Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

THE USE OF EVENT GRAPHS IN SIMULATION MODELING INSTRUCTION

Kevin J. Healy

School of Industrial Engineering
Purdue University
1287 Grissom Hall
West Lafayette, IN 47907-1287

ABSTRACT

There is a longstanding debate among educators about how best to teach the concepts involved in modeling and simulating *discrete event dynamic systems* (DEDS). The goal should be to convey fundamental modeling principles and to examine the implementation of these principles in real settings. One particularly contentious element of the debate deals with the choice of and degree of emphasis on means for implementing simulation models. This paper advocates the use of the *event graph modeling formalism* as a vehicle for conveying the underlying structure of DEDS models and their implementation.

1 INTRODUCTION

Building and implementing simulation models of discrete event dynamic systems is more than just a programming exercise. Too often though, programming aspects of simulation modeling are the focus of introductory courses in simulation. The tendency is to focus too closely on the syntax and semantics of the particular implementation to the exclusion of general principles of simulation modeling and analysis. Topics dealing with the statistical aspects associated with designing, conducting, and interpreting output from simulation-based experiments are a frequent casualty.

This is due in part to the nature of many traditional simulation tools which are not so much modeling formalisms as computer software systems designed to facilitate the implementation of a model. This makes it difficult at times to distinguish between a model and its implementation. It can also be attributed to the ad hoc design criteria employed in many of these tools which in some cases have produced a rather unharmonious assembly of features possessing both cumbersome syntax and semantics. Whatever the reason, as consequence of this misdirected focus, students may develop proficient programming skills but fail to develop more important modeling and problem solving skills.

At the lowest and most abstract level of implementation are general purpose programming languages. Before the advent of special purpose simulation software, general purpose programming languages like FORTRAN were the only generally available means of implementation. Many still advocate the use a general purpose programming language together with a library of routines to perform event-scheduling, file manipulation, and statistics collection, although the choice is more likely to be a modern language like C or Pascal.

More often than not, courses are designed around a commercial process simulation language like SIMAN (Pegden et al. 1992), SLAM (Pritsker 1986), GPSS (Schriber 1991), or SIMSCRIPT (Russell 1981). Facilities in these languages are designed to more closely mimic and describe interactions amongst the phenomena that characterize DEDS (e.g. entities, process delays, queueing, and resource allocation). Most also include higher level instances of these generic phenomena tailored to represent features characteristic of manufacturing systems such as material transport devices. It is also now common to provide exposure to companion software used to produce graphical animations of simulation output.

At the highest level are a newer generation of extensible data-driven simulators such as XCELL+ (Conway et al. 1987), Witness (Gilman and Gillingham 1989), ProModel (Harrell and Tumay 1992), and Arena (Pegden and Davis 1992). In contrast to traditional process languages, these systems employ elaborate interactive graphical means to combine and parameterize pre-defined (or in some cases user-defined) submodels to specify the logic of the simulation model and simultaneously the characteristics of the associated animation.

While the means of implementation have grown in sophistication, the relatively small group of basic concepts on which all event-driven simulations are based remains unchanged. It is relatively easy though for these concepts to become obscured by unwieldy languages or the glossy veneer of a graphical user interface. Only by understanding these basic concepts first can one make effective use of the technique or make meaningful comparisons among various means of implementation.

Some might argue that a more theoretical treatment is beyond the capacity of some and - to use an analogy - that a "cookbook" language-oriented approach can at least inculcate the skills sufficient to prevent death by starvation, if not instill an appreciation for the finer points of culinary science. Any benefits derived from such an approach though are usually short lived. The long term interests of a student are better served by a more broad-based treatment of the topic.

The same rationale applies to designing courses to teach programming language skills. The emphasis should be on the development of skills used in the formulation and specification of algorithmic solutions to problems as well as on the salient features of programming languages in general, and rationale for their design and inclusion in languages. It is both reasonable and convenient to then choose a particular language such as C, Pascal, or FORTRAN to examine an implementation of these principles in a real language as well as to develop practical skills in its use.

The structure underlying DEDS and their implementation involve relatively straightforward concepts. What is needed for educational purposes is a reasonably high level device with sufficient modeling power and minimal complexity that makes these concepts transparent. The event graph modeling formalism is a device conceived with these ideas in mind.

Situated on a level of abstraction above general purpose programming languages but below process simulation languages, event graphs are a simple yet expressive modeling tool. The resultant models lend themselves naturally to various means of implementation which helps to make clear the distinction between the two.

After characterizing the structure of DEDS, a more detailed description the event graph formalism is given followed by a simple example. A brief overview of SIGMA, a commercial software implementation of the event graph formalism is then given and followed by concluding remarks.

2 STRUCTURE OF DEDS

The behavior of a dynamic system can for modeling purposes be characterized as being *discrete event* if changes to the essential elements can be associated with identifiable events that occur at particular (possibly random) instances in time. The disposition of the elements that comprise the system are represented by the values of status variables and changes in the state of the system are modeled through the occurrence of the specified events.

One aspect of the system dynamics is described by how the isolated occurrence of an event changes the state of the system. In addition, the occurrence of an event might, under specified conditions, affect the behavior of the system by triggering one or more other events to occur at some future point in time. These dependencies constitute the other aspect to system dynamics. Together, the specification of the system state, event logic, and relationships between events comprise the model.

From this standpoint, the underlying structure of what are traditionally called *discrete event* and *process interaction* models is fundamentally the same. Each is merely an alternate representation arising from the particular perspective employed in defining the system state and driving events. The process approach focuses on the flow of transient entities - process languages are designed to facilitate this perspective - whereas the discrete event approach typically focuses on cycles of resident resources (e.g. the busy/idle status of a server).

It is instructive to think of the event logic being implemented (at some level) as functions in a general purpose programming language. The main procedure serves in an executive capacity, its job being to execute events in the proper time ordered sequence. This is accomplished by means of a global *simulation clock* and *future events list* that contains (to the extent possible) known information about future events and their time of execution.

After initialization, the process evolves by advancing the simulation clock to the time of the most imminent event that is scheduled to occur and then invoking the associated event routine. The latter alters the values of system status variables and may schedule future events by adding appropriate information to the future events list.

3 EVENT GRAPHS

The event graph formalism (Schruben 1983) uses simple directed graphs to model the activity in a DEDS. Unlike many other modeling tools, there is a simple and well-developed mathematics connected with the syntax and semantics of the formalism (Schruben and Yucesan 1988).

Events and their associated state changes are represented as vertices (nodes) while the logical and temporal relationships amongst events are represented by directed edges (arcs) between pairs of event vertices. Almost everything else there is to know can be explained by means of the simple construct depicted in Figure 1 which is interpreted as follows.

Whenever event A occurs...
1. *Execute the set of state changes* $\{S_A\}$.
2. *If condition* (i) *is true, then schedule event B to occur t time units later.*

Each element in the set of state changes, $\{S_A\}$, represents an assignment to the value of a state variable. In their most general form, the values of the assignments as well as the time delay, t, and triggering condition, i, are expressions involving combinations of constants, arithmetic operators and system status variables. The condition i may also contain logical and relational operators. The syntactic rules for specifying these expressions are similar to those employed in the C programming language.

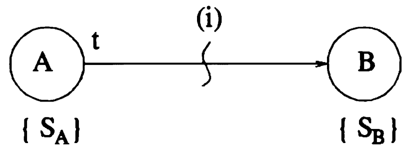

Figure 1: Basic Event Graph Construct.

Figure 2 depicts an event graph model of the single server queue. The system status variables are S, the status of the server (0 = *busy*, 1 = *idle*), and Q, the number of customers residing in queue. The time between successive arrivals and time required for service are denoted by t_a and t_s, respectively.

Although it is not reflected in the graph, t_a and t_s would generally be specified as expressions involving the generation of random variates. By convention, we will also assume the triggering condition, i, when omitted defaults to *unconditional* and t, the scheduling delay, defaults to 0.

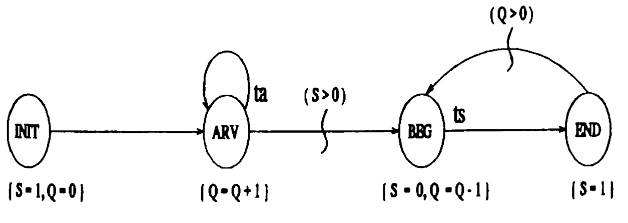

Figure 2: Event Graph Model of Single-Server Queue.

Conceptually, the *INIT* event (which has no triggering events) is processed once at the beginning of the simulation. It initializes the queue to empty ($Q = 0$), server to idle ($S = 1$), and triggers the first *ARV* event to occur without delay. The *ARV* event adds the arriving customer to the queue ($Q = Q + 1$), and unconditionally schedules the next *ARV* event to occur t_a time units later. If the server is idle ($S > 0$), the arriving customer also triggers a *BEG* event to occur immediately. An occurrence of the *BEG* event removes a customer from the queue ($Q = Q - 1$), sets the server to busy ($S = 0$), and unconditionally schedules an *END* event to occur t_s time units later. The *END* event frees the server ($S = 1$) and, if possible ($Q > 0$), initiates service on the next customer in line by triggering the next *BEG* event to occur without delay.

Persons familiar with commercial process simulation languages might think of event graphs as having a single, generic model building block as opposed to many special-purpose process symbols. At the same time, event graphs combine the advantages of process modeling with the more general and efficient but abstract event-oriented approach.

4 SIGMA

SIGMA (Schruben 1991) is a commercial software implementation of the event graph formalism for DOS-based microcomputers. It provides a graphical windowing interface for interactive input, editing, and execution of event graph models. The structure of the graph is established by positioning and interconnecting nodes on the screen using a mouse-controlled cursor. The associated state changes, triggering conditions, and time delays are entered by means of pop-up dialog boxes.

Several concessions have been made to more practical aspects involved in building and executing simulation models by embellishing the otherwise minimalist event graph formalism with such features as built-in random variate generation, priority queues, output display and analysis capabilities, and links to user-defined events coded in the C language. The program also provides several means for intervening during the execution of a model to display the values of state variables or the status of the future events list.

One of the most interesting additions is a feature that allows the modeler to automatically generate an equivalent C or Pascal programming language implementation of an event graph model. In addition to providing a means for porting a model to other platforms, the well-structured and commented code that is produced is useful for conveying the structure underlying the implementation of DEDS models. The same facility has been extended to include a self-documenting English language translation of the model.

5 CONCLUSION

Several recent simulation textbooks (Hoover and Perry 1989), (Law and Kelton 1991), (Pegden et al. 1990) include limited material on event graphs. It is the author's opinion that the formalism should play a more central role in the instruction process and that language specific courses are more naturally the domain of commercial software vendors. This is not to say that traditional commercial simulation software does not have a place in university courses. In fact the roles of the two can be complementary when event graphs are used to help introduce the ideas behind more complicated commercial simulation software. The goal should be to make the modeling process easier to understand so that more class time can be spent on other crucial elements of the simulation process such as validation, experimentation, analysis, and communication of results.

REFERENCES

Conway, R., W. Maxwell, J. McClain, and S. Worona. 1987. *User's Guide to XCELL+.* The Scientific Press.

Gilman, A.R., and C. Gillingham. 1989. A Tutorial on See Why and Witness. *Proceedings of the 1989 Winter Simulation Conference:* 192-200.

Harrell, C.R., and K. Tumay. 1992. ProModel Tutorial. *Proceedings of the 1992 Winter Simulation Conference:* 405-410.

Hoover, S.V, and R.F. Perry. 1989. *Simulation: A Problem Solving Approach.* Addison-Wesley.

Law, A., and W.D. Kelton. 1991. *Simulation Modeling and Analysis (2nd Ed.).* McGraw-Hill.

Pegden, C.D., R.E. Shannon, and R.P. Sadowski. 1990. *Introduction to Simulation Using SIMAN.* McGraw-Hill.

Pegden, C.D., and D.A. Davis. 1992. Arena: A SIMAN/Cinema-Based hierarchical Modeling System. *Proceedings of the 1992 Winter Simulation Conference:* 390-399.

Pegden, C.D. 1988. *Introduction to SIMAN (2nd Ed.).* Systems Modeling Corp.

Pritsker, A.A.B. 1986. *Introduction to Simulation and SLAM II (3rd Ed.).* Wiley and Systems Publishing.

Russell, E. C. 1981. *Building Simulation Models with SIMSCRIPT II.5.* CACI, Inc.

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

Schruben, L.W. 1991. *SIGMA: Graphical Simulation Modeling.* The Scientific Press.

Schruben, L.W., and E. Yucesan, 1988. Simulation Graphs. *Proceedings of the 1988 Winter Simulation Conference*: 738-745.

Schruben, L.W. 1983. Simulation Modeling with Event Graphs. *Communications of the A.C.M.* **26**: 957-963.

AUTHOR BIOGRAPHY

KEVIN J. HEALY is an assistant professor in the School of Industrial Engineering at Purdue University. He holds B.S. and M.S. degrees in Industrial Engineering from The Pennsylvania State University and a Ph.D. in Operations Research from Cornell University. He has taught numerous simulation courses at both the undergraduate and graduate level as well as short courses to practitioners of simulation from industry, government and academia. In a former life, he was Vice-President of the Systems Modeling Corporation and a key developer of the SIMAN and Cinema simulation software. His research interests include stochastic optimization, simulation output analysis, and modeling formalisms.