Proceedings of the 1995 Winter Simulation Conference
ed. C. Alexopoulos, K. Kang, W. R. Lilegdon, and D. Goldsman

BUILDING REUSABLE SIMULATORS USING HIERARCHICAL EVENT GRAPHS

Lee W. Schruben
Cornell University
Ithaca, NY 14850

ABSTRACT

Hierarchical event graphs are an easy way to build special purpose simulators. At the lowest level, event graphs are created to represent particular components of the system being simulated, steps in a process flow, or hyper-events. These low-level graphs can then be viewed as different classes of vertices that make up the next higher level graph. A special purpose simulation toolkit is thus developed.

1 INTRODUCTION

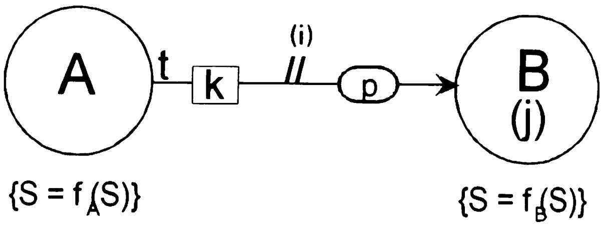
Three very different types of hierarchical event-graph simulation toolkits are discussed in this article: a Petri Net simulator that is used to teach the activity-scanning approach to simulation modeling, SIMAN and GPSS network simulators that are used to teach process-interaction modeling and introduce these languages, and a industrial process simulator called QUALPLAN that is used for planning quality inspection systems.

2 ELEMENTS OF AN EVENT GRAPH

A discrete event model of a system can be created by defining a set of state variables that describe the system, a set of events that will change the values of these state variables, and the relationships between the events. An event graph (Schruben 1983) has vertices (nodes) representing elementary events connected by directed edges (arrows) representing the cause-and-effect relationships between the events. Each event vertex is associated with the state changes that happen when the event occurs. Each edge is associated with the conditions and time delays that govern how one event might cause another event to occur in the future. Only one basic modeling object provides enough modeling power to simulate *any* discrete event system (Yucesan 1993).

2.1 Event Graph Elements and Behavior

In complete detail, an object in an event graph would appear as follows:



The detailed behaviors of the basic elements of an event graph are defined as follows:

1. *Edges:* After each occurrence of event A, if condition (i) is then true, event B will be scheduled to occur after a delay of t. Potential time ties are broken with event B receiving an execution priority, p.

2. *Vertices:* Whenever event B occurs, the state variable(s) in the set of event parameter(s) j will be assigned the values of the expression(s) k computed when B was scheduled. The state of the system will then change from S to $f_B(S)$. The edge conditions for all edges exiting B will then be tested and those found true will have their destination events scheduled.

Also, to make certain modeling tasks easier, events are allowed to cancel one another; an event cancellation edge is represented in an event graph as with a dashed arrow. Vertices can schedule or cancel further instances of themselves.

All of the elements of the event graph can be statements or expressions; for instance, event execution priorities, p, can be real-valued expressions allowing for dynamic state-dependent event sequencing.

There are two fundamental features of a simulation model that distinguish it from other types of computer programs. First, there is the ability to represent the passage of time; second, there is the ability to represent randomness. Thus, two variables (actually functions without arguments) are reserved for use by the event graph to represent the dynamic and stochastic behavior in a model: CLK, which is the current simulated time, and RND, which is a random number.

2.2 Event Graph Simulation

Associated with each event vertex in an event graph is a number (which is initially the order in which the vertices in the graph were created). When an event graph (or sub-graph) simulation is "run", the lowest numbered vertex will always be the first vertex executed. The event graph simulation run is stopped when the highest numbered event vertex is executed. (Software implementations of event graph modeling may allow different types of stopping criteria.) It is useful in what follows to think of an event graph as a function that maps the "initial" system state S (input) into a real valued number (output). By using the C construct of allowing a state to be the address of a variable, an event graph function can be quite general.

3 HIERARCHICAL EVENT GRAPHS

3.1 Graph Elements as Sub-Graphs

A possible hierarchy for event graph models is to use event graphs as elements of a higher level event graph. For example, the delay time on an edge might be the output value of a lower-level event graph simulation. This might be used, say, when running a factory simulation. The delay time for a machining process might be the output from the event graph of a more detailed machine simulation. The machine level simulation might be run whenever there is a change in the operation.

3.2 Nested Event Graphs

The approach to hierarchical event graphs used here is to allow each vertex in an event graph to itself be an event sub-graph. Edges into this sub-graph/vertex will connect to the lowest numbered vertex in the sub-graph. Edges exiting this sub-graph/vertex will originate at the highest numbered vertex in the sub-graph. The sub-graph can be grouped into a single vertex and treated just as if it were a conventional vertex. With this vertex numbering scheme and entering and exiting rule for sub-graph/vertices, executing a sub-graph/vertex is exactly equivalent to running the sub-graph simulation. The sub-graph/vertices are essentially the same as the "super events" in the seminal article by Som and Sargent (1989). In the remainder of this paper several examples of event graph simulations are presented.

4 A PETRI NET SIMULATOR

A special purpose simulator of generalized timed Petri Nets (Peterson 1985) was created as a hierarchical event graph. This Petri Net simulator is used in an undergraduate simulation course to introduce students to the activity scanning approach to simulation modeling. Figure 1 shows a Petri Net representing two tandem multi-server queues.

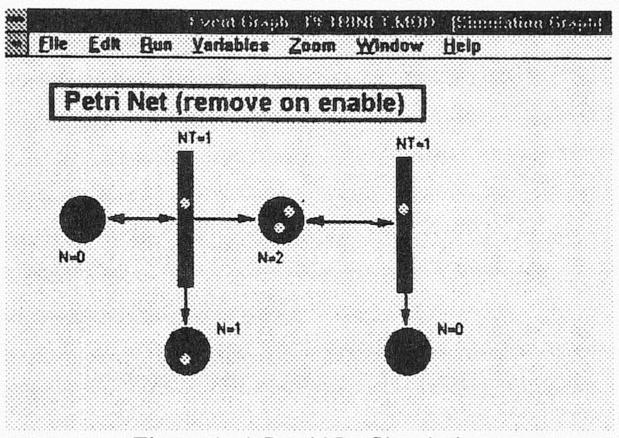


Figure 1: A Petri Net Simulation

5 SIMAN and GPSS SIMULATORS

Sub-graphs are created representing various SIMAN and GPSS modeling blocks. An example of the SIMAN (Pegden 1986) simulator of primary and overflow processing resources is shown in Figure 2.

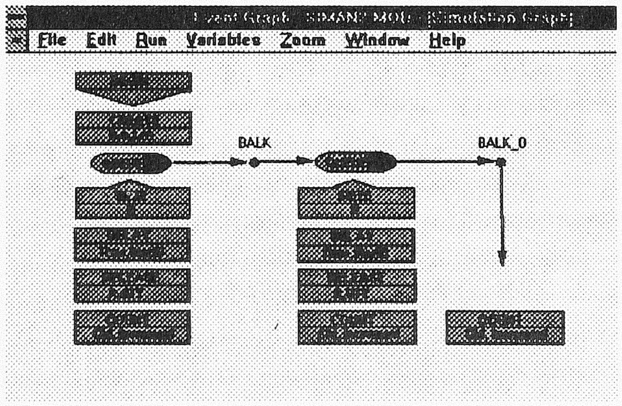
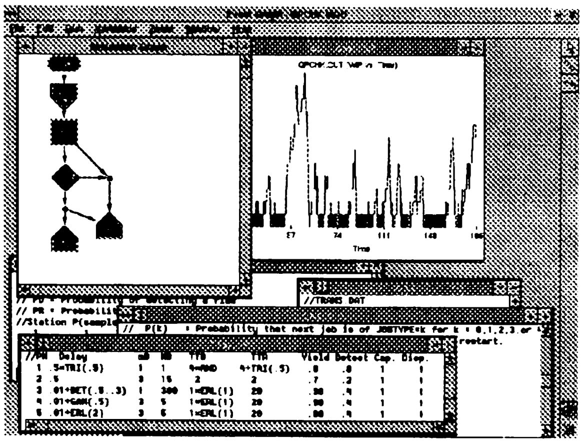


Figure 2: A Simulator of a SIMAN Model

6 QUALPLAN

The QUALPLAN process planning tool is a high-level process simulator designed specifically to aid in planning quality inspection in production operations. Once a flowchart description of the process flow is determined, quality inspections can be added to improve the average outgoing quality level (AOQL) of the finished products. QUALPLAN is intended to be flexible and easy to learn and use.

A basic QUALPLAN modeling session might look as follows: (Note that various data (`*.DAT`) files are open.)



```
// PR = Probabilit
//Station P(sample
// P(k)  = Probability that next job is of JOBTYPE=k for k = 0,1,2,3,or 4
restart.
//PN  Delay          [illegible] [illegible] TTF      TTR       Yield Detect Cap. Disp.
  1 .5*TRI(.5)        1   1    4*RND    4+TRI(.5)  .8    .8     1    1
  2 .5                3   15   2        2          .7    .2     1    1
  3 .01+BET(.5..3)    1   300  1*ERL(1) 20         .98   .4     1    1
  4 .01+GAM(.5)       3   5    1*ERL(1) 20         .98   .4     1    1
  5 .01+ERL(2)        3   6    1*ERL(1) 20         .98   .4     1    1
```

Figure 3: A Qualplan Working Session

QUALPLAN is a commercial simulator that is used to design quality inspection systems. It allows modeling different process routes for multiple products and contains eight basic high level modeling blocks. Transparent to the user, these blocks are themselves multiple level event graphs. The modeling blocks included are:

STEP: This is a generic processing step. Each STEP is associated with a record in the data file `STEP.DAT`. This data file completely specifies:

processing time distributions,
number of parallel resources available,
step yield,
probability of detecting an incoming flaw,
minimum batch size,
maximum batch size,
resource failure time distributions, and
resource repair time distributions.

These data files can be edited while a model runs.

CHECK: This is a generic inspection step. A CHECK block points to a particular record in the `CHECK.DAT` file which specifies rules and probabilities for sampling a product, the probability of detecting a flaw in a sampled product, and the probability of being able to rework a detected flaw.

ENTER: These blocks are where new jobs enter the system. The data file, `ENTER.DAT`, contains initial delays and inter-event time distributions as well as the average incoming quality of jobs received at the block and probability distributions for part types and routings.

EXIT and **SCRAP:** These blocks are where jobs exit the system. Cycle time, WIP, and quality level statistics are automatically generated in an EXIT block.

TRANSPORT: A transport block has an associated delay time probability distribution in a data file.

CHANGE: When a job enters this block, its attributes can be changed. Job attributes include

AGE (time since it entered the system),
GONOGO (good or flawed),
JID (job ID number), and
JOBTYPE (type of job).

START: This is used to initiate the simulation.

7 SUMMARY

Two approaches to hierarchical event graphs are suggested. One of these methods was used to implement different types of simulators using an activity (Petri Net) world view, a process (SIMAN or GPSS) world view, or for a specialized application in quality control. The variety of these applications illustrate the generality of hierarchical event graphs in modeling different systems and offer examples of modeling with different simulation world views.

ACKNOWLEDGMENT

I am grateful to the National Science Foundation for sponsoring a research project of which this paper is a part. Though these simulators appear and behave very differently, all were created using the same event graph paradigm and software, SIGMA for Windows (Schruben 1994). QUALPLAN is an example of a special purpose simulator built using the SIGMA

simulator design tool kit and can be easily extended and modified with SIGMA.

REFERENCES

Pegden, C. D. 1986. *Introduction to SIMAN, 2nd ed.*, Systems Modeling Corp.

Peterson, J. L. 1977. Petri Nets. *Computing Surveys*, 9(3), 223-252.

Schruben, L. 1983. Simulation modeling with event graph models. *Communications of the Association of Computing Machinery* 26(11), 957-963.

Schruben, L. 1994, *Graphical Simulation Modeling and Analysis using SIGMA for Windows (3rd ed.)*, The Scientific Press, Danvers, MA.

Som T. K. and R. G. Sargent, 1989. A formal development of event graph models as an aid to structured and efficient simulation programs. *ORSA J. on Comput.* 1, 107-125.

Yucesan, E. 1993. On the modeling power of simulation graphs, Technical Report, INSEAD, Fontainebleau, France.

AUTHOR 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 and a Ph.D. is from Yale. His research interests are in statistical design and analysis of simulation experiments and in graphical simulation modeling methods.