Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

NEW DIRECTIONS FOR THE DESIGN OF ADVANCED SIMULATION SYSTEMS

Michael G. Ketcham, Ph.D.
Department of Industrial Engineering
University of Massachusetts
Amherst, Mass. 01003

John W. Fowler
Department of Industrial Engineering
Texas A&M University
College Station, Texas 77843

Don T. Phillips, Ph.D., P.E.
Department of Industrial Engineering
Texas A&M University
College Station, Texas 77843

ABSTRACT

This paper will review the philosophy behind advanced simulation methodologies and the advantages these concepts offer to simulation analysts. It also proposes a generic design of an integrated simulation system based on an analysis of theoretical studies, a review of recently developed simulation systems, and current research.

INTRODUCTION

In 1979, Tuncer Oren and Bernard Zeigler [1] identified several weaknesses in conventional simulation languages and proposed developing new simulation systems based on two concepts. First, simulation languages should functionally separate the logically distinct stages in model development, and, second, simulation environments should be created to take advantage of current computer capabilities for database management, graphics, and program verification.

Over the last five years several special purpose simulation systems and general purpose languages have been developed that, either independently or influenced by Oren and Zeigler, have included a functional separation of model development, entity description, scenario specification, and output analysis. Furthermore, in separating these capabilities, these systems use sophisticated database management software to maintain separate databases for storing model components, entity attribute values, and simulation output (see Golub and Soper [2], Ludwigs [3], Pegden [4], Pritsker and Associates [5], Trott and Franz [6], Willis and Austell [7]).

In addition to these major architectural features, recent simulation systems have incorporated tools to facilitate the user's interaction with the system. These tools have included graphical input and output, the ability to develop models hierarchically, and the separation of different classes of users with different privileges in developing and experimenting with system models (see Comer [8], Engelke, et al. [9], Rose [10]). As a result of these developments, we are now seeing the first generation of simulation packages based on radically new design concepts. This paper reviews the advantages these concepts offer to simulation studies and proposes a generic design for an integrated simulation system based on recent theoretical studies, recently developed simulation systems, and current research. We intend to present a general architecture that will provide a unified view of advanced simulation methodologies and provide a framework for considering the implementation of specific simulation systems.

SYSTEM DESIGN

Recent studies of decision support have identified four functions provided by decision support software: 1) it must provide capabilities for model development and modification; 2) it must provide capabilities for interactive data entry; 3) it must allow users to execute models with user-specified experimental conditions; and 4) it must provide flexible, user-selectable display formats.

Simulation systems used for decision support must provide these same four capabilbities. Toward this end, we have distinguished four frames that provide four distinct views of simulation models and simulation output. These include:

* Model Frame, for model development
* Entity Representation Frame, for entering parameters describing system entities
* Experimental Frame, to specify experimental conditions
* Output Analysis Frame, for examining simulation results

We have adopted Oren and Zeigler's terminology to characterize three of these frames, although our account of their function is drawn from several practical studies in addition to theoretical research. The Entity Representation Frame is not explicitly represented in Oren and Zeigler's formulation.

Figure 1 gives an overall view of our generic simulation-based decision support system, with eight principal components:

* User Interface Controller
* Model Frame
* Entity Representation Frame
* Experimental Frame
* Output Analysis Frame

* Execution Controller
* Statistical Capabilities Package
* System Library

Each of the four frames is itself composed of several parts, including an interactive user interface, an associated database, and a communications link with other system components. Also, simulation software should communicate with external databases that hold historical data from the actual system being modeled. This more complete picture of the system is given in Figure 2, with the User Interface Controller removed for clarity.

Using Figure 2, we can also represent the system's data flows. Each of the four frames interacts with the System Library, which acts as a controller for maintaining an integrated data flow. The Entity Representation, Experimental, and Output Analysis frames interact with the Statistical Capabilities package which provides statistical routines for data analysis or experimental design. The Entity Representation Frame also interacts with an actual system database to retrieve current data about the behavior of the real system.

The following sections discuss the four frames along with their interactions with other system components. However we should emphasize that many details regarding component interactions will be implementation decisions that we have examined but have not tried to resolve in this paper. It should be an implementation decision, for example, whether there should be "hard" or "soft" boundaries between frames. A hard boundary would require the user to make a deliberate choice to enter one frame or another, whereas with soft boundaries the system would automatically shift contexts if the user wishes to perform a function that is logically part of another frame. Regardless of the decisions made for implementing these components, they will share the same basic architecture and the same underlying logic, and it is this underlying logic that we have tried to capture in our discussion of the fours frames.

MODEL FRAME

The Model Frame consists of three principal parts: a user interface for interactive model development, a data base for storing models and component modules that can be linked to form complete models, and a model translator that converts the user defined model into an executable form.

The user enters the Model Frame through the User Interface Controller. Once in the Model Frame he or she can build new model components and save them in the model database or can retrieve and modify existing model components. Models may be developed graphically, using the cursor or a mouse to move icons across the screen as a way of representing system configurations. In effect, the user draws a network on the CRT while the system software automatically maintains a data base of model specifications in response to graphical inputs.

This kind of graphical entry is part of a development editor (MDE), which should also allow for more traditional statement entries and which should provide context-sensitive HELP messages. The MDE should provide support for automatic documentation of established software development stages such as requirements definition, design, specifications, and coding. It should also provide immediate, interactive checks of syntax and should allow the user to freely shift between graphical and statement formats and other input modes. The MDE should be able to automatically generate statements from graphical input and a graphical representation from statement input.

The model itself is specified within a model development language (MDL). In different systems there may be different types of MDLs designed for different types of models. That is, one implementation may emphasize continuous modeling, while another may emphasize process or event-oriented simulations. For any of these model types, however, the MDL should be structured so that models can be developed modularly and hierarchically. The MDL should guide model development according to the logic of the system being simulated by tailoring modeling constructs as closely as possible to the features of the system being modeled. At the same time, the hierarchical structure of the MDL should allow users to run high-level prototype models to examine gross system performance and also be able to specify multiple levels of detail, if needed, for more detailed representations of the system. A corporate, model for example, may allow a broad view of plant operations for high-level planning, but the broad corporate model may permit another set of experiments that depict transportation flows among plants. Or, moving down the model hierarchy, it may allow production engineers to examine the behavior of a particular plant or production cell.

Because of the requirement for hierarchical modeling, we see system models as being comprised of model components that can be linked to form complete, executable models. This concept of system modeling leads to three views of the model code. First, there will be model components store separately in the model database. Second, these model components may be grouped through a system linker to collectively form a "declared model."

A declared model may combine several model components, any of which may also be used in other declared models. As a result, a declared model presents a logical view of the real system that may vary from other views presented in other declared models. Before being declared, a grouping of model components will first be verified by a system linker to insure that labels, module parameters, etc., are consistent among the collected model components. Once these linkages are verified, the model is declared in the System Library for use by other system users. Pointers to the model components and the groupings that make up a declared model are maintained in the System Library. When a user asks to execute a "model," the library identifies the separate modules that are to be joined to make up the executable code. An executable model exists only at run time and is the collected, translated, and linked form of a declared model that can be run as an experiment.

The movement from model components to an executable model is carried out by the translator and linker. Ideally, the translator should have syntactic and lexical checks and at least minimal logic checks for model verification. The result of model translation may be either executable object code if the MDL is

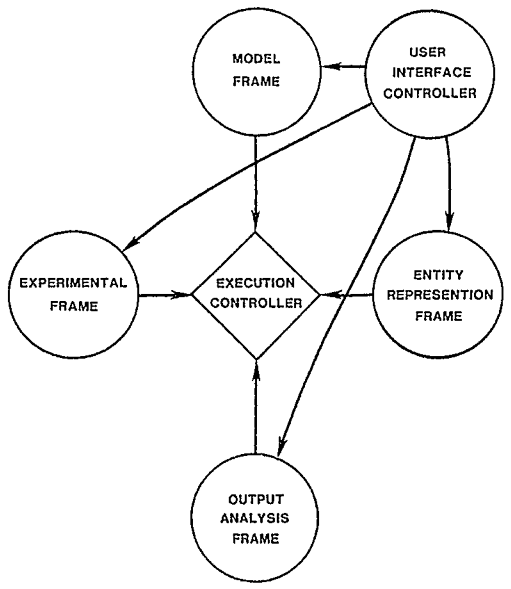

Figure 1

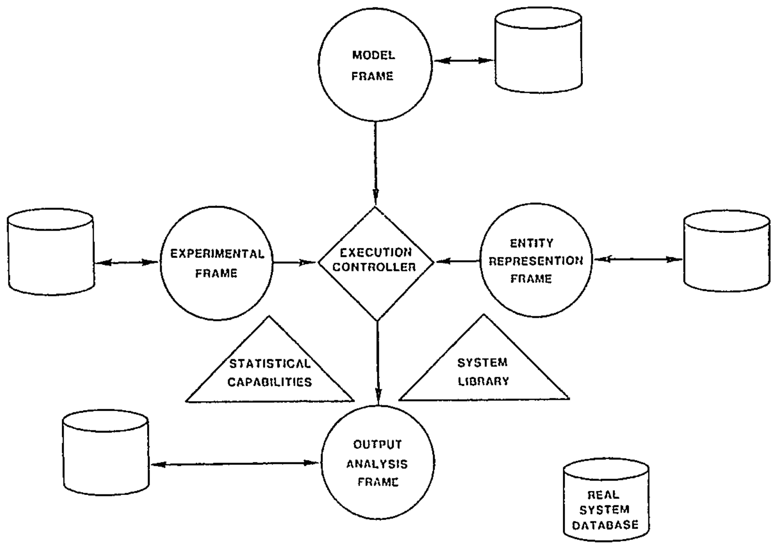

Figure 2

directly compiled, or input statements for an existing simulation language, such as SIMSCRIPT, SLAM, CMSP, etc. Furthermore, as model components are translated into an executable form, specifications for entities described in the model are provided to the Entity Representation Frame where initial attribute values and parameters will be entered.

ENTITY REPRESENTATION FRAME

Like the Model Frame, the Entity Representation Frame includes a user interface, access to its own database, and control software that supervises the user's interaction with the database. The Entity Representation Frame also has a close interaction with the system library and system statistical capabilities, and can potentially communicate with actual system data maintained in an external database such such as a management information system (MIS).

As a user translates a model in the Model Frame, the simulation system establishes entity descriptors to characterize entity attributes and process parameters. Entities represented in the Entity Representation Frame include temporary entities (transactions), permanent entities (resources or facilities), queues (storages), etc. The descriptors used to characterize these entities thus include entity attributes, part routings, processing times, initial resource levels, queue capacities, etc.

The Entity Representation Frame accepts requirements for entity descriptors from the Model Frame and prompts the user to enter the required data through a Data Entry Interface. Separating the Entity Representation Frame from the Model Frame allows users to enter information about the actual system using the Entity Representation database without knowing the structure of the model or knowing how the data will eventually be formatted and used during a simulation run, so that data can be entered by data entry or production personnel. The separate Entity Representation Frame, in fact, allows data entry to be fully automated by having the Entity Representation database updated from an operational database, such as an MIS, as the real system changes over time.

In addition to maintaining a database of system descriptors, the Entity Representation Frame will allow users to perform statistical tests by means of the Statistical Capabilities Package to determine appropriate distributions and distribution parameters to be used in generating artificial data for the simulation. Also, by interpreting experiment specifications, the Entity Representation Frame can automatically generate initialization statements for entity descriptors using values stored in the Entity representation database, and can generate multiple copies of system entities or form aggregate entities, depending on the scenario a user wishes to explore. Thus, these kinds of entity initialization are removed from the user's responsibility, unless of course, he or she chooses to override these defaults in specifying an experiment through the Experimental Frame.

EXPERIMENTAL FRAME

The design of this system allows a user three sources of flexibility in specifying experimental conditions. First, he or she can specify which declared model is to be executed. Second, he or she can specify decision variables and their initial values. Third, he or she can specify run conditions, such as stopping rules, number of replications, random number seeds, etc. Also, within the Experimental Frame, the user will be able to specify restrictions on statistics collection, if the user is confident that certain types of statistics will not be needed in output analysis. The actual computation and display of statistics will take place in the Output Analysis Frame.

In executing an experiment, the Experimental Frame interacts closely with the System Library and Statistical Capabilities Package. While in the Experimental Frame, the user can browse through and select from the library of declared models. When the user requests that an experiment be performed, the system will first check the library to see if the same experiment has already been conducted. If so, the user will be notified and he or she will be given the choice of rerunning the experiment or using previous results. The user can also query the library to determine if related experiments have been conducted with similar values for user-specified decision variables.

When the user does conduct an experiment, the library will record information regarding the time of the run, the user's name, the model executed, the set of system descriptors read from the Entity Representation Frame, and run results(which would phsically reside in the Output Analysis database). This library record can include any user comments regarding the purpose of the run or the run results.

The Experimental Frame database itself stores user-specified scenarios, including number of experiment replications, initial variable values, specifications for overriding Entity Representation defaults, etc. These scenarios can later retrieved and modified for further experimentation.

The Experimental Frame will also interact with the system's Statistical Capabilities Package to enable the user to determine elements of experimental design, such as the number of replications, factorial experiment design, and search strategies for system optimization.

The Experimental Frame should also give the user the option of conducting an experiment as a batch or interactive process. As an extension to batch execution, the system should provide pause or interrupt capabilities so that the user can review intermediate results and intervene in the simulation, if necessary, as we will explain in the discussion of output analysis.

OUTPUT ANALYSIS FRAME

Output analysis involves two distinct functions, the recording of observations and computation of statistics. For most existing simulation languages, statistics such as mean values, standard deviations, etc., are computed during the simulation run. These statistics are generally aggregate values determined by the variations in system state variables. For most languages, the user must accept language-specific default statistics or must specify, in the model it self, the statistics to be computed. Recording observations, in contrast to computing statistics, involves writing the value of state variables, such as the number of items in queue, event occurence times, utilization of resources, etc. into an output database. We have followed the lead of ICAM [5], and have separated observation and computation because the user should not have to know at run time what statistics he or she may want to see from the finished simulation. The aim of a simulation system of the type we are describing is to provide flexible support for decisions regarding systems design. In this context users will frequently not know in advance which statistics will provide useful or necessary information about system performance. The design of the Output Analysis Frame, therefore, should allow for recording a generous set of observations as a default, and should then allow users to query the simulation results and to display the results in user-determined formats. This design, however, does not preclude the user specifying, in the Experimental Frame, statistics to be computed during the run.

Separating observations from statistics computation gives the Output Analysis Frame a wide range of capabilities. Primarily, it allows the user to view simulation results either as trace data or aggregate data, with statistics being computed according to the user's decision requirements. To derive aggregate statistics, the output analyzer scans the database of observations generated during simulation execution and computes and displays the required values. Once these values are computed, they are stored in the output database and identified so that the Output Analysis Interface can immediately display these results in response to future requests.

Analysts may use both trace data and aggregate data to verify and validate system models. If an analyst finds unexpected results, he or she can review a trace and ask unanticipated questions about system performance measures at any point in the simulation. Anyone who has gone through the process of debugging a complex model will see the value of being able to retrieve information about system variables without writing debugging statements for repeated runs of the same model. The Output Analysis Frame also allows an analyst to view traces as the simulation is running. If the analyst interrupts the simulation in response to the interactive run-time displays, he or she can compute intermediate statistics and terminate the run on the basis of these statistics.

Decision makers can also tailor displays for both types of data to meet their requirements, as well. Frequently, user's reports will be non-graphical, summary reports in tabular form. The user can also have results displayed in various graphical forms, including histograms, line graphs, pie charts, etc. Displays should allow for windowing, light-pen or mouse interactions, and other user-defined image manipulations. Similarly, the decision maker should be able to use recorded observations to produce an animated trace, perhaps overlaid on graphical image of the real system.

After examining the simulation results, the user can choose to retain the run-time observations, or computed statistics, or both in the System Library. He or she can also create a "comments" file associated with the current run to record his or her reactions to the data obtained. Both the results and the comments file can then be accessed through the System Library by any member of the decision team who may use these, and related results, to guide organizational planning.

USER ENVIRONMENT

The institutional character of large scale simulations has been the driver for many of the software engineering and decision support techniques incorporated into simulation software. Traditionally, simulation packages have been developed to be run and interpreted by specialists, such as the members of an operations research group. Large scale simulations for a corporation or government agency, however, may be run, maintained, and modified over a period of years by people other than members of the original design and development team. The requirements of this extended class of users has placed new demands on simulation modeling and output analysis:

* For large-scale simulations, logic of a monolithic program becomes difficult, if not impossible, to verify and document so that modular design becomes a necessity in simulation design as in other large-scale software projects.

* Because the model may be used and modified by people other than members of the original design team, simulation systems need to support standardized modeling techniques and standardized documentation.

* Because the system may be used by an extended class of users, there need to be procedures for running the model and changing experimental conditions that do not require the user to be familiar with the model code. Instead, the simulation system software should provide an appropriate view of the model and results for each of the potential user groups.

* Advanced simulation techniques require the user-adaptable interactions that have come to be associated with decision support systems. These include interactive queries, easily established "what-if" scenarios, interactive, user-specified techniques for data analysis and output display, etc.

A system of the type we have described is intended to support several classes of users throughout the life cycle of a simulation model. The models themselves will be developed, as they are at present, by trained programmers and analysts. The graphical modeling features embodied in the Model Frame, however, are intended to make the completed models comprehensible to other users who will need to review or exercise the model. Because the design specifies a separate Entity Representation Frame, model-developers can separate data collection from the job of capturing the logic of the actual system. the Data Entry Interface can be tailored to the needs of additional users, which may include the analysts responsible for determining relevant distributions and statistical parameters but which may also include production personnel or data entry clerks who can enter data without knowing the internal form of the model itself. The Data Entry Interface may, in fact, be tailored to accept data from an actual system data base so that, once a model is developed and declared, data can be routinely entered, without the need of trained analysts.

Finally, the Experiment Interface allows analysts to specify simulation control conditions (such as stopping rules), decision variables, and special statistics collection points. A decision-maker may choose to ignore the more specialized simulation control conditions, once defaults have been established by analysts. In this case, a manager, engineer, or other decision maker can execute the model as he or she chooses, manipulating the decision variables of interest at each execution. These experimental capabilities, combined with the fact that the model representation is designed to be comprehensible to non-technical users, provide unparalleled flexibility in using simulation as a management decision tool. Commonly, an experiment's results will be reviewed by the users who established the experimental conditions in the Experimental Frame. The Output Analysis Interface, however, allows queries and displays to be tailored to the needs of higher level executives, policy makers, or other users still further removed from the original model formulation.

As we have explained, many of the details of implementing a simulation system of this type will be based on the specific circumstances for which the system will be designed. However, we have emphasized the need for adapting the architecture of the simulation system to the user environment. Regardless of the implementation decisions, advanced simulation systems should be founded on an underlying logic and software architecture that will provide for multiple user views of a simulation and that will meet user's requirements for planning and decision making for several classes of user.

In summary, these advanced simulation systems present flexible modeling and output analysis features, with multiple views of a simulation, all supported by sophisticated software and by an integrated system design. It has been the purpose of this paper to provide a generic system design, midway between the general requirements that have been identified in theoretical studies and the specific implementations used in recently developed simulation packages. We feel that a generic design of this type provides the simulation community with a basis for describing new simulation systems and a framework for future studies of simulation capabilities.

REFERENCES

[1] Oren, T. I. and Zeigler, B. P., "Concepts for advanced simulation methodologies," Simulation, 69-82, March, 1979.

[2] Golub, J. and Soper, W. A., "Prototyping for Naval Battle Group Simulation Development." Record of Proceedings: 16th Annual Simulation Symposium. IEEE Computer Society Press, 79-82, 1983.

[3] Ludwigs, H., "Simis II—An Environment for Material Flow Systems Simulation." Record of Proceedings: 16th Annual Simulation Symposium. IEEE Computer Society Press, 69-78, 1983.

[4] Pegden, C. D., Introduction to Siman. Systems Modeling Corp., 1982.

[5] Pritsker & Associates, ICAM Definition Method: IDEF2 Dynamics Modeling, Architects Manual. Softech, Inc., 1980.

[6] Trott, K. C. and Frantz, F. K., "A Detailed Interactive Simulation Sytem for Developing Command and Control Systems." Record of Proceedings: 16th Annual Simulation Symposium. IEEE Computer Society Press, 11-31, 1983.

[7] Willis, R. R. and Austell, W. P., "GMSS: Graphic Modelling and Simulation System." record of Proceedings: 16th Annual Simulation Symposium. IEEE Computer Society Press, 315-331, 1982.

[8] Comer, E. R. "Structured Model Specifications with a Supportive Simulation Architecture." Record of Proceedings: 15th Annual Simulation Symposium. IEEE Computer Society Press, 315-331, 1982.

[9] Engelke, H., Grotrian, J., Scheuing, C., Schmackpfeffer, A., and Solf, B., "Structured Modeling of Manufacturing Processes." Record of Proceedings: 16th Annual Simulation Symposium. IEEE Computer Society Press, 55-68, 1983.