, UK, July 1985; to appear in *Proc. IEE* (UK).

[5] Birdwell, J. D., J. R. B. Cockett, R. Heller, R. W. Rochelle, A. J. Laub, M. Athans, L. Hatfield, "Expert systems techniques in a computer-based analysis and design environment", *Proc. Third IFAC Symposium on CAD in Control and Engineering Systems,* Lyngby, Denmark, August, 1985.

[6] Rimvall, M., "The Extended List of Control Software (ELCS)"; contact M. Rimvall, Dept. of Automatic Control, Swiss Federal Institute of Technology (ETH), CH-8092 Zurich, Switzerland or D. K. Frederick, ECSE Dept., RPI, Troy NY 12180.

[7] Maciejowski, J. M., "Data Structures for Control System Design", *Proc. EUROCON '84,* Brighton, UK, 1984.

[8] Rimvall, M., "A Structural Approach to CACSD", in *Computer-Aided Control Systems Engineering,* M. Jamshidi and C. J. Herget, Editors, Elsevier Science Publishers, 1985.

[9] 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.

[10] 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.

[11] Spang, H. A., III, "The Federated Computer-Aided Control Design System", *IEEE Proceedings, Vol. 72,* 1724-1731, December 1984.

[12] Elmqvist, H., "SIMNON - An Interactive Simulation Program for Nonlinear Systems", *Proc. Simulation 77,* Montreux, 1977.

[13] Larsson, J. E. and K. J. Åström, "An Expert System Interface for IDPAC", *Proc. 2nd IEEE CSS*

Symp. on CACSD, Santa Barbara, CA, March 1985.

[14] Larsson, J. E. and P. Persson, "Knowledge Representation by Scripts in an Expert System", *Proc. American Control Conf.,* Seattle, WA, June 1986.

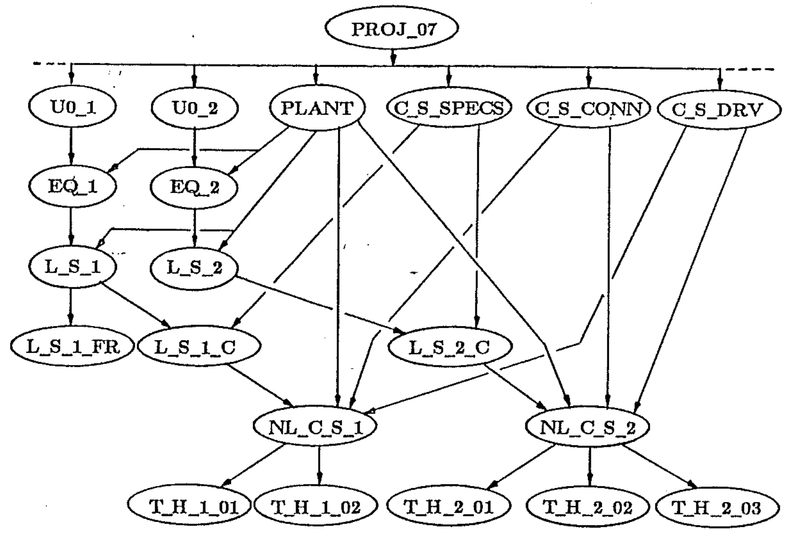

Figure 1. Typical CACE Data Base (Informal Schema)

Table 3. Subsystem Relation

RELATION 'PLANT'

name	type	parent	default
airframe1.t	file	user	-
u(1)	input_1	plant	-
u(2)	input_2	plant	-
u(3)	input_3	plant	-
y(1)	output_1	plant	-
y(2)	output_2	plant	-
y(3)	output_3	plant	-
x(1)	state_1	plant	-
x(2)	state_2	plant	-
...	...	...	-
x(9)	state_9	plant	-
x0(1)	state_1_init_val	plant	-
x0(2)	state_2_init_val	plant	-
...	...	...	-
x0(9)	state_9_init_val	plant	-
dx(1)	x_deriv_1	plant	-
dx(2)	x_deriv_2	plant	-
...	...	...	-
dx(9)	x_deriv_9	plant	-
K01	param	plant	1.0
K12	param	plant	15.0
...	...	...	-
K78	param	plant	81.0

Table 1. Project Relation

RELATION 'PROJ_07'

name	type	ident	id_type	parent	date	status
plant_0	nl_sys_mdl	(name)	relation	user	02-18-87	ok
plant_0_pr	nl_sys_prop	(name)	relation	plant	02-18-87	ok
l_s_1	l_subsys_mdl	lin1a.dat	file	plant+ eq_1	02-18-87	ok
l_s_2	l_subsys_mdl	lin2a.dat	file	plant+ eq_2	02-19-87	ok
l_s_3	l_subsys_mdl	lin3a.dat	file	plant+ eq_3	02-19-87	ok
l_s_1_pr	l_subsys_prop	(name)	relation	l_s_1	02-18-87	ok
l_s_2_pr	l_subsys_prop	(name)	relation	l_s_2	02-19-87	ok
l_s_3_pr	l_subsys_prop	(name)	relation	l_s_3	02-19-87	ok
c_s_spec	l_sys_prop	(name)	relation	user	02-23-87	ok
l_s_1_c	l_subsys_mdl	lc1a.dat	file	l_s_1+ c_spec	02-23-87	ok
l_c_s_1	l_sys_mdl	(name)	relation	user+ l_s_1_c	02-24-87	ok
nl_c_s_1	nl_sys_mdl	(name)	relation	user+ l_s_1_c	02-24-87	ok

Table 2. System Relation (Configuration)

RELATION 'NL_C_S_1'

name	type	parent	connect_from	connect_to	no_of_sigs
refin	input_sig	c_s_drv	refin[csdriver]	u1[esummer]	3
sysout	fdbk_sig	plant	y[plant]	-u2[esummer]	3
compin	err_sig	esummer	y[esummer]	u[compen]	3
compout	comp_out_sig	l_s_1_c	y[compen]	u1[dsummer]	3
distin	dist_sig	c_s_drv	distin[csdriver]	u2[dsummer]	3
plantin	input_sig	plant	y[dsummer]	u[plant]	3
rollrate	output_sig	plant	y(1)[plant]	-	1
yawrate	output_sig	plant	y(2)[plant]	-	1
sideslip	output_sig	plant	y(3)[plant]	-	1