Proceedings of the 1987 Winter Simulation Conference
A. Thesen, H. Grant, W. David Kelton (eds.)

THE FLEXIBLE ADA SIMULATION TOOL (FAST) AND ITS EXTENSIONS

Michael L. Samuels
James R. Spiegel
Ford Aerospace and Communications Corporation
4920 Niagara Road
College Park, MD 20740, U.S.A.

ABSTRACT

Discrete-event simulation is often considered the method of last resort because of the excessive time needed to develop and debug models, as well as run experiments and analyze results. The Flexible Ada Simulation Tool (FAST), is designed to alleviate these problems through extensive use of Ada design methodology (Ada is a registered trademark of the U.S. Government, Ada Joint Program Office). Object-oriented design permits rapid expansion of language features, including interfaces to data base systems, statistical tools, and graphics packages. Ada exception handling greatly improves the user interface and debugging environment, while Ada tasking provides the user with control of the computer resources. How these features contribute to the reduction of major simulation bottlenecks is explored in this paper. The ease in which FAST has been extended to encompass all facets of computer simulation, from creating models to generating reports, is also demonstrated.

1. INTRODUCTION

Computer models are commonly used during the Design Phase of the Project Life Cycle to analyze the logic and performance of proposed system architectures. Because of its flexibility, discrete-event simulation is ideal for such analysis: alternatives can be readily compared without resorting to many of the simplifying assumptions required by other modeling techniques. Unfortunately, simulation is often considered the method of last resort because of the excessive time needed to create and validate the model, implement and debug the program, and analyze and present the results.

The Flexible Ada Simulation Tool (FAST), currently under development at Ford Aerospace and Communications Corporation, College Park, Maryland, is designed to meet the time constraints imposed by various life cycle deadlines while maintaining the inherent flexibility of discrete-event simulation. The features of FAST contributing to this increase in productivity are discussed.

First, the bottlenecks hindering simulation development will be summarized. The key features of FAST, including the design of the user interface, will be described in terms of these modeling barriers. The use of Ada software engineering techniques will be discussed in terms of the FAST design, demonstrating how easily Ada permits extensions to the modeling environment. These extensions arise from a more global view of simulation, incorporating problem conception, model development, experimental definition, output analysis, and report generation into the modeling framework.

2. BOTTLENECKS IN SIMULATION ENVIRONMENTS

Henriksen (1983) notes that two major bottlenecks have plagued simulation development from the outset:

1. the barrier between editing and compiling;

2. the barrier between compilation and runtime support.

Developing a syntactically correct simulation program can be tedious. A typical scenario is to use a general purpose editor to create the simulation, submit the program to the compiler, and then wait several minutes or even hours until the program listing is produced. This delay would be more acceptable if the compiler flagged a majority of errors. However, many simulation languages are replete with awkward data structures and program statements resulting from the desire to provide English-like code while maintaining compatability with previous versions of the language (Bratley, Fox and Schrage, 1983). The lack of strong typing hides many syntax errors until runtime.

The second major barrier is even more problematic. Execution errors are divided into programming mistakes (e.g., division by zero) and exception conditions raised during a particular simulation run (e.g., a buffer overflow). Simulation errors are often subtle; error messages may not occur until long after the event causing the problem. Since the usual approach to debugging is to record the state of the system at pre-defined "breakpoints", the time needed to locate and correct runtime errors depends on proper placement of these print statements.

While many PC-based simulation packages have greatly reduced these barriers, productivity has not been increased to the extent needed for rapid model development. Compilation remains a slow process, and trace files continue to provide the primary runtime support mechanism. The result is that rapid prototyping of alternative system architectures remains an elusive goal. To reach this goal, the Flexible Ada Simulation Tool is designed to take advantage of mainframe power while improving the user interface along the lines of PC-based simulation.

2.1. The Edit/Compilation Barrier

FAST reduces the bottlenecks between editing and compiling in two ways: the use of a syntax-directed editor and the elimination of external program calls to high-level languages.

Language Sensitive Editor. FAST has been implemented under Digital Equipment Corporation's VMS operating system and takes advantage of the Language Sensitive Editor (LSE). LSE provides verified coding structures for every programming statement, as well as on-line assistance for specifying attributes of these structures. The components of a simulation model are defined as "tokens" in the editor. The user requests help by placing the cursor on the token and pressing "CTRL-E". A pop-up window appears, and the user selects the desired option with the "UP" or "DOWN" cursor keys. Pressing the carriage return replaces the token with the properly coded simulation structure or a lower-level token (Figure 1). For example, the GENERATE statement is used to create new events. Three attributes are used for the interarrival distribution, and a fourth indicates the number of events that should be created. If the use selects a constant interarrival time, the editor indicates that attribute 1 is designated by the "FIXED" option, attribute 2 is the constant interarrival time, and attribute 3 is given the value of "0". If a UNIFORM distribution is desired, the editor indicates that the second and third attributes represent the lower and upper bounds, respectively.

The Language Sensitive Editor follows the conventions of the VAX EDT Editor so that experienced users can develop models without being burdened by the on-line assistance capabilities described above. As Henriksen (1983) points out, there is a fine line between editors that provide enough assistance to the beginner without becoming a burden to the experienced user; the Language Sensitive Editor achieves this balance.

Fast Compilation. FAST provides a user-friendly interface that guides the beginner through program development, debugging, and runtime execution without constraining the experienced modeler. A series of four menus ("Max", "Net Flow", "Limits", and "State") provide on-line assistance for every possible action, including opening files for input and output, modifying files, and transferring control between these four levels of control. These levels are discussed in detail in Section 3. What is important to note with respect to the barrier between editing and compiling is that FAST "compilation" occurs during the transfer from the "Net Flow Menu" to the "Limits Menu". If a compilation error is found, a

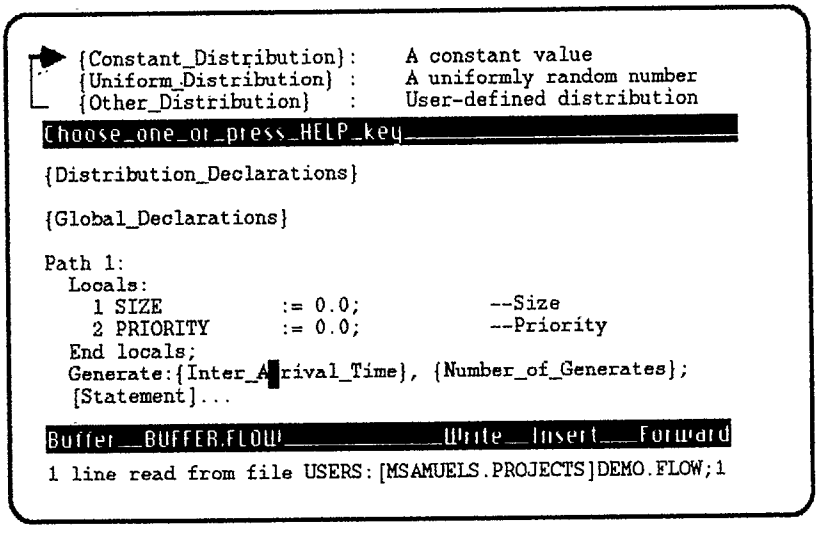

FIGURE 1. LANGUAGE SENSITIVE EDITOR. Statements in braces and brackets are "tokens" that expand to correct FAST structures. In this example, CTRL-E was pressed with the cursor on the interarrival token, opening the pop-up help window for setting input distributions.

descriptive message is typed to the screen at the point of the error; one command places the user back in the editor. If no errors occur, the user is automatically transferred to the next level of control.

English-like macros control all aspects of the simulation. A parser translates the Net Flow program into the appropriate actions. Though FAST is implemented in Ada, the modeler does not use Ada to create simulations. Nevertheless, FAST provides the user with the basic constructs found in high-level languages (Ghezzi and Jazayeri, 1982), including sequential statements (e.g., "Assign"), logic statements (e.g., "If...Else...Endif"), and repetition structures (e.g., "Loop...Endloop"). Simulation functions are also provided (e.g., "Request"), and statistical results can be used to trigger events at any time during the simulation (e.g., "AvgQueueLen"). An important design decision made early in FAST development was that this macro language would provide all necessary programming constructs. In many simulation languages, external calls to FORTRAN and other high-level languages are needed to provide programming details. This extra step greatly increases compilation time.

2.2. The Compilation/Execution Barrier

FAST reduces the bottleneck between compilation and runtime support in two ways: interactive monitoring of simulation statistics and interactive control. These will be discussed in the context of debugging and sensitivity analysis.

Interactive Debugging. FAST allows visibility into the "internal" structures of a simulation. Every traffic entity is assigned an identifier, or "passport", when it enters the system. A "Active Passport Summary" provides information about the current status of each entity, including the Path number and programming statement currently being executed. The "Future Events Queue" provides a list of passports in order of execution, while individual "Queue" displays list the passports waiting for service from a particular resource (see Figure 2).

FAST also includes a step mode, which allows the user to increment the model one event at a time. In conjunction with the display pages described above, the step mode lets the user observe the very fine details of the system. One proven debugging technique is to use the "Set_Duration" command to stop the simulation just prior to a simulation error condition. A "Go" command will cause the simulation to run to this point. The user may now proceed with the "Step" command, observing the Future Events Queue and other displays, to determine exactly when, where, and why the error occurred. Clearly, such a capability is invaluable to the debugging process.

Sensitivity Analysis. In addition to the display pages mentioned above, FAST provides the user with statistical summaries of queues, global and local variables, resource utilization, and several other runtime parameters. The user interface not only displays these values throughout the simulation run, but also allows the user to change values. For example, if a communications network

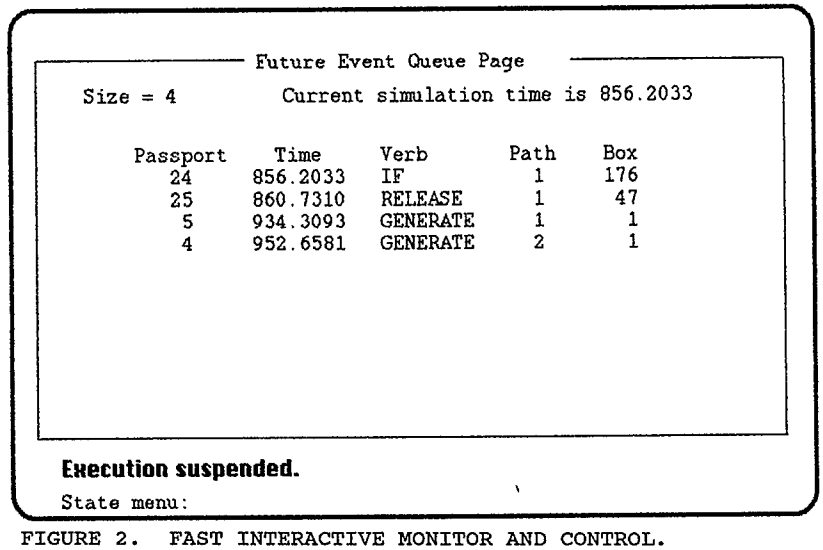

FIGURE 2. FAST INTERACTIVE MONITOR AND CONTROL. The Future Events Queue is one of many runtime displays available to the user. Events are listed in time order with an identifier ("Passport"), the associated data flow ("Path"), and the next statement to be executed ("Verb" and "Box").

is being modeled, the link bandwidths may be represented by global variables. The bandwidths can be changed during the course of a simulation run to analyze the effects of small changes in the vicinity of particular thresholds.

In addition, FAST offers automated monitor and control capabilities. Runtime statistics can be assigned to global variables, providing feedback within the simulation. For example, if the size of the queue exceeds a certain limit in a bank teller model, a new teller can be added automatically. In addition, the user can set the threshold during runtime, further tuning the model. Manual and automated sensitivity analysis provides a great deal of flexibility without the need to re-compile the program.

These capabilities allow the user to gain more insight into the actual behavior of the network being modeled. Based on the ability to easily and quickly modify model parameters and observe the resulting statistics, the user is encouraged to experiment with a large set of combinations of input parameters. The feedback which the user receives regarding the effects of different input values on model behavior is almost immediate.

3. THE FAST ENVIRONMENT

Reducing the major bottlenecks to simulation requires more than improving the editor, compiler, and runtime support mechanism. A modeler should be able to focus his or her attention on analyzing results, not coding. FAST provides a user-friendly environment that guides the beginner through program development, debugging, and runtime execution without constraining the experienced modeler. To implement this environment, FAST follows Zeigler's (1976) distinction between the simulation model and the experimental frame in which that model is executed.

The FAST interface consists of four levels of interaction, represented by four menus:

1. Max
2. Net Flow
3. Limits
4. State

The Max and Net Flow menus are used to create the simulation program, while the Limits and State menus provide the capabilities needed to run several experiments using a particular model (Figure 3). Each of these menus is discussed in detail below.

3.1. Max Menu

The maximum sizes permitted for various simulation data structures are set in the first menu presented to the user. The modeler either chooses a previously defined ".MAX" file or enters the Digital Equipment Corporation's Language Sensitive Editor (LSE) to create a new one. FAST provides a template that lists all pertinent data structures; the modeler merely estimates the upper limits (Figure 4). These limits include modeling constructs, such as the number of queues and resources, as well as programming structures, such as the size of the future events table. Memory allocation can be used most effectively by permitting the

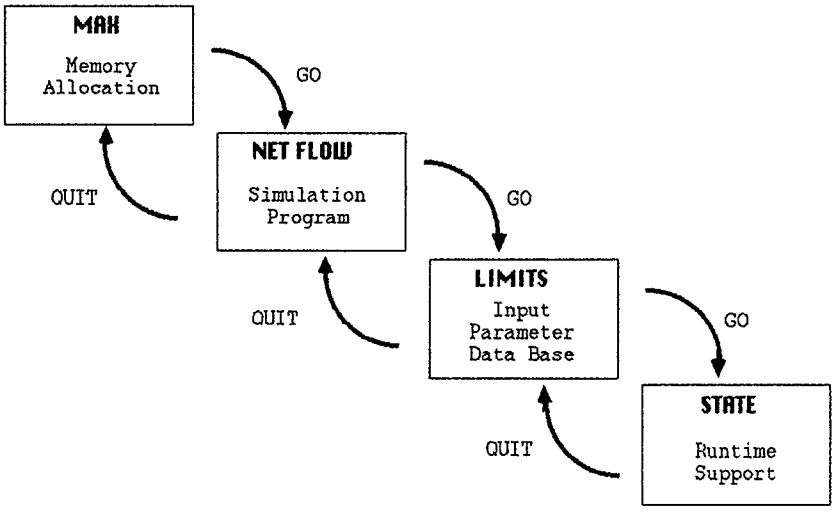

FIGURE 3. FAST LEVELS.
Interactive FAST guides the user through four levels of model creation and execution. The Max and Net Flow menus pertain to program development, while the Limits and State Menus control the simulation experiment.

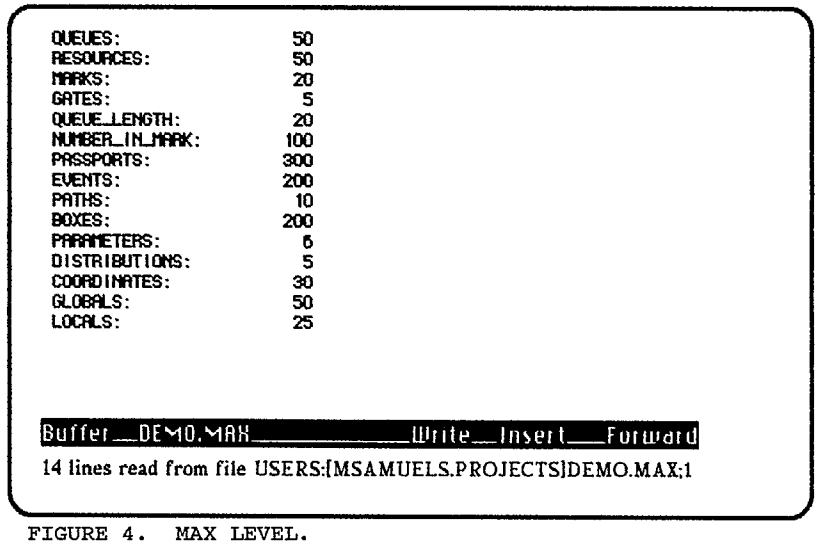

FIGURE 4. MAX LEVEL.
Memory allocation for a FAST simulation is set by user. A default file is automatically provided with all the FAST components that need to be estimated.

user to set storage limits. For example, a model of a data base system might require very large queues, whereas an automated teller machine would probably have a small finite queue with customer balking.

3.2. Net Flow Menu

Once memory has been allocated, the user enters the Net Flow Menu to create the simulation model. As mentioned, FAST takes advantage of the Language Sensitive Editor to insure proper programming constructs.

All FAST program consists of seven modeling components (see Figure 5):

1. Header
2. Queues and Resources
3. Marks
4. Gates
5. Distributions
6. Globals
7. Paths

The Header is a one-line descriptive title for the program that identifies different versions of a model.

Each simulated resource is associated with a queue of the same name. Only the names of different resource types must be declared here. If more than one server is available for each resource type, that number is set in the "Limits Menu" (see Section 3.3).

Marks represent user-defined statistical collection points. A simulated entity enters a mark, proceeds through a series of queues and resources, and then exits the mark. FAST automatically tracks the accumulated time spent between the "Mark" and "Endmark" statements, as well as the number of entities passing through this "checkpoint" during the course of the simulation. For example, a Mark structure would be used to record the elapsed time between a user request and the system response in a computer network.

Gates provide communication between different parts of a model. For example, a simulation of a computer center might restrict hours for a particular set of users. A Gate would be used to control requests from those users during the designated time periods.

Statistical distributions are listed next. FAST provides the most commonly used distributions, such as UNIFORM and EXPONENTIAL, as well as permitting the inclusion of empirical distributions. The user creates a ".DIST" file containing the cumulative distribution function from within the Net Flow menu or with any editor before entering FAST.

Globals refer to variables available to all parts of the model throughout the duration of a simulation run. In addition, globals are also accessible to the user at runtime. Not only are the values summarized on a "Globals" display page, but the user may change the value of any global variable at any time.

Paths are the key to a FAST simulation. Within each Path, the user designates the sequence of events that each entity encounters as it flows through the system. For example, one path could be used to model jobs flowing through a

```
DEMO SIMULATION - SINGLE SERVER QUEUE        -- 21 OCT 87

Queues and Resources:
  1: SINGLE_SERVER;

Marks:
  1: TIME_IN_SYSTEM;

Gates:

Distributions:
  0: FIX,     FIXED;
  1: UNIFORM, UNIFORM;
  2: EXP,     EXPONENTIAL;

Globals:
  1: INTERARRIVAL_TIME;                      -- minutes
  2: SERVICE_TIME;                           -- minutes

Path 1;
  Locals:
Buffer__BUFFER.FLOW_____________Write__Insert__Forward
LSE> 
54 lines read from file USERS:[MSAMUELS.PROJECTS]DEMO.FLOW;1
```

FIGURE 5. NET FLOW LEVEL.
Sample FAST program illustrates the major components of a simulation. Note that declarations are specified here, but values are set at the Limits Level.

machine shop, while a second path could be used to control machine availability. Or, two jobs with very different sequences of events could be modeled as two paths, keeping the code more readable for later modifications. This "process" approach is widely used in the simulation world and offers a clear, concise method of presenting models to the customer.

3.3. Limits Menu

The "Limits Menu" lets the user set the following parameters:

1. Simulation Duration
2. Resource Sizes
3. Queue Limits
4. Global Variables

Simulation duration refers to the length of a particular run, while the other limits assign values to simulation constructs defined in the Net Flow. The "Limits Menu" is similar to the "Max Menu" in that the user is provided with a template that lists all of the parameters requiring user input (Figure 6). Different ".LIM" files may be used with a given Net Flow; the values in each, however, must be within the restrictions set at the "Max Level".

3.4. State Menu

The "State Menu" is the best illustration of the inherent advantages of the FAST design. Four windows provide complete monitor and control capabilities (Figure 7):

1. Display
2. Simulation State
3. Error
4. Input

The Display Window not only includes statistical summary pages of modeling constructs (e.g., queues and resources), but also improves debugging capabilities by providing access to all queues, including the future events queue. The Simulation State Window shows the current status of the simulation (e.g., "Execution Suspended"), while the Error Window explains exception conditions when they are raised (e.g., "Error: Maximum queue size exceeded"). The User Input Window accepts input from the user at any time during the simulation.

4. THE ADA ADVANTAGE

As demonstrated above, FAST not only reduces the bottlenecks to simulation development by improving the component parts of the simulation package (i.e., editor, compiler, runtime support), but also by guiding the user through the steps necessary for building and using computer models. The key to reducing these simulation bottlenecks is to create a simulation tool with an underlying structure designed for flexibility. Design details have been described elsewhere (Spiegel, 1987) and need not be

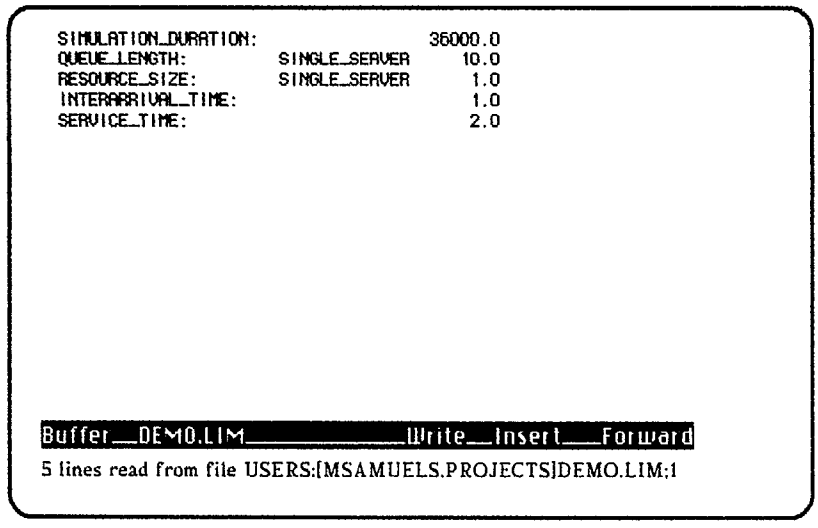

FIGURE 6. LIMITS LEVEL.
After compiling the Net Flow, FAST provides the user with names of model components that require initialization. Different limits files can be used with the same Net Flow.

repeated here. However, the requirements for rapid prototyping have been achieved because of several key features, unique to Ada, that were essential to the design and implementation of FAST. These are discussed below.

4.1. Object-Oriented Design

Every simulation language uses entities with assigned attributes, as well as a small subset of permissible operations on those entities (e.g., creating a particular instance of an event). Process-oriented simulation languages mimic tasking, and some languages even allow primitive forms of message passing. These are essentially object-oriented structures. Indeed, much of the development of simulation languages during the past twenty-five years can be viewed as an effort to add object-oriented constructs to procedural languages.

There are some continuing efforts to implement more extensive forms of these object-oriented structures to existing languages (West, 1985). However, the advantage of using Ada is that it provides an object-oriented design methodology, in addition to the necessary structures for implementation.

4.2. Simulation Error Handling

In many simulation languages, runtime errors cause the program to terminate prematurely, leaving the user with an unreadable postmortem dump to locate the source of the error. Simulation errors are often subtle, requiring hours or days to isolate and fix.

One of the advantages of using Ada to implement FAST is the extensive "exception handling" capability of the language. Simulation errors, such as the buffer overflow mentioned above, do not cause the simulation to terminate. Instead, the user is notified that processing has been suspended, and the user can search through the various runtime monitoring displays to uncover the source of the error. For example, a queue overflow can be resolved by changing the upper limit to the queue with the "Set_Queue_Size" command, by altering the number of resources available with the "Set_Resource_Size" command, or by changing the processing time associated with the resource using the "Set_Global" command. In all three cases, the simulation can continue from the point of suspended execution. In other words, a queue overflow is not an error condition, but an experimental result that teaches the modeler something about the behavior of the system.

4.3. Ada Tasking And Cpu Utilization

A major problem with executing a simulation is the heavy burden such models place upon the host computer processor. Interactive simulation has long been considered impossible for all but the simplest models because of the length of time needed to make each run.

Ada tasking provides the modeler with control of host resources. The runtime simulation is a separate Ada task from the monitor and control facilities so that CPU resources can be tailored to specific user needs. For example, if the modeler does not wish to examine statistics until execution is suspended after a certain

time period, the "Set_Speed" command can be used to allocate more host resources to the simulation task. On the other hand, debugging would require constant monitoring of simulation progress. The user would then divide host resources among the simulation, monitor, and control tasks to watch statistical updates more carefully. In most simulation languages, such monitor and control capabilities are limited to a small subset of input parameters visible just before the next event is selected from the future events queue.

5. EXTENDING THE LANGUAGE

Once a model has been developed and thoroughly debugged, the process becomes more global in nature. The object is to run simulations for a large number of experimental cases and then present the results in a format that managers can appreciate. This raises two points. The first is the design of the simulation experiment; the second is the capability to search and present simulation results.

Figure 8 presents an overall illustration of the automated global environment. This shows how the entire simulation project may be automated. The key element which enables this concept to be implemented is a simulation language that is driven by a script. This was an easy extension to FAST because it was originally intended to operate interactively. The implementation of batch mode maintains the capability to respond to commands. The only difference with the interactive mode is that batch FAST responds to commands provided by a script, as opposed to the keyboard. This permits further extensions to the language, as well as maintaining the separation of the simulation model and the experimental frame.

5.1. Experimental Data Base

Once a model has been debugged, the usual course of action is to go into "production" mode. This consists of running a large number of simulations in order to learn the effects which specific input variables, or combinations of input variables, have on the overall performance of the system being modeled. The most natural approach is to set up data files which contain the values to be varied for the different runs. A controller is then instantiated to oversee the multiple

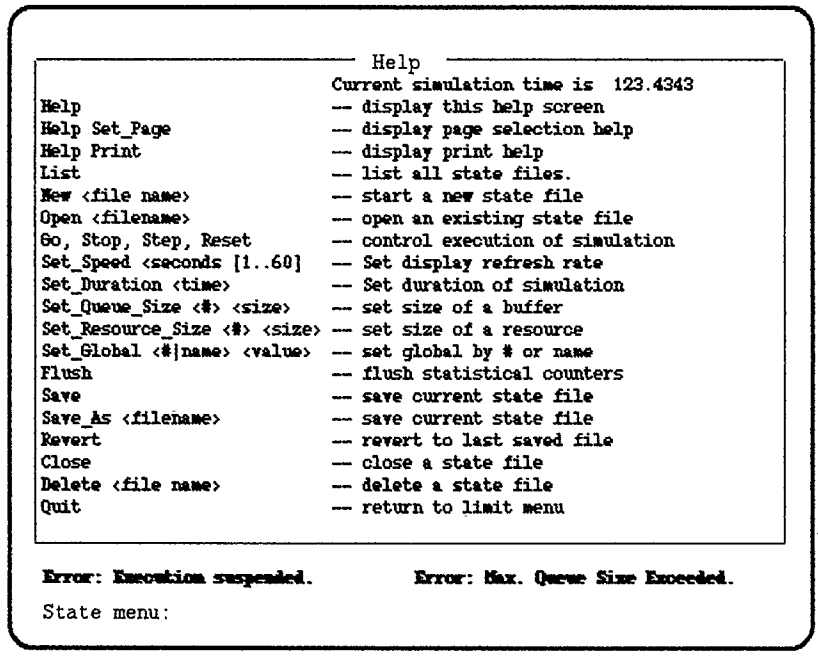

FIGURE 7. STATE LEVEL.
Four windows provide runtime support at the State Level. The Display Window currently contains the general Help Menu. Two other help windows, as well as more than ten different kinds of statistical summaries, provide invaluable aid during execution. The "Execution Suspended" message is in the System Status Window, while the Error Window lists the reason why the simulation stopped. The "State Menu:" prompt is the Input Window, always ready to accept requests from the modeler.

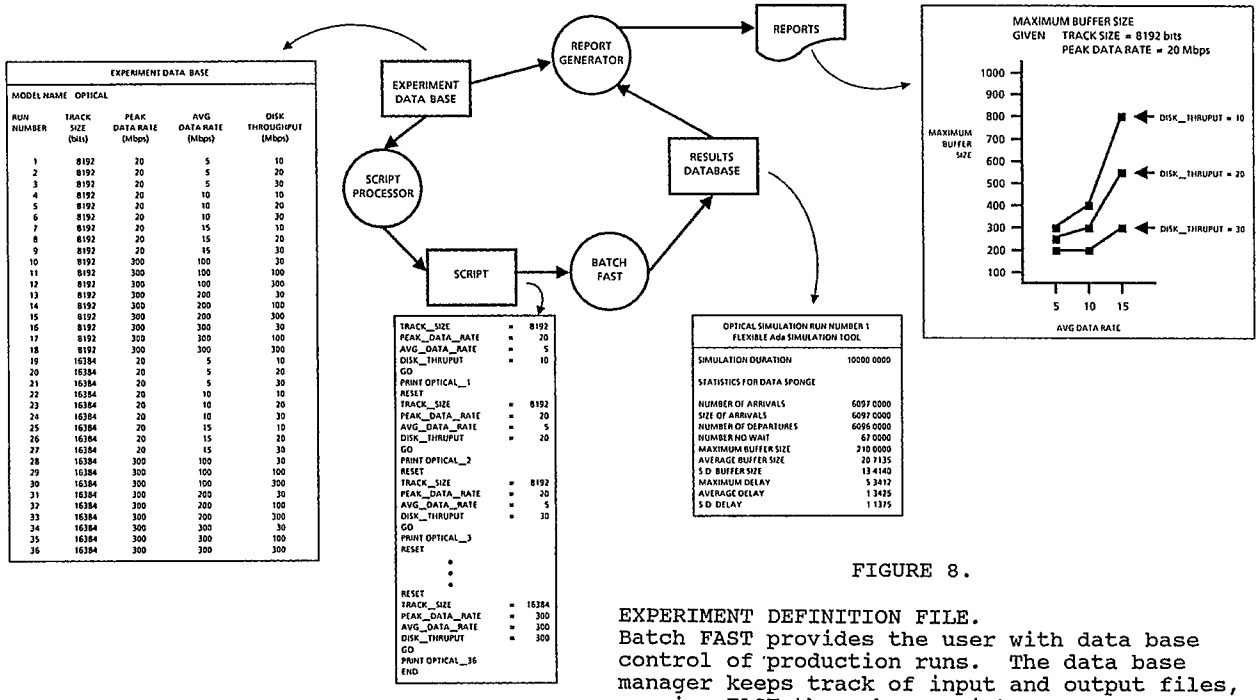

FIGURE 8.

EXPERIMENT DEFINITION FILE.
Batch FAST provides the user with data base control of production runs. The data base manager keeps track of input and output files, running FAST through a script.

executions of the simulation and keep track of the resulting output files.

The use of an experiment data base extends this concept beyond the single model approach. Simulation is often used to compare alternative architectures. To do this, separate models are built to represent each of the architectures being considered. The final analysis entails analyzing the results of different models in order to compare the performance of each design alternative. In addition to specifying input parameters and output requirements for each run, the data base also includes the particular architecture to be used for each run. The data base may then be coordinated with the output generator to automatically produce comparisons.

5.2. Running The Experiment

FAST was originally designed to operate interactively. As such, it provides an inherent capability to respond to command inputs. When FAST runs in batch mode, commands are provided through a script. The process of running an experiment is reduced to translating the experiment data base into a script to be processed by batch FAST. The resulting simulation outputs are stored in the results data base as specified by the experiment data base.

5.3. Report Generator

The significant advantage which is achieved by providing this automated environment is realized through the use of the automated report generator. The important design issue here is that the data base is used to decouple the report generation software from the simulation software. An extremely friendly generator could be provided which is able to produce any number of user specified reports. Through simple queries, the user can request graphical displays which provide information regarding relative performance of alternative architectures. The report generator is able to interpret the user queries, analyze the input data base to determine where the required information resides, extract this information from the results data base, and present the information in graphical format.

6. CONCLUSIONS

Simulation provides many features useful for prototyping real systems. However, simulations are software projects in their own right, often requiring as much time to develop as the systems they are to model. While recent additions to the commonly used simulation packages have greatly improved their capabilities, they still require too much time to be used for rapid prototyping. The Flexible Ada

Simulation Tool was designed as a solution to this problem. Through the use of the Ada methodology and programming language, a highly interactive simulation tool has been developed. The two major simulation bottlenecks, the interfaces between editing and compiling, as well as between compiling and executing, have been greatly reduced.

Flexibility in simulation has not only been demonstrated at the programming level, but in terms of the modeling environment, as well. As a consequence of the Ada design, an overall automated simulation project environment has been built which allows "global" flexibility. An experiment data base is coordinated with a results data base in support a "smart" graphics interface. The resulting system provides an extremely powerful simulation system.

REFERENCES

Booch, G. (1983). Software engineering with Ada. Benjamin/Cummings, Menlo Park, California.

Bratley, P., Fox, B.L. and Schrage, L.E. (1983). A Guide to Simulation. Springer-Verlag, New York.

Ghezzi, C. and Jazayeri, M. (1982). Programming Language Concepts. John Wiley and Sons, New York.

Henriksen, J.O. (1983). The integrated simulation environment (simulation software of the 1990s). Operations Research 31, 1053-1073.

Spiegel, J.R. (1987). Interactive discrete event simulation in Ada. Proceedings of the Joint Ada Conference: Fifth National Conference or Ada Technology and Washington Ada Symposium, March 16-19. pp. 121-125.

West, J. (1985). Object-Oriented Distributed Simulation. C.A.C.I., California.

Zeigler, B.P. (1976). Theory of Modeling and Simulation. John Wiley, New York.

AUTHOR'S BIOGRAPHIES

MICHAEL L. SAMUELS is a Systems Engineer at Ford Aerospace and Communications Corporation in College Park, Maryland. He received a B.A. in the College Scholar Program at Cornell University in 1977, focusing on models of cultural evolution and human ecological systems. He received an M.A. in Anthropology in 1981 and an M.S. in Systems Engineering in 1983 from the University of Arizona, where he used simulation techniques to investigate both ecological and engineering problems. He has applied modeling techniques to network management of the long-distance phone system, air-traffic control, and satellite communications networks. At Ford Aerospace, he is involved with the development of FAST and models of computer communications networks.

Michael L. Samuels
Ford Aerospace and Communications Corp.
4920 Niagara Road
College Park, MD 20740, U.S.A.
(301) 345-0250

JAMES R. SPIEGEL is a Systems Engineer at Ford Aerospace and Communications Corporation in College Park, Maryland. He received a B.S.E. and an M.S.E. in Systems Engineering from the University of Pennsylvania in 1979 and 1980. At Ford Aerospace, he has been involved in simulation of the Space Station Information System, as well as the development of future versions of FAST. Currently, he is developing a model of ground data handling system architectures for processing, buffering, and archiving high rate space station science data.

James R. Spiegel
Ford Aerospace and Communications Corp.
4920 Niagara Road
College Park, MD 20740, U.S.A.
(301) 345-0250