Distributed modelling and data base management in simulation

François E. Cellier
Dept. of Electrical & Computer Engr.
University of Arizona
Tucson, AZ 85721

Magnus Rimvall
Institute of Automatic Control
Swiss Federal Institute of Technology
ETH-Zentrum
CH-8092 Zürich - Switzerland

ABSTRACT

This paper describes a concept of structuring simulation systems in such a way, that the individual program modules remain manageably small. This concept shows some considerable advantages over currently applied programming techniques in that it provides the user of the software with increased flexibility, in that it allows several persons to interact with different portions of the software system in a basically independent manner, and in that it opens new perspectives with respect to multiprocessor implementations.

INTRODUCTION

In a recent paper (1), we listed 141 different desirable features of simulation software systems *vs* 15 different software systems currently on the market or under development. The most comprehensive of these systems, COSY (2,3), is a huge system with roughly 300 keywords, and yet a good number of the 141 features are not implemented even in COSY.

Worse, those 141 features are strictly features of a simulation system leaving the model building process entirely up to the user. Oren (4) recently listed a large number of features, he would like to see implemented in a modelling system. Compared to a system providing a good number of those features, COSY is still just a dwarf.

What are we going to do about this? Are we going to code up a gigantesque program of some several 100000 lines of code to implement a supercompiler for a supersoftware system? Who is going to maintain such a software thereafter? Who is going to further enhance it by implementing yet additional features (which are most certain to be requested!)? Obviously, this is not a conceivable solution to the problem.

We must find ways to decompose this complex problem into a series of simpler tasks which are basically independent of each other, each of which must be kept sufficiently small to make it implementable, maintainable, and updatable. These modules are interfaced through a data base (we might want to call it information base) in which simulation trajectories are stored but also sets of parameter values, table-look-up functions, experiment descriptions, models, and finally note books.

Now, all the involved graphics commands can be thrown out of the simulation, and replaced by a simple STORE command to store data in the data base. Graphics are preformed by a separate interactive graphics program which picks up the required data from the data base, and combines them into fancy graphs which can be even more powerful than in any of the existing simulation languages, as the graphics program is now entirely independent of the simulation language, and can handle data produced by eventually any program. The same holds true for statistical computations which can be performed by a separate interactive statistics program hooked to the data base. And these are just two examples of programs which can easily be separated out. We shall see further examples in due course.

DATA BASE ORGANIZATION

This concept has first been implemented by Standridge (5) who designed the SDL/I simulation data language embedding a relational data base. Each relation consists of one or several tables ideally suited to store sets of trajectories from one simulation run (one table) or sets of trajectories from several replications (several tables). SDL/I has two user interfaces:

a) A batch-operated command interface (OIL definition language) for standardized data operations, and

b) a program interface (Fortran subroutines) for more general data manipulations.

These two interfaces make SDL/I a powerful tool, as standard type applications can be programmed quickly by creating a file containing a couple of simple command lines (usually 10 to 20 lines of code) which is then submitted as input to the (batch-operated) data base manager which interprets them, and executes the requested actions. More involved applications are performed by coding a Fortran program (or any program capable of calling Fortran subroutines) in which individual data entries in the data base can be accessed and modified according to any user-specified algorithm.

Disadvantages of SDL/I are as follows:

a) The command language should be interactive rather than batch-operated.

b) The command language should allow to refer to relations and variables by name rather than by index.

c) The command language is not powerful enough with respect to data manipulations supported. The only primitive data element of SDL/I is in fact a two-dimensional matrix. The command language should allow us to perform any data operation on these data elements in a calculus-oriented and user-friendly manner, say:

$$Z(TOM) = X(JOHN) + Y(MARY[3])$$

where TOM and JOHN are names of relations, MARY[3] refers to the third table (replication) in the relation called MARY, and X, Y and Z are column names (single trajectories = variables) in these relations. Thus, the above statement in fact denotes a vector operation. A good example of an interactive language for matrix manipulations is

MATLAB (6). An immediate solution to the problem might be to enhance MATLAB by two commands: LOADREL and SAVEREL to retrieve and store individual relations, and code the above command in MATLAB as:

```
LOADREL('SAMPLE.DBS',JOHN,J)
LOADREL('SAMPLE.DBS',MARY,M1,M2,M3,M4)
LOADREL('SAMPLE.DBS',TOM,T)
T(:,7) = J(:,4) + M3(:,6);
SAVEREL('SAMPLE.DBS',TOM,T)
```

Obviously, this is just a quick work-around rather than a true solution, as we still would have to refer to the variables by their relative position (Z being the 7th column of relation TOM, etc.) rather than by their mnemonic name. A better solution in the long run may be to throw away SDL/I altogether, and start from scratch using MATLAB as a basis.

d) SDL/I contains some graphics and statistics operators which really ought to be part of the application packages rather than being part of the data base manager.

e) SDL/I stores all data in one single file which is operated sequentially (for portability considerations) by simple READ/WRITE-commands. This makes the data access extremely slow and inefficient as the data grows in volume. A better (and almost as portable) solution would be to store just the data base organization in this central file. Each relation should occupy a file of its own (with a generic name: ZZ0001.DAT, ...). Where currently the actual data is stored, pointers should indicate the generic file name where the actual data can be found.

f) SDL/I does not provide for simultaneous multiple-access mechanisms. This feature is important for our last application: the multi-processor implementation of our simulation system. Again, the solution is conceptually simple: Rather than letting the user communicate directly with the data base manager, a message passing mechanism is placed between user and data base manager (... making the data base almost batch-operated again !). The data base manager can now reside in a CPU of its own, the visible "operating system" of which is the command interpreter of the data base manager. Each user sends a "program" (single command) to the data base processor through remote-job entry (RJE) whenever he requests a data base access. The queue manager of the local area network (LAN) queues the commands (requests) for the data base manager, requests which are passed through the LAN by the file server of the LAN. One user sitting at one workstation can perform a simulation, while another user on another workstation can simultaneously analyse data produced by previous simulations in a graphical or statistical sense. A specialized microprocessor can take measurements from a real plant which are simultaneously stored in the data base through the same access mechanisms. The project manager can retrieve data, compare simulated to measured trajectories graphically, write note book files, etc.. Using the graphics module, we may even decide to look at data which are concurrently generated by another processor running a simulation program. The only delay involved is the one created by the two message passing mechanisms involved. However, while the simulation still goes on, we may decide to do something else, e.g. compute some statistics, etc.. The simulation program shall never even notice. Experimenting with simulations finally feels just the same as walking a Voltage meter over to a power plant, and measuring and analysing data taken from that plant.

THE HISTORY OF SIMULATION SOFTWARE SYSTEMS

10 years ago, life was still fairly easy for a simulationist. He coded his CSMP program on a card deck simply by repunching all cards which were in error, sorting them by hand, and placing them in a sort of cookie sheet entitled "CSMP-INPUT". He then waited for a couple of hours, and finally found his listing in a special cabinet entitled "OUTPUT". The computer for him was just a magic box reading in punched cards, doing something with them, and finally printing out listings. He did not need to know anything except what was written in his CSMP manual.

The software systems themselves were simple enough. A CSMP manual could be digested in a couple of hours, and here we went analysing our 2nd order oscillatory network or whatever.

Alan Pritsker calls these systems *first generation simulation software*, and amazingly enough, most of the simulation systems in fashion today still fall into the same category (although the time of punched cards is long passed -- we used them for a couple of years to mark on their back side where we were, and placed them behind the name tag on our office doors ... until they were used up). The only real changes are with respect to the input medium (using a terminal and a text editor which have to be mastered), with respect to the output media (graphical output on either plotter or graphical terminal), and partly with respect to the execution mode (interactive modification of parameter values and rerun) -- thus, basically reducing the turn around. An ACSL manual is still as easy to read as the meanwhile oldfashioned CSMP manual.

COSY is a typical example of a *second generation simulation software system*, much more complex with respect to the features offered, but still a monolithic piece of software primarily for batch execution of large-scale simulation problems such as complex missile simulations, drum-boiler-turbine systems, etc..

SDL/I introduced the *third generation of simulation software*. A set of independent programs:

1) SLAM : combined continuous/discrete simulation (7,8)
2) SIMCHART : graphical postprocessor
3) AID : statistical analysis and distribution function fitting

were hooked around the data base (SDL/I) in the way described in the introduction to this paper. The major concern with this approach was the problem of appropriately embedding the software in the underlying operating system. Prior to executing any of these modules, a fair number of command language instructions (file assignments, etc.) had to be given to the operating system. The required knowledge of the operating system was quite substantial. Moreover, the same instructions had to be typed in over and over again, as e.g. the execution of a simulation program (from a users point of view an intrinsic operation) possibly involved:

a) assigning the input file

b) calling the preprocessor translating the model into an intermediate language (mostly Fortran),

c) reassigning some files,

d) calling the Fortran compiler (possibly several times for several files in a sequence),

e) calling the linker,

f) executing the linked program,

g) reassigning files again,

h) calling a graphics postprocessor.

Clever users started soon to write their own short command procedures to circumvent this boring repetition of one and the same command sequence, and this brings us to:

THE SIMULATION OPERATING SYSTEM

As we just learnt, the "primitive" operational instructions of a simulation user (to execute a problem, to delete a problem, to store a problem in the model base, etc.) consist of an entire series of "primitive" operational instructions of the underlying general-purpose operating system. In fact, many of the previously discussed data base operations are just internal communications between different program modules from which the user should be protected altogether.

It makes thus a lot of sense to combine all those previously developed short command procedures into one large "command procedure" (which, for efficiency considerations, may eventually again be coded in a high level programming language such as Pascal, making calls to operating system functions (on VAX/VMS: LIB$... - functions)), protecting the user (as in the good old days) entirely from any deeper understanding of the underlying general-purpose operating system. On a VAX, this program may be started from within the LOGIN.COM-file, followed by an immediate LOGOUT. In this case, the user never even enters VMS, and thus gets to believe that this command procedure constitutes the operating system of the machine. We call it therefore the simulation operating system. As high level languages protect the user from underlying machine language instructions, special-purpose operating systems protect him from the underlying general-purpose operating system instructions.

The first commercially available simulation operating system (*4th generation simulation software*) was TESS (9) released by Pritsker & Associates in summer 1984. This system contains as subsystems:

1) SLAM-II : enhanced version of SLAM

2) SDL/II : enhanced data base

3) AID : data analysis module

4) SIMCHART/II : enhanced SIMCHART (color graphics)

5) An interactive color-graphics PERT-network builder

6) A form-driven data-input builder

7) An interactive color-graphics facility diagram builder

8) A rule builder for simulation run-time animation

Although the TESS system as a whole is pretty large (roughly 125'000 lines of Fortran code -- according to private communication by Charlie Standridge), each individual program module is manageably small. The TESS language itself is amazingly simple. Each command consists of one out of 4 verbs (BUILD, REPORT, GRAPH, and DEFINE) followed by one out of 8 nouns (NETWORK, CONTROLS, SCENARIO, DATA, SUMMARY, FACILITIES and RULES) -- reminding us very much of the famous "ADVENTURE" games ! The following table states all meaningful combinations of this orthogonal structure.

		MODELING & SIMULATION			INPUTS & RESULTS		PRESENTATION SPECIFICATIONS		
		NETWORK	CONTROLS	SCENARIO	DATA	SUMMARY	FORMATS	FACILITIES	RULES
FUNCTIONS	BUILD	X	X	X	X	X	X	X	X
	REPORT		X		X	X	X		X
	GRAPH	X			X	X	X	X	
	DEFINE				X	X			

Each node (that is: each cross in the table) represents one program module. Thus, TESS consists of 20 independent program modules, all communicating with each other through the data base (21st program), unified through the TESS command language (22nd program). This approach protects the user from the nitty-gritties of the underlying operating system, and to a large extent even from a direct confrontation with the data base manager, as the majority of data base access commands are executed by the TESS language itself.

The disadvantage of the approach chosen lies in the fact that we cannot possibly foresee what TESS commands the actual TESS user would like to have at his disposal. To give an example: Here comes a user who claims that his SLAM-II program is so large that he could not possibly execute it in an interactive way. In fact, the accounting system at his installation has been set up to give a bonus for night-time operation. For this reason, he would like to submit his SLAM-II program to the batch queue for night execution only. Unfortunately, this is the end of TESS usage, as the TESS designers had not foreseen this request (and obviously never could foresee all possible needs). The poor user has to resort to good old VMS, digging in the files to find his SLAM program (no easy task as his program forms part of the model base!), and submit it to the batch queue manually. This again requires a good knowledge of both VMS and the internal structures of TESS.

For these reasons, MIDGET (10,11) goes yet another step further. Rather than providing one static TESS-like command language, MIDGET provides for a framework for the development of such simulation operating systems (called "development systems" in MIDGET). A ".DSD"-file contains the syntactic description of the command language together with the documentation to be included in the interactive HELP-library. A ".COM"-file contains the semantic routines associated with each command. A Pascal-coded compiler-compiler (NEWDEV) which is part of the MIDGET kernel system then generates the individual menu-driven development system. Each development system is in itself then a TESS-like language for a particular task. Inclusion of a new command is a matter of ½ hour programming, development of a completely new development system is

a matter of roughly 3 days programming. Currently available are development systems for a couple of simulation languages (including: ACSL, SLAM, SDL/I, GASP-V/INTERACTIVE), for the TEX word processor of Knuth (12), for a syntax analysis system (13), and for playing games on the VAX (our current QIX record is at 29000 points -- how about that!). The syntax analysis system is of particular interest: Here NEWDEV is a compiler-compiler to create the environment in which to run another compiler-compiler for LL(1) grammar analysis.

MULTIPROCESSOR IMPLEMENTATIONS

Multiprocessor implementations of the previously elaborated concepts are not far down the road. The first generation of local area networks (LAN's) provided for communication between terminals and computers. At ETH Zurich, we have installed the Sytek Localnet 20 (14) with several 100 PCU's and more than 100 miles of coax cable. This network makes all main frames and some process computers available to hundreds of users sitting at 100s of terminals throughout the Campus. Gateways connect this system to the Sytek network of the University of Zurich and to Telepak, the Swiss X25 switching data network. At the University of Arizona, we have installed several local area networks within the Electrical & Computer Engr. Dept. (including Localnet 20, Ethernet, and Concord Datasystem's Token Net). A colleague of ours (Ralph Martinez) has a project ongoing to build gateways among these networks, and also to the Defense Data Network (15).

The second generation of LAN-software shall include enhanced capabilities for:

1) file transfer protocols,

2) electronic mail,

3) printer servers,

4) queue managers (for the ports), and

5) remote-job-entry (RJE) protocols.

Ralph Martinez is working on such a system (for the Sytek network), and the software shall be released in the second quarter of 1985. A specialized data base processor is already on the market (INTELs iDBP 86/440 (16)) which allows to store files in a host-independent filehandling system hooked to a LAN (Ethernet).

Currently, we are about to design the new data language (on the basis of MATLAB). The first implementation shall be made available on VAX/VMS. However, the program shall be kept as portable as possible, and we consider to transfer it lateron to an INTEL 286/310 running iRMX 286 hooked to the Localnet 20, where it shall reside as a special-purpose operating system. In fact, this "operating system" shall look like a time-shared, batch-operated main-frame system. "Programs" (data access commands) are passed to it by means of the RJE network protocols.

A prototype of the interactive color-graphics system (including threedimensional graphics with hidden lines removed, envelope graphics, split-screen mode) is already available (on VAX). It shall be reimplemented during the coming months (on an INTEL 286/310) to make it commercially available.

We are also working on a raw data analysis module implementing the concepts of SAPS (17).

A wide range of simulation software is already available including DARE/INTERACTIVE, DESCTOP, and GASP-V/INTERACTIVE (all on VAX/VMS). All systems shall obtain the required interfaces to link them to the simulation data processor.

REFERENCES

(1) F.E.Cellier, "Simulation Software: Today and Tomorrow", Proc. of the IMACS Symposium on Simulation in Engineering Sciences, Nantes, France, May 9-11, 1983, (J.Burger, Y.Varny, eds.), North-Holland Publishing Company, pp. 3-19.

(2) F.E.Cellier, A.P.Bongulielmi, "The COSY Simulation Language", Proc. of the 9th IMACS Congress on Simulation of Systems, Sorrento, Italy, September 24-28, 1979, (G.Savastano, G.C.Vansteenkiste, L.Dekker, eds.), North-Holland Publishing Company, pp. 271-281.

(3) F.E.Cellier, *et alia*, "Discrete Processes in COSY", Proc. of the European Simulation Meeting on Simulation Methodology, Cosenza, Italy, April 9-11, 1981 (F. Maceri, ed.), to appear.

(4) T.I.Oren, "Computer-Aided Modelling Systems", *Progress in Modelling and Simulation*, (F.E.Cellier, ed.), Academic Press, 1982, pp. 189-203.

(5) C.R.Standridge, A.A.B.Pritsker, "Using Data Base Capabilities in Simulation, *Progress in Modelling and Simulation*, (F.E.Cellier, ed.), Academic Press, 1982, pp. 347-365.

(6) C.Moler, "MATLAB Users' Guide", Technical Report: CS81-1 (Revised), August 1982, Dept. of Computer Science, University of New Mexico, Albuquerque, NM 87131, 60p.

(7) A.A.B.Pritsker, *Introduction to Simulation and SLAM-II*, Second Edition, Halsted Press, 1984, 612p.

(8) C.D.Pegden, A.A.B.Pritsker, "The Many Interfaces of SLAM", *Progress in Modelling and Simulation*, (F.E.Cellier, ed.), Academic Press, 1982, pp. 247-261.

(9) Pritsker & Associates, Inc., *The TESS User's Manual*, P.O.Box 2413, West Lafayette, IN 47906, May 1984, 515p.

(10) F.E.Cellier, M.Rimvall, "MIDGET: A Framework for Developing Special-Purpose Operating Systems", in preparation.

(11) M.Rimvall, F.E.Cellier, "MIDGET, Ein flexibles, simulationstechnisches Entwicklungssystem", Proc. of the ASIM'84 Symposium, Vienna, Austria, September 25-27, 1984, Springer, Informatik Fachberichte.

(12) D.Knuth, *The TEX Text Processing System*, Digital Press, 1979.

(13) A.P.Bongulielmi, F.E.Cellier, "On the Usefulness of Deterministic Grammars for Simulation Languages" *Simuletter*, Vol. 15, No. 1, 1984, pp. 14-36.

(14) Sytek, Inc., *LocalNet 20, Reference Manual and Installation Guide*, Document No. R2000.1/1-02B, 1225 Charleston Rd., Mountain View, CA 94043.

(15) R.Martinez, J.Sheppard, "Internet Gateway Protocols and VLSI", Proc. of the Government Microcircuit Applications Conference, November 6-8, 1984, Las Vegas.

(16) Intel, Inc., *iDBP DBMS Reference Manual*, 1983.

(17) H.J.Uyttenhove, *SAPS System Approach Problem Solver*, Ph.D. Thesis, SUNY Binghampton NY, 1981