Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Teaching Simulation with Σ

Professor Lee Schruben
School of OR&IE
Cornell University
Ithaca, NY 14853

and

Mr. David Briskman
School of OR&IE
Cornell University
Ithaca, NY 14853

ABSTRACT

Σ (pronounced SIGMA denoting Simulation Graphical Modeling and Analysis) is an interactive graphics approach to building, testing, and experimenting with discrete event simulation models on personal computers. Σ is written in C but is self–contained and does not need a compiler or special graphics software. Σ is an extension of the simulation teaching system report in [2].

The version of Σ described here requires an IBM PC compatible computer (AT preferred) with at least 420K of free memory, a floppy disk drive, an EGA or equivalent monitor with the corresponding graphics card, and a mouse.

1. CONCEPTS AND COMMANDS IN Σ

In [2] the basic concepts of Σ are presented with an overview of the command structure for the PC implementation. All commands are selected from the main menu. A more detailed description of the commands is presented here.

Σ is based on an extension of the event graph representation of a discrete event system [1]. The elements of a simulation model are the state variables, the events that change the state variables, and the relationships between the events. An event 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 (nodes); each is associated with a set of state changes.

Logical and temporal relationships between events are represented in an event graph as directed edges (arrows) between pairs of vertices. The edges define under what conditions and after how much of a time delay one event will schedule or cancel another event.

There can be multiple edges between any pair of event vertices in the model; these edges can point in either direction. Multiple edges are depicted with the prescripts and postscripts telling the number of sub–edges contained in that edge. The postscript, (the smaller number to the right of the edge number) tells how many edges are going in the direction of the arrow (tail to head). The prescript, gives the number of edges going in the direction opposite of the arrow. If only the larger (graphics) edge number appears, there is only one edge there. The displayed edge will be referred to as the *graphics edge.*

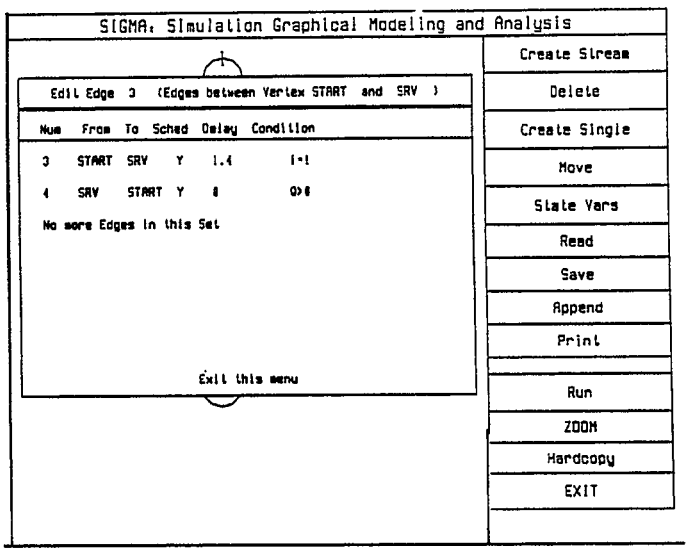

Figure 1. A Popup Menu from a Multiple Edge

1.1. The Main Menu Options

All options on the main menu are functional. When an option (or button) is "on", the mode or system status is defined by the function represented by the button. When no buttons are "on", Σ is in *Edit Mode.* Edit Mode is the only mode by which data may be entered.

Create Stream, Create Single, State Vars and Delete: These commands allow the creation and deletion of state variables or arrays, event vertices, and event relationship edges in the simulation graph.

Read, Save, and Append: These commands allow the users to read a previously saved model, to save the current model for future recall, or to append another model onto the current working model. The append command permits the user (or teams of users) to create separate parts of a simulation model and then connect them together into a larger simulation.

Move and Zoom: These commands allow the graphs to be moved and viewed from various windows.

Dos, Clear 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). The Clear command will restart Σ from scratch. 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.

Run: This evokes a sequence of run control choices. The user can select the initial conditions (state variable values, random number seed, etc.), control the run termination conditions (time or event count), and determine which variable values will be recorded during the run. The user is also asked the name of the disk file where the output is to be recorded. Finally, one can select one of three run modes.

The three run modes are **HIGH SPEED** which simply gives the output file on a disk, **SINGLE STEP** which allows the user to slowly step through the execution of each event viewing state changes, output, and the events list. If neither **SINGLE STEP** or **HIGH SPEED** run modes are selected the run mode will default to **Graphics** mode. The **Graphics** run mode uses color extensively to illustrate the dynamics of running the simulation model. Edges between events turn yellow when their conditions are tested as being true, event vertices shade with a yellow pattern as they are scheduled and when they are executed, the vertices turn solid yellow. (Note: While in Graphics mode, you may halt execution by hitting control–q.)

1.2. The Output Files

The output file contains an entry for each event that is executed during the run. Each entry gives the clock time, the event name, a count of the number of times the event has been executed, and the values of the user selected trace variables.

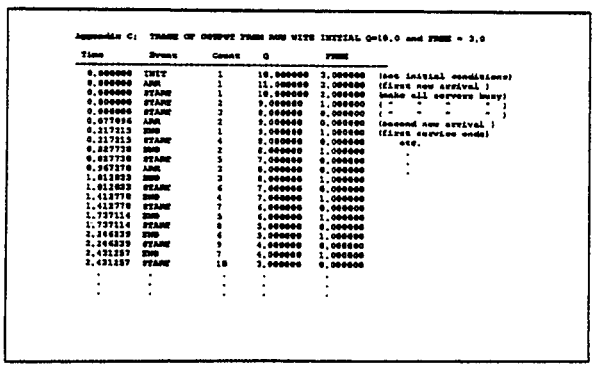

Figure 2. An Example Output File

2. DETAILS ON Σ

2.1. Using Σ

Σ displays each command as a button on the main menu. to choose any command you must select it by clicking the mouse while the graphics cursor is on the button.

To edit a vertex or an edge, all buttons must be "off" before you select an object.

2.2. Finding Your Way Around

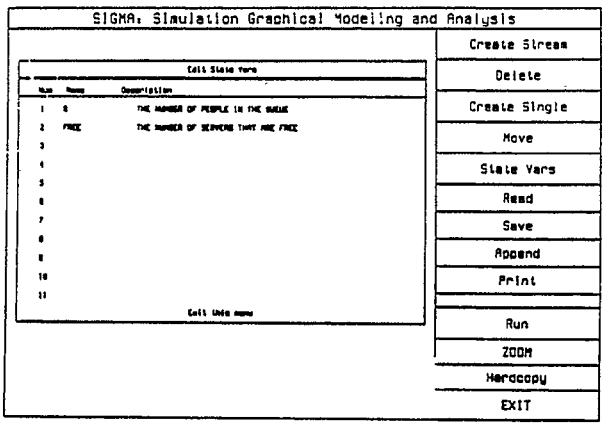

Figure 3. A sample Popup Menu (State Vars Popup)

Exit: The first and most important thing to know about Σ or any other piece of software is how to escape if it exhibits weird behavior or you get bored and want to do something else. The *correct* way to exit Σ is to select the EXIT button.

Create Stream: This will be the easiest way to create a graph from scratch. *Create Stream* creates a vertex at the location where you click the mouse. It also creates an edge between the last vertex created and the current one. As you keep clicking, Σ will create a "stream" of vertices and edges. If you are starting a new graph, the first mouse click will create vertex #1, the initial event. To make self edges, (edges that have the same origination and destination vertex) you can click twice at the same location; the first click will create the vertex, and the second click will create the self edge. NOTE: *Create Stream* will always remember the last vertex that was created, thus, it will continually create an edge between vertices.

Create Single: *Create Single* is just like *Create Stream*, except it will only remember one vertex at a time. First, you are asked to select an "out vertex" and then an "in vertex". These will correspond to the origination and destination of one edge. After

you have selected an "in vertex", you will be prompted to select another "out vertex"; unlike *Create Stream*, your next selection will not create a vertex. This would be very useful for adding edges and/or nodes to different places of your graph. As before, make sure you are out of *Create Single* before you attempt to edit any object.

Delete: Once you are in *Delete* mode, any attempt to edit an object or a *State Var* will prompt you to delete it. This prompt will always require keyboard input of either "Y" for yes or "N" for no. A carriage return will be taken as the default "N". If you select an edge while *Delete* is on, a popup will appear with all the sub–edges in that graphics edge. (Even if only one sub–edge exists.) You must then select which edge you would like to delete. Deleting *State vars* will operate in the same manner. Just select the *State Var* option while *Delete* is turned on.

Move: *Move* allows you to move a vertex. Σ prompts you to select a vertex and then to select a new location for it. All edges will be appropriately redrawn. You may not *M*ove a vertex on to another vertex or on an edge.

State Var: This option lets you create and edit state variables. First, a popup menu containing all the defined *State Vars* will appear. To edit an existing state variable, just select it. (Remember that a selection in a popup is done with the mouse.) To create a new State Variable, select an empty one. Or, if you move the highlight to an empty option, all you need to do is start typing. It is also important to remember that all State Variable names have a maximum length of 10 characters.

Once you have selected a State Var in the first popup, a new popup will appear. (See Figure 5.) This popup contains all the data fields associated with each State Var. These are *Name*, *Description*, and *Make it an Array*. If you select *Make it an Array* you may either enter a "Y" or "N". All arrays are referred to as *Name*[#] where # represents the index into the array.

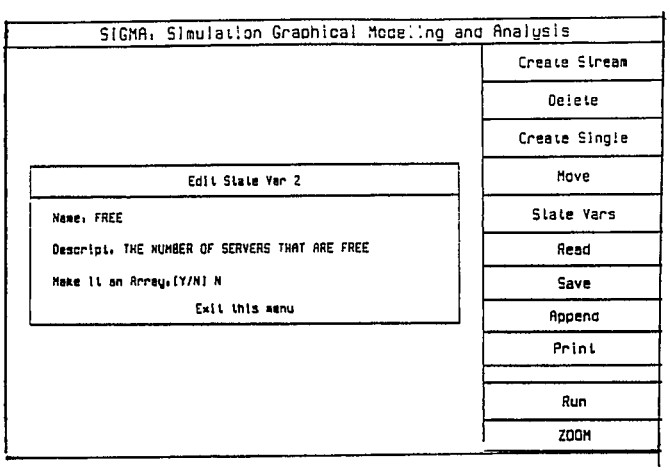

Figure 4. Sample State Var Popup

Zoom: *Zoom* allows the user to create larger simulation models. When selected, *Zoom* will zoom out to make the screen four times as large. You will then be prompted to choose a center of your screen. This location will be selected at your next mouse click. *Zoom* then makes that location the center of your viewing area and makes the Graphics window the same size as it was before.

Clear: As expected, this clears the system of all models. You will always be prompted to make sure you do not *Clear* accidentally. Once a system is cleared, the model is lost, so make sure you save your models often.

DOS: You can get from Σ to DOS by selecting the *DOS* button. You can execute all common DOS commands by entering them after this prompt. This is useful, for example, to look at your disk directory or to copy an output file. To return to Σ from DOS type "exit".

2.3. Saving, Reading, and Running a Σ Model

Save: Before you invest a lot of effort in building a simulation graph you need to know how to save your work. This is done with the "Save" command. You will be prompted for a name under which the model will be saved; the convention for files containing simulation model graphs is "xxxxxx.mod" where x's are your filename. It is suggested that all models be saved with the file extension ".mod". This is useful for debugging and homework documentation.

Read: Read lets you recall a simulation graph constructed earlier. You will be asked to enter the name of the file in which the simulation graph has been saved.

Run: Run will cause Σ to execute your simulation model. You are prompted for a sequence of parameters that control the initial conditions and ending conditions for a run.

1. Starting conditions: The first prompt will ask for starting attributes; these will be discussed later under the topic of event attributes. One of the starting conditions is a random number seed which is any number typed in by the user. Since there is a default seed, you may just hit return to continue.
2. Ending conditions: The next prompt concerns the duration of the run. A simulation run can be terminated after a particular event occurs a specified number of times or after a particular time. For example, suppose that the simulation of a queue is to be run until the 10th customer departs. Then the run would be controlled by the "customer departure" event and the number of executions of this event will be set at 10. To control the run duration based on a particular event, press "E" (for Event) at the prompt. You will then be asked which event (enter the event name) and the number of executions of that event before the run is terminated. Run

control is often based on the implicit event that the simulation clock reaches a particular time. If you want to run for a specific Time, press "T" at the prompt. You will then be asked how much simulated time the run is to take (Σ runs until the first event after the specified time). Note: there are no defaults for these questions so you must hit either "T" or "E".

3. Trace Variables: Next you are asked for the state variables to be displayed and recorded during the run. You are to list the state variables (with subscripts if appropriate) in a string separated by commas.
4. Run Mode: There is an option of using a single step run mode. In this run mode simulation execution will halt after each event and wait until the enter key is pushed. This is useful in debugging the logic of the simulation graph. Or you may choose high speed mode. High speed mode turns off the graphics and just creates an output file. High speed is very useful for running large simulations for a long time. NOTE: The default values for both modes is "N", and by pressing return you can continue.
5. Output file name: Here you are asked for a DOS file name in which values of the trace variables are recorded during the run. For all runs the clock time is recorded along with the values of the trace variables after the execution of *every* event during the run. A running total of the number of event executions is also recorded.

2.4. Building a Simulation Graph

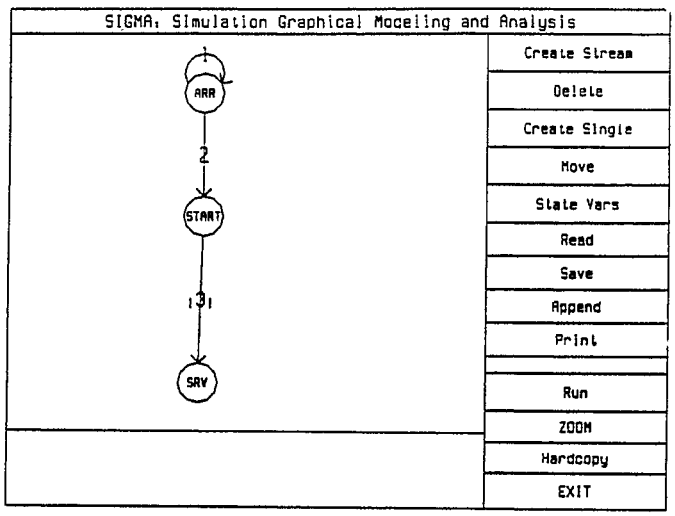

Figure 5. Simple M/M/1 queue Model in Sigma

Create : Typically three sets of objects must be defined in order to construct a simulation program using Σ. These are the set of state variables, the set of event vertices, and the edges connecting the vertices of the simulation graph. These sets of objects can be defined in any order (not necessarily sequencially).

Creating State Variables : You must give the variable a name by typing a name for the variable. It is this name that will be used in expressions in the event vertex and edge definitions. You can also type a description of the variable by selecting "Description". Finally, you will be asked whether you want the variable to be an array or not.

You can also use the powerful modeling device called an attribute Handler which will be explained after we discuss editing edges in the graph.

Event vertex 1 is always executed first. If several events are to be simultaneously executed or scheduled at the start of a run, then create a "start simulation" event in vertex 1 and have this vertex schedule the other vertices (typically unconditionally with zero delay times). Vertex 1 uses attribute handlers to assign initial values to state variables.

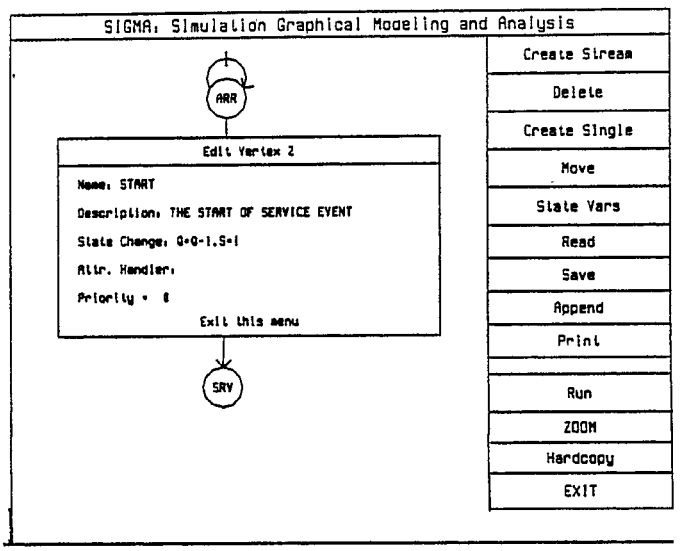

Figure 6. A Sample Vertex Popup menu

Associated with each event there will typically be one or more changes in state variables. State changes are simply entered as expressions. The operators you can use are +, −, /, *, and ^ for add, subtract, divide, multiply and exponentiation respectively. Expressions are separated by commas and the string can be up to 100 characters long. For example, consider the "START" event window in Figure 6. The state changes are given as the strong, Q=Q−1, S=1. Here, when the "START" event occurs the number of customers waiting, Q, is decreased by one and the status of the server, S, is set equal to 1 (busy).

There are two special variables that you can use. These are RND and CLK. RND is a uniform random number between 0 and 1 and CLK is the current simulated time.

In editing an edge you have the following options.

1. **Scheduling or canceling edge:** This ia a yes/no switch (yes for scheduling, no for canceling). It indicates whether the occurrence of the originating event vertex causes the estimation event vertex to be scheduled to occur or if it cancels the destination event if it has been previously scheduled. Canceling edges appear as dotted arrows in the event graph whereas scheduling edges are solid.
2. **Edge delay time:** This is the expression for the delay time between the occurrence of the origination event vertex and the occurrence of the destination event vertex. (Event cancellations occur immediately). A typical expression is ((RND*30)+20) meaning that the time between the two events is distributed uniformly between 20 and 50.
3. **Edge conditions:** The scheduling (canceling) of the destination event can be made conditional on the state of the system at the time the origination event occurs. The conditions are entered as normal expressions. You may combine conditions by using the Boolean operators AND and OR. For example, the edge condition, QUEUE>1 AND STATUS=0 means that the destination event will be scheduled to occur (after the edge delay time) only if QUEUE is greater than one and STATUS is equal to zero. The operators you may use in edge conditions are (+,–,/,*,^) plus
   = ... equal to
   !=... not equal to
   < ... less than,
   > ... greater than
   <=... less than or equal to
   >=... greater than or equal to, and
   the Boolean operators "AND" and "OR", which can also be represented as "&" an "||".
4. **Event attributes** (edge Attributes and vertex attribute Handlers): Each edge has an Attribute list and each vertex has an attribute Handler list. These are rather simple but powerful modeling tools. The edge Attribute list is simply a string of expressions separated by commas. The vertex attribute Handler list is a string of state variables separated by commas. When the origination vertex for an edge is executed the expressions in the edge attribute list are evaluated. These values are placed in the destination vertex attribute Handlers. When the destination vertex is subsequently executed the state variables in its attribute Handler list take on the values assigned to the corresponding expressions in the scheduling edge attribute list. Simply think of the vertex Handler list as variables of the left hand sides of equations and the edge Attribute list as expressions for the right hand sides of the corresponding equations. Note: These equations are evaluated when an event is scheduled not when it is executed.

Example: Consider an edge with Attribute list X + 1, 5, QUEUE*N. Say that the destination vertex for this edge has the attribute Handler list, Z, N, QUEUE. This is as if the destination vertex had state changes given by Z = X + 1, N = 5, QUEUE = QUEUE*N; however, with one difference. The expressions giving the values of these state variables (the left hand sides of the equations) are evaluated when the event is scheduled not when it executes. Suppose when the origination vertex for an edge is executed that the state variables have these values; X = 5, N = 10, QUEUE = 2. When the destination event is executed the values for the state variables Z, N, and QUEUE will be assigned to be 6,5, and 20 regardless of the current values for X and QUEUE.

The main value of event attributes is in defining the particular system entities to which an event pertains. For example, suppose that there were two identical machines in a simulated factory. The same "start processing" event vertex may be used for both machines if the event has need for an attribute telling which machine is to start.

Another common use of event attributes is in initializing a simulation program. When you run a model, you are first asked for initial attributes if any have been specified. You can enter a string of initial values for the attribute handlers for event 1 which is always the initial event. An example is a multiple server queueing system where the number of servers might change from run to run.

Event cancellation: When an event canceling edge is invoked any previously schedule event that has an exact match of the edge attribute values in its attribute handler is canceled.

2.5 A SAMPLE MODEL

Figure 7 is a screen dump from Σ. It is the representation of a multiple server queuing system simulation as developed on Σ. The events are as follows:

INIT: Here we initialize the state variables and start the simulation run. Values for state variables are requested from the user at the beginning or Run.

ARR: This is the arrival of another customer to the system.

START: This is the beginning of service for a customer.

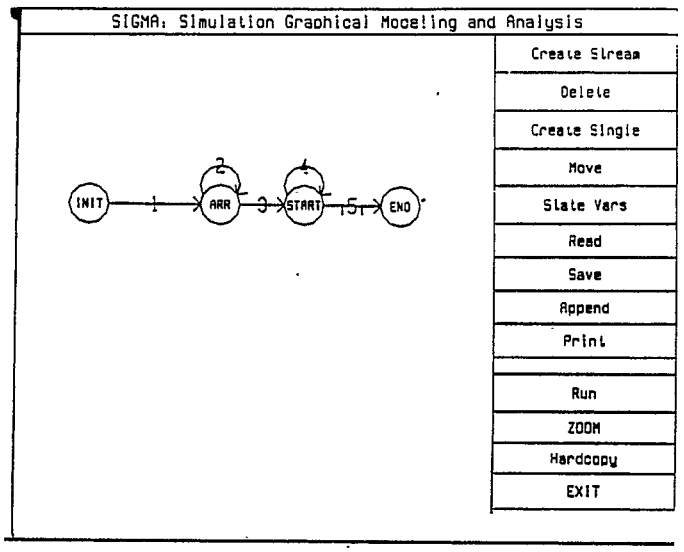

Figure 7. Multi-Server Queue Model in Sigma

END: This is the event where the server completes service on a customer.

2.6. USER DEFINED FILES (PSEUDO–VARIABLES)

The user can create a "user" pseudo–variable by enclosing a valid DOS file in curly brackets, { }. This pseudo–variable can be used anywhere a state variable can be used. When this expression is encountered during execution a number from the file is read into the simulation. User input data files are simply strings of real numbers (floating point numbers) separated by a space or line feed. The number read from the user data file, depends on the offset parameter which follows the filename separated by a semicolon.

Example: The expression, (USER { DELAY.DAT;3 }), will cause the third number in the disk file named *delay.dat* to be read. For example, consider the expression,

NUM = RND*5+1,QUEUE
= (QUEUE + USER { SIZE.DAT;NUM }).

This can be used to model arrivals of randomly sized groups of customers to a service facility; as in a simulated bus station. Note that the offset number is a random integer between 1 and 5 inclusive. The possible group sizes will be listed as 5 numbers in the disk file named groupsiz.at. Each of the five different group sizes will be selected uniformly at random each time the above expression is executed. Note also that the expression after the ";" must either be a constant or a single variable; other expressions must be evaluated as above.

To increment through a file sequentially, use the constant "0" as the index. (I.e. USER { DELAY>DAT;0 }).

When the offset parameter in a user pseudo–variable is 0 the file is read sequentially during the run. When the end of a file is reached a prompt asks whether the user wants to start over at the beginning of the file or terminate the run. In HIGH SPEED run mode the user file is cycled through as many times as needed without the prompt.

REFERENCES

[1] Schruben, L. "Simulation Modeling with Event Graphs," *Comm. A.C.M.* Vol. 26.11.

[2] Schruben, L. "A 'Disposable' Graphical Event Synthesizer for Teaching Simulation Model Building," *Proc. 1987 WSC*, pp. 72–76.

Authors' Biographies

Lee Schruben is on the faculty of the School of Operations Research and Industrial Engineering, Cornell University. He holds degrees from Cornell, the University of North Carolina and Yale.

David Briskman recently completed his Master's of Engineering at Cornell University and is presently employed as an industrial engineer by General Foods.