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

A "DISPOSABLE" GRAPHICAL EVENT SYNTHESIZER FOR TEACHING SIMULATION MODEL BUILDING

Lee Schruben

School of O.R.I.E.
Cornell University
Ithaca, NY 14853

ABSTRACT

This article describes an easy to use graphical simulator specifically designed for teaching the fundamental concepts of discrete event simulation modeling. The simulator permits students to have the satisfaction of building working models early in a course without having to learn a specific commercial language. The simulator is "disposable" in the sense that students can replace the various components of the simulator with their own routines as they progress through the course. It is thus possible for the students to customize their own personal graphically oriented modeling language. Models built on this simulator are also easily implemented in higher level simulation languages making the simulator a useful modeling aid even for simulation courses that are built around a specific language.

1. INTRODUCTION:

We who teach discrete event simulation must make the important decision of whether to structure our course around a specific simulation language or to design a general course that is language independent. The main advantage of a language oriented course is that students have the satisfaction of building working simulation models relatively early in the course. They also gain experience and develop a loyalty to (or distaste for) a particular commercial language. Proficiency in a simulation language may also improve a student's short-term job prospects. The merits of having students develop language loyalty in a university course is a continuing subject of dispute among simulation educators and language vendors [1].

Unfortunately, students who approach simulation through a specific language might not learn some of the fundamental simulation techniques and concepts. To illustrate: the main distinctions between simulations and other types of computer programs are the modeling of time and randomness. A course that is tied to a language may not emphasize methods for pseudo-random number generation or mechanisms for managing the simulated clock. Since most commercial languages insulate the user from these tasks, students are not likely to gain a real understanding or control of these basic ideas. The attitude too often is, "why learn it if the computer can do it?" Furthermore, if a specific language is used then modeling approaches that do not fit comfortably within the language viewpoint may not be introduced to the class.

I have taught (and taken) both language oriented and language independent simulation courses. The language oriented courses are much easier to teach and more fun for the students. Teaching a rigorous course on simulation modeling is hard work and the students are easily discouraged during the early part of the course as they learn fundamentals. It is not until relatively late in the course, when they start assembling the concepts and techniques into actual simulation models do they start to enjoy the course.

The graphical simulator described here is designed to make learning (and teaching!) discrete event simulation enjoyable without adopting a particular language. Students can build working simulation models of virtually anything (see the appendix) in the first week, but they do it at a fundamental level. There is a heavy reliance on personal computer graphics.

The simulator is "disposable" in the sense that students can replace the various components of the simulator with their own routines as they progress through the course. When students study such topics as pseudo-random number generation or event list management they can write their own algorithms (in FORTRAN, C, PASCAL, or ASSEMBLY...) to replace the algorithms used by the simulator. It is thus possible for the students to build their own personally customized graphically oriented modeling language as part of a course.

Using the simulator it has been possible to design a simulation course to be both fundamental and fun. Without being tied to a particular language, students can experience simulation modeling early the course. They can better appreciate how the parts fit together as they learn simulation fundamentals. This permits a reversal of the usual progression of a typical non-language oriented simulation course where students would first learn about modeling components in isolation and then later assemble these components into models.

The simulator is a useful modeling aid even for models that are eventually going to be implemented in a high level simulation language. The simulator can be used in simulation courses that are built around a specific language.

2. THE SIMULATOR:

The current version of the simulator requires an IBM PC compatible computer with at least 120K of free memory and a graphics board. The simulator is written in C but is self-contained and does not need a compiler or special graphics software. In this section we present an brief overview of the simulator. All commands are selected from menu trees. Only the top level menu is described here.

A more detailed tutorial is handed out to the simulation class.

2.1. Events and their Relationships:

The elements of a simulation model are the state variables, the events that change the state variables, and the relationships between the events. The simulator represents the simulation model as a graph [2]. This graph is a structure of the objects in a discrete event system that facilitates the development of a correct simulation model. Events are represented on the graph as vertices (balls or nodes); each is associated with a set of changes to the state variables. Each vertex of the graph is a pointer to string of system state change expressions that occur when a particular event happens.

For example, in a single server queue one of the state variables might be the number of customers in the system; assume that the user calls this variable "number". A "customer_arrival" event will cause this state variable to be increased by one; hence, the vertex points to the string "number = number + 1". The window for such an event might read as follows;

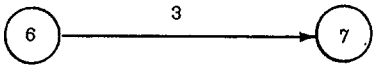
Event Vertex #1

Description:	A new customer arrives
Event Name:	customer_arrival
State Changes:	(number = number + 1)
Parameter variables:	queue
Priority:	6

This event would have a parameter of "queue" telling perhaps at which queue in the system the customer arrives. The event has an execution priority of 6 to break ties in simultaneously scheduled events.

Between the vertices are directed edges (or arrows) that point to the logical and temporal relationships between events. Basically, the edges define under what conditions and after how much of a time delay one event will cause another to occur. Associated with each edge is a set of conditions that must be true in order for the originating event of the edge (tail of the arrow) to schedule or cancel the termination event (the head of the arrow). Also associated with each edge is a delay time that tells how long until a scheduled event occurs. Finally, there are a set of attribute values that can be passed from the scheduling event to the scheduled event. All event vertices and edges are numbered on the graph.

To illustrate: suppose the following edge is displayed on the screen as part of a simulation graph,



In addition, assume the user calls up the window for edge 3 and is given the following;

Edge #3

Scheduling/Cancelling:	Scheduling
Originating Event:	6 "end_service"
Destination Event:	7 "queue_up"
Condition:	(QUEUE > 2)
Delay Time:	(time.fnt)
Passed Parameters:	8,MACHINE+1

This would indicate that each time event 6 (here called an "end_service" event) is executed: if QUEUE is greater than 2 then event 7 (here called a "queue_up" event) will be scheduled to occur time.fnt time units later using parameter values of 8 (maybe a part type) and MACHINE+1 (maybe the next machine).

The delay time, time.fnt, may be a function (say a theoretical random variate), or a value in a data file (for trace driven models), or value from a table (say of an empirical distribution). It is not necessary to change the model in any way to accommodate different types of input streams. The conditions, delay times, and the attribute lists can contain any valid expressions.

There can be multiple edges between any pair of event vertices in the model; these edges can point in either direction.

2.2. Top Menu:

The top level menu of the simulator has thirteen command selections; here we present only a brief description of each logically related group of commands. On computers without a mouse, the user presses a key of a letter that is capitalized in each command.

CREATE, EDIT, and DELETE: These commands allow the creation, editing, and deletion of state variables or arrays, event vertices, and event relationship edges in the simulation graph. Each command will bring up an appropriate menu.

READ, FILE, and APPEND: These commands allow the students to read a previously saved model, file the current model for future recall, or to append another model onto the current working model. The append command permits the students (or teams of students) to create separate parts of a simulation model and then easily connect them together into a larger simulation.

MOVE and ZOOM; These commands allow the graphs to be moved and viewed from various windows.

PRINT; This gives hard copy model documentation for the students to hand in with their homework reports. Their are several simple rules that indicate when two event graph representations of a simulation are equivalent [2]. This greatly helps in grading homework; instructors and teaching assistants never look at a single line of computer code! (I think this is the main benefit to the instructors in using this system... I hate trying to find out what is wrong with a students code)

RUN: This evokes a series of run control selections for the student. They can control the initial conditions (state, event list, random

number stream,etc.). They can control the termination conditions (time, event count, or state dependent). They can determine which variable values to monitor and the disk file that will contain the multiple output series. Finally, they can select one of three run modes.

The three run modes are "high-speed" which simply gives the output file on a disk, "interactive" which allows the user to change the states and event lists while running in single event steps. Finally, there is the "default" run mode. In the default run mode graphics are used extensively to illustrate the dynamics of running the simulation model. Edges between events are highlighted (or change color) when their conditions are true, event vertices "glow" when they are scheduled and flash when they are executed. On a color monitor with EGA or better graphics, this gives the effect of a liquid flowing through a network from event vertices that are executed to the event vertices that they schedule.

DOS and EXIT: The DOS command allows the user to toggle back and forth from the simulator to the operating system (say to check directories or edit an output file). An interesting feature that I do not tell the students about is that one may have several layers of different simulation models called up simultaneously on the same PC... it is a credit to the operating system that this does not seem to confuse the computer...The exit command allows one to leave the simulator and return the PC to the state (modes, paths, etc.) that it was in before the simulator was invoked.

2.3. The Output Files:

The output files contain an entry for each event that is executed. The event name, a count of the number of times the event has been executed, the clock time and the values of the user selected trace variables. The student can then call data analysis and plotting routines of their choice with this file as input. This is where the students have been quite creative and makes reading their reports very interesting. Under development is a set of default output analysis and plotting commands but I am not sure that this is really a good idea. Maybe giving them the raw data file is really the most instructive?

3.0. AN EXAMPLE:

The following is an example of a multiple server queueing system simulation as developed on the graph simulator. The events are as follows:

1. "START_RUN" Here we initialize the state variables and start the simulation run. The state variables Q (number of customers in line) and FREE (number of free servers) are set through the parameters of this event. Values for these variables are requested from the user during run initiation.

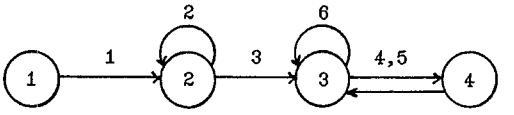
2. "ARRIVAL" This event is the arrival of another customer to the system. The time between scheduled arrivals is given in the function (or . data file) arrive.tim.

3. "BEGIN" This is the event where the server starts service on a customer.

4. "END" This is the event where the server finishes service on a customer. The service times are in the function (or data file) named service.tim.

3.1. The Model:

The event graph displayed on the screen is as follows:



Note that the bold face edge between event #3 (start) and event #4 (end) is a multiple edge (edges 4 and 5 are between these two events). The elements of the graph are described in the windows that follow.

To save space only an abbreviated form of the windows are given below.

STATE VARIABLE WINDOW

Variable Name	Description
Q	Number of customers waiting
FREE	Current number of idle servers

EVENT VERTEX WINDOWS

Event vertex #1
```
Name:          INITIAL
Description:   Initialize run with FREE and Q
State Changes:
Parameters:    FREE,Q
Priority:      0
```

Event vertex #2
```
Name:          ARRIVAL
Description:   Customer arrival to the system
State Changes: Q = (Q + 1)
Parameters:
Priority:      1
```

Event vertex #3
```
Name:          START
Description    Start service on a customer
State Changes: FREE = (FREE-1),Q=(Q-1)
Parameters:
Priority:      0
```

Event vertex #4
```
Name:          END
Description    End service, freeing a server
State Changes: FREE = (FREE + 1)
Parameters:
Priority:      0
```

EVENT RELATIONSHIP WINDOWS

Edge #1
```
Scheduling     yes
Origin:        1
Destination:   2
Condition:     (1) - always true
Delay:         (0)
Attributes:
```

```
    Edge #2
Scheduling      yes
Origin:         2
Destination:    2
Condition:      (1)
Delay:          ({arrival.tim})
Attributes:

    Edge #3
Scheduling      yes
Origin:         2
Destination:    3
Condition:      (FREE > 0)
Delay:          (0)
Attributes:

    Edge #4
Scheduling      yes
Origin:         3
Destination:    4
Condition:      (1)
Delay:          ({service.tim})
Attributes:

    Edge #5
Scheduling      yes
Origin:         4
Destination:    3
Condition:      (Q > 0)
Delay:          (0)
Attributes:

    Edge #6
Scheduling      yes
Origin:         3
Destination:    3
Condition:      ((Q>0) and (FREE>0))
Delay:          (0)
Attributes:
```

3.2. The Output:

The above simulation was run for 3 servers (initially FREE = 3) starting with an initial queue size of 10 customers (initially Q = 10). The first few lines of the output file are as follows.

OUTPUT FROM RUN WITH Q=10.0, FREE=3.0

Time	Event	Count	Q	FREE
0.000000	START_RUN	1	10.000	3.000
0.000000	ARRIVAL	1	11.000	3.000
0.000000	START	1	10.000	2.000
0.000000	START	2	9.000	1.000
0.000000	START	3	8.000	0.000
0.077096	ARRIVAL	2	9.000	0.000
0.217215	END	1	9.000	1.000
0.217215	START	4	8.000	0.000
0.827738	END	2	8.000	1.000
0.827738	START	5	7.000	0.000
0.967278	ARRIVAL	3	8.000	0.000
1.012833	END	3	8.000	1.000
1.012833	START	6	7.000	0.000
1.412778	END	4	7.000	1.000
1.412778	START	7	6.000	0.000
1.737114	END	5	6.000	1.000
1.737114	START	8	5.000	0.000
2.246239	END	6	5.000	1.000
2.246239	START	9	4.000	0.000
2.431257	END	7	4.000	1.000
2.431257	START	10	3.000	0.000
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

4. SOFTWARE AVAILABILITY AND CURRENT DEVELOPMENTS:

The latest version of the software described here is available at cost to instructors of university simulation courses.

5. APPENDIX:

The claim that anything that can be simulated on a computer can be simulated using the graph simulator described here is based on the Church-Turing thesis [3] and the fact that a Turing machine can be simulated using an abstract form of the simulation graph presented here and in [2]. A rigorous argument will not be presented. We will need the abstraction of a semi-infinite "tape" or array just as in the description of M.

The Turing machine described on page 148 of [3] is known to be able to simulate random access memory. That machine is denoted as $M = (Q,\Sigma,\Gamma,\delta,q_0,B,F)$. To simulate such a machine with a simulation graph in the most straightforward manner let everything be the same as in M and define two vertices to represent the left move and the right move. These vertices can schedule themselves and each other and have identical state transition functions given by δ.

While the ability to simulate a Turing machine may seem trivial, it does have some important implications; eg. cancelling edges are never necessary and are merely a convenience, this simple description of a simulation with the above graph is all that is necessary.

Acknowledgements: Many people have helped in the development of the graphical simulator described here. In particular the work of Christian Outzen of DEC is appreciated. Comments freely offered by David Briskman, Arnold Buss, Chris Read, Steve Roberts, Paul Sanchez, and Robert Sargent have also stimulated the author to further develop the ideas presented here. David Tate of Cornell suggested simulating a Turing machine with the graph simulator.

REFERENCES

[1] Various Past Panels on Simulation Education at the Winter Simulation Conferences.

[2] Schruben, L. "Analysis of Simulation Event Graphs", Comm.A.C.M. Vol. 29.11.

[3] Hopcroft, J. and J. Ullman, Introduction to Automata Theory, Languages, and Computation Addison-Wesley, 1979 Chpt. 7.

AUTHOR'S BIOGRAPHY

LEE SCHRUBEN is on the faculty of the School of Operations Research and Industrial Engineering at Cornell University. He received his undergraduate degree in engineering from Cornell University and a Masters degree from the University of North Carolina. His Ph.D. is from Yale University. Before going to graduate school he was

a manufacturing systems engineer with the Emerson Electric Co. in St. Louis, Mo. His research interests are in the statistical design and analysis of large scale simulation experiments. His consulting activities have been primarily focused in the area of manufacturing systems simulation. He is currently the chairman of the TIMS College on Simulation and serves on several the editorial boards for several journals.

Prof. Lee Schruben
School of O.R.&I.E.
Cornell University
Ithaca, NY 14853
ph: (607) 255-9133