N89-16355

Simulation of the Space Station Information System in Ada*

James R. Spiegel
Ford Aerospace & Communications Corporation
College Park, Maryland

INTRODUCTION

The Flexible Ada Simulation Tool (FAST) is a discrete event simulation language which is written in Ada. FAST has been used to simulate a number of options for ground data distribution of Space Station payload data. The results of these analyses include on-board buffer requirements due to the TDRSS zone of exclusion, as well as bandwidth versus buffer and bandwidth versus delay tradeoffs within the ground system.

The fact that the Ada language is used for implementation has allowed a number of useful interactive features to be built into FAST and has facilitated quick enhancement of its capabilities to support new modeling requirements. The use of tasks and packages has enabled the development of an interactive environment which allows the user to monitor and control the simulation. As a simulation is executing, a concurrent display task is updating pre-defined pages which contain simulation output statistics. A user command interface allows the user to pick from a number of display pages. This command interface also allows the user to interactively modify network parameters (e.g. number of servers or link bandwidth).

This paper discusses general simulation concepts, and then how these concepts were implemented in FAST. The FAST design is discussed, and it is pointed out how the use of the ADA language enabled the development of some significant advantages over classical FORTRAN based simulation languages. The advantages discussed are in the areas of efficiency, ease of debugging, and ease of integrating user code. The specific Ada language features which enable these advanced are discussed.

SIMULATION CONCEPTS

FAST is a general purpose discrete event simulation tool. Currently, there are a number of simulation languages that are recognized in the field of discrete event simulation. The list includes SLAM, GPSS, SIMSCRIPT, and others. The key feature that defines a "discrete event" simulation is that the state of the modelled system changes at discrete points in time. The simulation "language" automatically performs the task of keeping records of what events are planned to occur, and when they will

*Ada is a registered trademark of the U.S. Government (Ada Joint Programming Office)

occur. The language also performs the task of maintaining statistics that describe the performance of the network elements. The job of the user of a simulation language is to model a given system within the constraints of the particular language being used.

The first step in the process is for the user make a abstraction of the system. Essentially, this means applying the terms of the simulation language such as "resource", "queue", and "traffic" to the user's particular problem. Figure 1 provides a table of different types of systems that may be modelled, and the associated meanings of each of the model elements.

One type of "network" which FAST has been used to model is the SSIS. In this case, the "traffic" entities are data packets, and the "servers" or "resources" are the communications links. Simulations were performed to answer such questions as:

How much bandwidth is needed ?

How long will data be delayed ?

What percentage of the time is the link busy?

For this example, the answer to the second and third questions are dependent on the first. The average wait time per packet is dependent on the link bandwith. The bandwidth is thus a "network parameter", while the delay times (queue statistics) and the link ("resource") utilization describe the system performance. The objective of a simulation activity is to predict the system performance as a function of the network parameters. This is usually done by performing a number of simulation "runs", while varying the network parameters. The result of each run is usually viewed as a point on a curve, and this curve describes the system performance as a function of input parameters.

The methodology used to implement an event-driven simulation is based on the concept of a future events queue. An event may be defined as any action or condition that changes the state of the system. Examples are when a data packet is generated, or when a transmission has been completed. The future events queue keeps a record of all of the events that may be planned. For example, when the transmission of a packet is initiated the time at which the transmission will be complete is calculated. This event is placed on the future events queue.

Each time an event occurs, a procedure is called that implements the logic associated with that event. This logic

SYSTEM	*TRAFFIC*	*RESOURCE*
SPACE STATION INFORMATION SYSTEM	DATA , COMMANDS	SPACE-GROUND LINKS GROUND-GROUND LINKS PROCESSORS
COMPUTER	DATA	PROCESSORS TASKS BUSSES DISKS
TELEPHONE	CALLS	CIRCUITS SWITCHES
MANUFACTURING	WIDGETS	WELDERS PAINTERS
BANK	CUSTOMER	TELLER
LOCAL AREA NETWORK	MESSAGES	BUSSES

Figure 1 - General Purpose Simulation Concepts

consists of decision making (is a link available?), updating the state of the system (the link is now busy), and performing calculations required to maintain statistics. When the processing for a given event is complete, then the future events queue is used to determine the next event.

One of the major problems in the area of simulation is efficiency. The process of discrete event simulation is inherently a Monte-Carlo process. This means that the input traffic is described by a statistical model. The simulation is thus performed using random inputs, and the statistics which describe the network performance are expected to converge in time. The number of events which need to be processed in order to achieve statistical convergence is both very large, and difficult to predict. The procedure usually adopted is to pick a safe duration, and to use this for all runs. This often results in two troublesome phenomena. The first is that more computer time is used that is actually necessary for a given run. The second is that the scope of the simulation activity is usually limited by the computing resource.

Another limitation of general purpose modelling languages is that they are usually not sufficient to model the complex interactions of a real world systems. Many simulation languages overcome this by allowing the user to write his own procedures. Mechanisms are provided for the user to write his own code (usually in FORTRAN), and integrate user written procedures in the model. The support available for this type of activity varies among languages, but in almost no cases can the support be considered "friendly". In most cases, the user is constrained to the use of a number of cryptic conventions in order to integrate his code. This process is both time-consuming and fraught with hazards. The bottom line is that one has to be a simulation "expert" in order to undertake such a task.

One final source of many headaches for users of simulation languages is the area of debugging. This includes both debugging of user written simulation routines (discussed in previous paragraph), and the debugging of models which do not work. Again, various languages provide various levels of support for this activity. As a minimum, most languages have the capability to list the names of what events occured and at what time. This results in a large listing which the user must search through in order to begin to understand where a problem is occuring. Once this information is found, it is sometimes useful, but oftentimes it does not shed enough light to solve the problem. When this happens the user is left little option other than staring long and hard at his input model, scratching his head, and trying to determine why he got the unexpected output. He may change one variable, rerun the model, and see what effect it had. When this fails to shed light, he will change others as deemed appropriate. This can be a very time consuming activity. Frustrated modellers have even been known to blame hardware.

FAST CONCEPTS

Three areas have just been described in which improvement is clearly welcome. These are :

- o EFFICIENCY
- o DEBUGGING
- o EASE OF INTEGRATING USER CODE

FAST has been designed with the objective of alleviating many of the obstacles which are encountered in these areas. In order to understand how these areas are addressed, it is necessary to first gain an appreciation for the overall FAST environment. This section provides a general description of the FAST environment, and then discusses the advances which have been recognized in these three areas.

FAST provides an unusually friendly environment in which to perform simulations. Figure 2 illustrates this "environment". FAST is designed to run interactively from a terminal. When FAST is running, most of the screen is dedicated to the display window. The user may specify which page is to be displayed by entering a "SET-PAGE" command in the input window. Figures 3 and 4 show the menus of user commands and pre-defined display pages which the user has to choose from. Figure 5 provides an example of one of these display pages. The other two windows are the error window, and the simulation state window. The error window is used when there is a syntax error in the user input, or when there is an error within the simulation run. The simulation state window displays whether a simulation is running, stepping, or suspended.

In a typical use of FAST, the user runs a simulation and monitors a statistic of interest. When the statistic has converged, the user changes the network parameters, and a new simulation "run" is started. This environment presents a number of advantages, the most important of which is that the user is able to observe the statistic as it is updated in accord with the progress of the system. The period of time required to obtained confidence in the results is significantly reduced.

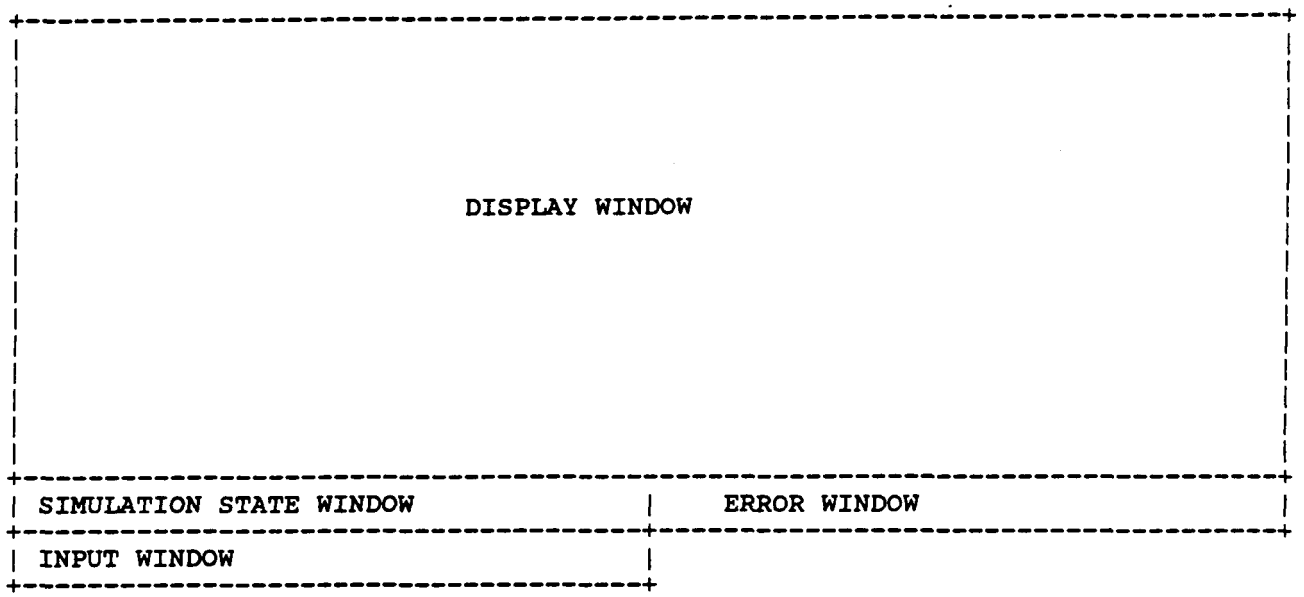

Figure 2 - The FAST Environment

```
State menu commands:

Help                                    -- Display this help screen
Help  <Set_Page>                        -- Display page selection help
List                                    -- List all state files
New  [<filename>]                       -- Start a new state file
Open [<filename>]                       -- Open a state file
Set_Speed <number of seconds from 1 to 60>    -- Set refresh rate of display
Set_Duration      <simulation time>           -- Set duration of simulation
Set_Queue_Size    <queue number>    <size>    -- Set size of one queue
Set_Resource_Size <resource number> <size>    -- Set size of one resource
Flush                                   -- Flushes Statistics
SAVE                                    -- Saves the current state file
Close                                   -- Close a state file
Print                                   -- Print Simulation Results
Quit                                    -- Return to limit menu
Execution Suspended.
Input :
```

Figure 3 - Input Command Menu

```
Page Selection State menu commands:

Set_Page Queue_Resource_Summary          -- Display queue / resource stats
Set_Page Mark_Summary                    -- Display mark statistics
Set_Page Limit_Summary                   -- Display limit statistics
Set_Page Queue_Resource <queue number>   -- Display statistics on one queue
Set_Page Queue          <queue number>   -- Display a queue
Set_Page Future_Events_Queue [<number>]  -- Display Future Events Queue
Set_Page Mark           <mark number>    -- Display statistics on one mark
Set_Page Passport_Summary  [<number>]    -- Display status of all passports
Set_Page Active Passports  [<number>]    -- Display status of active passports

Input :
```

Figure 4 - Display Page Menu

```
Queue and resource summary                Current simulation time is 10000.0000

Queue  Arrivals  Avg Length  Avg Wait  Avg Resource Usage
-----  --------  ----------  --------  ------------------
    1      8153      1.0111    1.2401              0.4144
    2      8251      0.0529    0.0642              0.4282
    3      8153      4.3402    5.3238              2.1731
    4       102      1.3550  133.8373              1.0000
    5       600     27.1731  465.1406              1.0000

Execution Suspended
Input :
```

Figure 5 - Sample Queue-Resource-Summary Page

The user may control the execution of the simulation through the use of "START", "STEP", "STOP", and "RESTART" commands. In addition, he may alter network parameters using the "SET-QUEUE-SIZE" or "SET-RESOURCE" command. The result of these ccapabilities is that an environment is provided in which the user may monitor and control the simulation process.

EFFICIENCY

The ability to provide the capability to build and monitor display pages was facilitated by the use of Ada tasking. The FAST system consists of a number of tasks. One of these is the simulation task, which performs the actual network simulation. In addition, there are tasks for displaying pages, as well as tasks to interact with the user.

There is no way of getting around the fact that Monte-Carlo simulation takes a long time in order to achieve statistical convergence. What FAST does provide, however, is a mechanism to monitor the statistics in question. The user may monitor a statistic during a simulation run, and when the statistic has stabilized to the user's satisfaction, the run may be stopped. This provides two advantages. The first is that a confidence range may be established. The second is that the user does not have to guess how long to run the simulation, thus saving a lot of personal and CPU time.

DEBUGGING TOOLS

Clearly, for an event driven simulation, the future events queue is vital to the inner workings of the simulation. The ability to view the future events queue on an event by event basis is extremely valuable for debugging purposes. One of the major advantages of FAST is that it does allow visibility into the "internal" structures of the simulation. These include both a "Future-Events-Queue" page, as well as an "Active-Passports" page. (A passport is a record that is used to keep track of the traffic entities as they flow through the system). The Future-Events-Queue lists which passports are scheduled to be activated, and when. The Active-Passports page describes where within the network each passport resides, as well as additional information about the passport. The combination of these two pages provides significant detail regarding the state of the system. FAST also includes a feature called step mode, which allows the user to instruct the model to process only one event at a time. Using the pages in conjunction with the step mode the user is able to observe the very fine details of the system, and can do so at whatever level is deemed appropriate for determining

exactly how the simulation is progressing.

In addition to the features that have been described, the process of debugging is reinforced by an error management philosophy that takes advantage of Ada's exception handling. If, in the process of a simulation, a logical simulation error is encountered (such as a queue overflow), this error is managed as an exception. The simulation is suspended, and an error message is displayed in the error window that describes the error. At this point, the user is able to investigate why this error has occured. All of the simulation structures are still in tact, so the user may use the display capability to observe any of the pre-defined pages.

One proven debugging technique is to use the "SET-DURATION" command to a time just prior to the simulation error. A "RESTART" command will then cause the simulation to run to a point just before the error occurs. The user may now proceed using the STEP command to determine exactly when, where, and why the error occured. Clearly, such a capability is invaluable in the debugging process.

EASE OF INTEGRATING USER CODE

FAST has been designed in such a way that makes adding user modules safe, efficient, and easy. This is due to the fact that an object oriented design has been implemented which not only protects the system from the user, but also provides maximum support for the user.

As an example, there is a "QUEUE" package which contains the data structures which are used to model the queues, and all of the procedures and functions which operate on queues(e.g. REQUEST, RELEASE). Within the queue package, all of the logic which is needed to model the queue (First-In-First-Out) is coded. All additional effort which is required in order to implement these functions is provided by support packages. All of the queue length statistics (average, standard deviation, maximum, minimum) are maintained by a statistics package. Within the queue package, whenever the queue length is altered, a message is sent to the "STAT" package. Similarly, communications between instances of queue and passport structures is through a message oriented protocol similar to Smalltalk in nature. Real-time displays are implemented through messages to a window manager (also implemented in Ada).

All of these support packages which are currently used by the existing FAST packages are available for user written

packages. This means that the inclusion of user packages is both safe and efficient. In addition, the debugging capabilities significantly reduce the time necessary to test and integrate large models. Finally, the user packages are written in Ada, and are thus blessed with the inherent advantages therein.

The use of object oriented design has already provided significant efficiencies in the development of the FAST system. In addition to all of the classical arguments espoused by the proponents of object oriented design, the methodology lends itself particularly well to the implementation of a simulation language. Specifically, objects that are built to model elements are limited in scope and complexity to the problem of modelling the logic of that element.

CONCLUSION

FAST uses the capbilities of the Ada language (packages, tasking, and exception handling) in order to enhance a classical simulation tool by providing an interactive, friendly simulation environment. The result is a tool which is easy for a beginner to use, and significantly increase the productivity of an experienced network simulation specialist.