Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

GRAPHICAL MODEL STRUCTURES FOR DISCRETE EVENT SIMULATION

Lee W. Schruben

School of Operations Research and Industrial Engineering
Cornell University, Ithaca, New York 14853
and
SEMATECH, Austin, Texas 78741-6499 U.S.A.

ABSTRACT

Several different graphical techniques for specifying models for discrete event systems are reviewed including process networks, generalized stochastic Petri nets, stochastic state diagrams, and event graphs. This paper presents a brief summary of these modeling approaches and highlights some of their similarities and differences.

1. BACKGROUND AND DEFINITIONS

A system is defined as a

set of entities that interact for shared purposes according to sets of common laws and policies for an interval of time.

A system can be either real or hypothetical. The interval of time for which the system is defined is called its *lifetime;* this is the amount of time we are interested in studying the system. The entities in a system are either *resident*, remaining in the system for its entire lifetime, or *transient,* entering and exiting the system as time passes.

A model is simply

a system that is used for a surrogate of another system.

A simulation is a computer program used as a model for some other system of interest. In a simulation model the entities are described by numerical (coded) attributes. The state of the simulation includes the values for all of its attributes as well as what is known about the future. We define an event as any situation where the state of the system might possibly change. In a discrete event dynamic system all changes in state occur at discrete instants of time. A discrete event system is typically an idealized or abstract system that is used as a model for a more complex system.

In this paper, we concentrate on tools for specifying the behavior of discrete event systems that are *complete* graphical representations. That is, a directed graph can be drawn with labeled edges (arrows) and vertices (balls or blocks) forming a network that completely defines the *structure* of a specific system. Along with an initial state, stopping conditions, and a specified input process, the graph completely describes a specific model's *behavior*. Structural and behavioral properties for simulations have been defined in [Yucesan and Schruben, 1992]. For each graphical model there is a set of implicit rules (e.g., time advance algorithms) that define how the input is processed to produce the model's output. There are many such graphical procedures; only some of the more popular approaches will be presented. In particular, we will not include the many special-purpose approaches for building simulators.

One of the most distinctive features of these modeling methodologies is the number of different types of modeling objects used. Objects are considered to be of a different type if they are defined by different *rules* of behavior; different types of objects are typically represented in the graphs by different shapes or as having different names. Different types of graphical objects are sometimes referred to as modeling "blocks".

Methodologies that have few types of graphical objects are easier to learn than those with many different types of objects. Methodologies that have many types of objects may be harder to learn but may be easier to use or understand once they are mastered.

Surprisingly, having a large number of different types of graphical modeling objects does not imply that the technique is more powerful. If anything, just the opposite tends to be true. This has to do with the modeling philosophy behind each approach.

One extreme is to try and identify all of the possible situations that might need to be modeled and define a special macro block or graphical object for each

situation. We will refer to this approach as the macro-modeling philosophy.

At the other extreme, a single elementary graphical object might be defined that theoretically can be used to model quite general situations. We will refer to this viewpoint as the minimalist philosophy.

The macro-modeling approach requires that a new graphical object be defined for each new system behavior that needs to be modeled. As new situations are encountered, new blocks are defined. Such techniques can easily swell into a relatively awkward modeling tool that requires a college semester or more to learn. An elegant general minimalist technique might be learned in a matter of seconds but require experience and cleverness when modeling complicated systems.

2. PROCESS NETWORKS

The process network modeling approach is one of the oldest approaches to modeling discrete event systems. One of the first simulation languages to adopt this technique is GPSS. An up to date accounting of one of the dialects of GPSS can be found in [Schriber, 1991]. With GPSS a controlled macro-modeling philosophy has successfully been followed where new modeling blocks have been introduced as their need is perceived. The original GPSS blocks that generate transient entities to enter a system and describe how they queue for and seize resident entities have been augmented by specialized blocks that model such things as the failure of system components. Currently, there are over 60 types of blocks defined for some versions of GPSS. The network modeling subsystem in SIMAN [Pegden, et. al., 1990], follows a similar approach. The current network modeling part of SLAM [Pritsker, 1986] is also similar to SIMAN and GPSS with the exception that the passage of time is modeled using edges of the graph rather than in time "advance" or "delay" vertices used by GPSS and SIMAN.

A minimalist approach to process modeling has been taken by some simulators of specialized types of systems. An example is the original XCELL factory modeling system, which defined only four types of blocks to model production systems. XCELL has since been enriched along the historical development lines followed by the other process networking modeling techniques mentioned above but in a much more conservative manner. However, the introduction of new graphical objects has been resisted; new features have been added only when deemed absolutely necessary [Conway, et. al., 1992].

3. PETRI NETS

Petri nets originated with the minimalist modeling philosophy but have been enriched to include elements needed for modeling stochastic discrete event systems [Peterson, 1977]. A simple Petri net is a graph with two types of vertex labels, called places (represented by balls) and instantaneous transitions (represented by bars). Places can be occupied by tokens (block dots). The behavior rule is simple: when *all* of the input places to a transition are marked with tokens, the transition fires and a token is placed in each of the output places for that transaction.

Simple Petri nets have been generalized to include multiple tokens at a place, time delayed transitions (graphically represented by thick bars), inhibitor arcs (represented by having little circles replace edge arrows), and branching (one of several output places is chosen at random to receive a token once an instantaneous transition fires). An inhibitor arc prevents a transition from firing if its corresponding input place is occupied by a token.

An example of a generalized stochastic Petri net (GSPN) model of the failure and repair process of three machines is shown in Figure 1. The current marking of this net with tokens indicates that two machines are working (top place) and one machine is under repair (lower right place). The timed transition at the far left models the generation of machine failures. This example can be found in the proceedings of the 1992 European Simulation Multi-conference in York, England. Exponential delays are assumed for all timed transitions. This graph shows most of the basic graphical objects in a GSPN: timed and instantaneous transitions, multiple tokens, and an inhibitor arc.

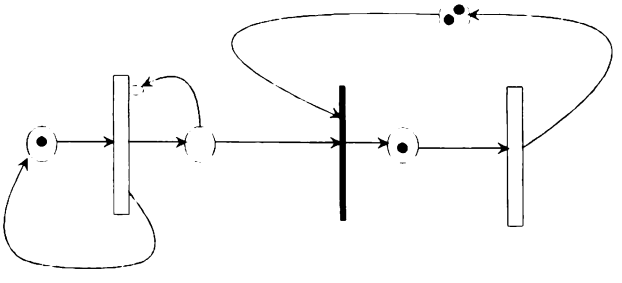


Figure 1: Exponential failure and repair of three machines

The model in Figure 1 is merely intended to give the flavor of a GSPN model. There are many other enrichments of Petri nets; some that are specifically designed for discrete event simulation can be found in [Torn, 1990].

4. STOCHASTIC STATE MACHINES

Stochastic state machine transition diagrams (SDs) have vertices that are labeled with each of the different possible states of a system [Heyman and Sobel, 1982]. The edges are labeled with the rates that the system moves from one state to another. An example of a stochastic state transition diagram for the above three machine exponential failure and repair system is shown in Figure 2. The failure rate is denoted as λ and the repair rate for a single machine is denoted as μ. Each vertex is labeled with the state denoting the number of machines in good repair.

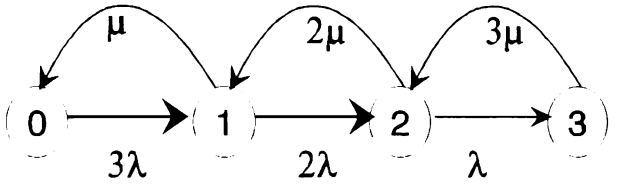


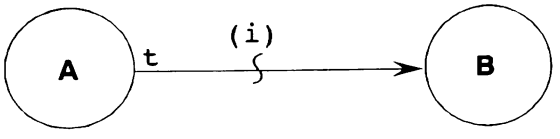
Figure 2. State Diagram for exponential failure and Repair of 3 Machines

Both GSPN models and state transition models permit analytical solutions for relatively simple systems with exponential delay times. However, they quickly become unwieldy when the systems being modeled are even moderately complex. The exponential delay assumption is critical for the analytical solution of these models, but not necessary for simulation models.

5. EVENT GRAPHS

Event graphs (EGs) can be used to build models of any discrete event system using just a *single* graphical object. The vertices represent changes in the values of system state variables. Edges resent the conditions under which one event might cause the occurrence of the other event as well as the time interval between the two events.

The graphical representation of the basic modeling object is as follows;



This edge is interpreted as follows:

if condition (i) *is true at the instant event* A *occurs, then event* B *will be scheduled to occur* t *minutes later.*

If the condition is not true, nothing will happen. The EG representation is completely general in that any discrete event system (indeed any computer program) can be represented using this object [Yucesan, 1989]. EGs epitomize the minimalist philosophy in that the above object is all that is necessary to have a completely general graphical modeling tool. Modeling conveniences such as parameterized vertices and canceling edges have been added as enrichments to the simple EG definition [Schruben, 1992]. Detailed analysis and definitions have also been developed [Yucesan, 1989, Som and Sargent, 1989].

An event graph for the exponential failure and repair for any number of machines is shown in Figure 3, where failure and repair times are subscripted with f and r respectively.

The vertices of this EG has been "marked" with tokens that indicate how many instances of each event are scheduled to occur in the future. There are 3 machines in this marking; two are working (left vertex) and one is broken (right vertex).

Like Petri nets, the marking transition rule is simple: when a scheduled event vertex occurs (it "times out"), a token is removed from the vertex and tokens are placed in all immediate neighbor vertices connected by exiting edges with true conditions. The count of the tokens gives the number of events that are scheduled in the future.

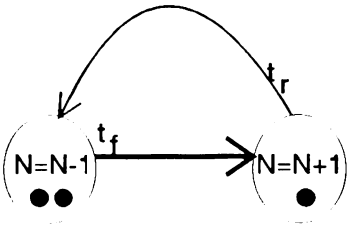


Figure 3: Failure and Repair of N(=3) Machines

The number of working machines is denoted as N. The failure and repair times here may follow any probability distribution and are not assumed to be exponential.

In a state transition diagram, there is a vertex for every possible *state;* in an event graph, there is a vertex for each possible *change* in state (event). The state transition diagram for N machines would require N+1 vertices. The event graph and the GSPN will not change as more machines are modeled; more tokens are merely added. However, the conditioning of the exponential delay times in the GSPN changes while

nothing changes in the EG. By modeling only the changes in state rather than every possible value of the state, EGs allow systems that may have many states to be represented with a finite graph. In fact, the state diagram for a classical M/M/1 queue is a graph with an infinite number of vertices whereas the EG model requires only a single vertex.

The edges of state diagrams and EGs are also complementary: state diagram edges are labeled with rates of change, EG edges are labeled with the times between changes. A combination of the two techniques, where vertices are labeled with changes in state (like EGs) and edges are labeled with rates of change (like SDs) is currently being studied.

A key enrichment to event graphs is the concept of parameterized vertices and edge attributes that permit a basic event graph to be used to represent different instances of similar subsystems. These graphs can be hierarchically linked into a model of a larger system. For example, an EG of a single generic machine cell can be parameterized to represent different types of cells which can be linked to a larger graph that models a complete factory. Thus, a generic factory simulator can be developed to an arbitrary degree of complexity.

A more detailed introduction to event graph modeling can be found in the tutorial by the author in this year's WSC proceedings.

6. SUMMARY

Several graphical techniques for specifying models of discrete event simulations have been presented. All of these graphs can be considered conceptually as representing generalized automata driven by random input processes [Wu and Chung, 1991], [Glasserman and Yao, 1991]. The different approaches have been described as primarily following a macro-modeling or minimalist philosophy. The macro-modeling philosophy has been successfully implemented in many process network simulation languages. This approach has the advantage of being easy to use within its problem domain and the disadvantage of requiring new graphical objects to expand its modeling capability. The minimalist approach is used in GSPNs, state machines, and event graphs. Event graphs have the advantage of having only one graphical modeling object but the disadvantage of requiring the modeler to understand the somewhat abstract concept of a system event. GSPNs and state machines quickly become so unwieldy as to make them ineffective graphical approaches for developing simulations of complex systems.

GSPNs have been demonstrated to have at least the modeling power for simulation of generalized semi-Markov Process (GSMP) models [Haas and Shedler, 1988]. It is easy to model all of the objects in a GSPN as an EG, giving EGs at least the modeling power of the other approaches. Recently an algebraic structure for GSPNs and GSMPs has been developed [Glasserman and Yao, 1991]. Set structures for EVs have also been proposed [Yucesan, 1989] [Som and Sargent, 1989].

ACKNOWLEDGMENTS

I appreciate the opportunity to join SEMATECH during 1992 and have benefited from many enlightening discussions with the staff of the Modeling and Statistical Methods Group.

REFERENCES

Conway, R., W. M. Maxwell, J. O. McClain, and S. Worona, (1992), *The XCELL+ Factory Modeling System (release 4)*, Scientific Press.

Glasserman, P. and D. D. Yao, 1991, Algebraic Structure of Some Stochastic Discrete Event Systems, with Applications, *Discrete Event Dynamic Systems: Theory and Applications*, 1, 7-25.

Hass, P. J., and G. S. Shedler, 1988, Modeling power of Stochastic Petri Nets for Simulation, *Probability in the Engineering and Informational Sciences*, 2, 435-459.

Heyman, D. P., and M. J. Sobel, 1982 *Stochastic Models in Operations Research*, Vol I, McGraw-Hill, 58-60.

Pegden, C. D., R. P. Sadowski, and R. E. Shannon, 1990, Introduction to Simulation Using SIMAN, Systems Modeling Corporation.

Peterson, J. L., Petri Nets, *Computing Surveys*, 9.3, 223-252.

Pritsker, A. A. B., 1986, An Introduction to Simulation with SLAM II 3rd. Ed., Halsted Press.

Schriber, T. J., 1991, *An Introduction to Simulation Using GPSS/H*, John Wiley and Sons.

Schruben, L., W. 1992, *Event Graph Modeling Using SIGMA*, (2nd release), The Scientific Press, S. San Francisco, CA.

Som T. K. and R. G. Sargent, 1989, A Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs, *ORSA J. on Comput.*, Vol 1, 107-125

Torn, A. A., 1990, Simulation Graphs: A General Tool for Modeling Simulation Designs, *Simulation* 37, 187-194.

Yucesan, E., 1989, Simulation Graphs for the Design and Analysis of Discrete Event Simulation Models,

Ph.D. dissertation, School of OR&IE, Cornell University, Ithaca, NY.

Yucesan, E., and L. W. Schruben, 1992, Structural and Behavioral Equivalence of Simulation Models, *Transactions on Modeling and Computer Simulation*, (to appear).

Wu, J-H, and C-N Chung, 1991, Timed Finite Automata as The Theoretical Foundation for Simulation Modeling with Event Graphs, Technical Report, Dept. of Dec. Sci. and Inf. Sys., University of Kentucky, Lexington, KY

AUTHOR BIOGRAPHY

Lee Schruben, a Professor in the School of Operations Research and Industrial Engineering at Cornell University, is currently visiting SEMATECH in Austin Texas.