Proceedings of the 1994 Winter Simulation Conference
ed. J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila

TRANSFORMING PETRI NETS INTO EVENT GRAPH MODELS

Lee Schruben
Cornell
U. S. A.

Enver Yucesan
INSEAD
FRANCE

ABSTRACT

Stochastic Petri Nets and simulation Event Graph models both have attractive graphical representations and simple rules that govern their dynamic behavior. A mapping of Stochastic Petri Nets into Event Graph models is presented and discussed. This mapping can be used to develop simulations of Petri Nets that exploit the efficiencies of the event-scheduling paradigm. It also permits the application of some of the rich analytical methodologies in the Petri Net literature to the analysis of event-oriented simulation models. Indeed, these two graphical representations of discrete event dynamic systems work in a complementary manner. We first present the structural and behavioral properties of standard Stochastic Petri Nets and Event Graph models and then discuss their relationship.

1. TERMINOLOGY AND NOTATION

Unfortunately, the monikers "event graph" and "simulation graph" have been widely used to describe different types of modeling paradigms. For example, in [Baccelli, et. al.] event graphs are a restricted class of Petri Nets and in [Törn] simulation graphs are an enriched class of Petri Nets. Here we refer to the Event Graph models described in [Askin and Standridge], [Hoover and Perry], [Law and Kelton], [Pegden], [Sargent], [Som and Sargent], [Schruben], [Yucesan], [Yucesan and Schruben], and others.

In this article, we discuss two types of networks that are used to model discrete event dynamic systems. The underlying structure of these networks are directed graphs. Graphs are typically pictured using icons such as balls as the vertices and arrows representing the directed edges. There is a rich mathematical theory of graphs much of which applies here; however, the intent of this article is to be expository rather than rigorous so pictures of graphs will be used wherever possible. The mathematics behind these pictures will be presented in a companion paper.

We will use the convention from the C programming language. In particular, the C shorthand X++ and X-- will be used to denote the statements X=X+1 and X=X-1 respectively. A double ampersand (&&) and a double bar (| |) are used for Boolean AND and OR operators. For Boolean testing, zero-valued conditions are false, all other conditions are true.

2. STOCHASTIC PETRI NETS

2.1. Stochastic Petri Net Structure

The graph for a standard Petri Net consists of two types of vertices, called *places* and *transitions*. Places are only connected to transitions and transitions are only connected to places; such a graph is called bipartite. In the picture of this graph, place vertices are depicted using balls and transition vertices are depicted using bars. Since we are dealing only with graphs where the edges have a direction, we can refer to the input and output places for a transition vertex and the input and output transitions for a place vertex.

The places in a Petri Net are sometimes occupied by one or more *tokens* that depict the state of the system being modeled. Pictorially a token is often a dot; however, we will sometimes need to use a number to represent the count of tokens in a place (a marking of ∞ represents an infinite source of tokens).

In order to have the modeling power to represent interesting discrete event dynamic systems, we will need to enrich the standard Petri Net to include (possibly random) transition firing times. Unless the transition is instantaneous, a time (usually a random variable) will be appended to each transition representing the time it takes to fire. Such Petri Nets are called Stochastic Petri Nets.

2.2. Stochastic Petri Net Behavior

The rule governing the behavior of a standard Petri Net is simple: whenever each input place for a transition vertex is occupied by at least one token, the transition *fires,* causing one token to be removed from each input place and deposited in each output place of the transition. When the transition firing is not instantaneous, the tokens are removed from the input places when the firing starts and placed in the output places when the firing ends.

2.3. An Example

Our example with be a single line G/G/N queue that will process 100 customers. The customer interarrival times will be successive values of the random variable t_a, the service times are given by the random variable t_s, and N is the number of servers. This model in its initial state of having the queue empty and all of the servers idle is depicted in Figure 1. Places are labeled with capital letters and transitions with lower-case letters. Place labels will also be used as variables whose values are the token counts for the corresponding places. Edges have labels that might represent transition functions specifying the number (and perhaps color) of tokens that are removed or deposited when a transition fires. For this example, edge labels are not necessary but will be referred to later.

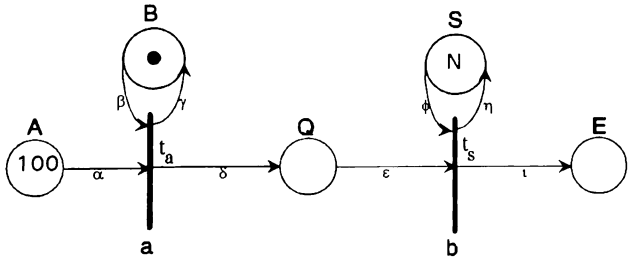

Figure 1: A Stochastic Petri Net for a G/G/N Queue

Place A initially contains the 100 customer arrivals; place B blocks customers from arriving more than one at a time; place Q is the queue where arriving customers wait for service; place S is where idle servers wait for work, and place E counts the number of customers that exit the system. This initial state is reflected by the values of state variables being A=100, B=1, Q=0, S=N, and E=0. Transition 'a' models successive customers entering the system and transition 'b' models customers being serviced.

Notice each place in the Petri Net in Figure 1 has exactly one input and output arc. [Baccelli, et. al.] refer to this special subclass of Petri Nets as an "event graph"; this should not to be confused with the Event Graph models discussed in the next section.

Typical of Petri Net models, this model uses a resident-entity paradigm, where counts of waiting customers and idle servers are dynamically maintained. See [Schruben, 1994] for a discussion of the relative advantages and disadvantages of resident-entity and transient-entity (also know as process) modeling.

3. EVENT GRAPH MODELS

3.1. Event Graph Structure

Event Graph models can be used to build models of any discrete event system using just a single graphical object [Yucesan]. Event Graphs are fundamentally different from state diagrams and automata. In state diagrams, vertices are used to represent the *values* of the system state. In an Event Graph model, vertices represent *changes* in the system state. Event Graphs are similar to sets of differential equations used to specify the dynamic behavior of continuous state models by describing how states change. A conventional state diagram for our G/G/N queue requires an infinite graph, the event graph for this system is not only finite but can actually be reduced to a single vertex for a Markov system [Schruben].

State changes are associated with the occurrence of a system event (such as the arrival of a customer to a queueing system) and are pictured as event vertices. The edge between two event vertices represents the conditions under which one event might cause the occurrence of the other event as well as the time interval between the two events. Associated with each edge may be a set of *conditions* that must be true in order for one event to schedule another. Also associated with each edge may be a *delay time* equal to the interval until the scheduled event occurs. The graphical representation of the basic modeling object is as follows;

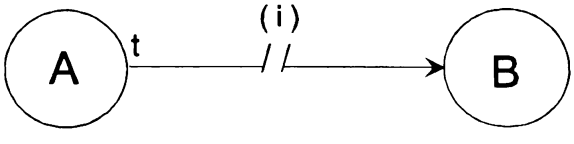

Figure 2: The Basic Edge for an Event Graph

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, and the edge can be ignored until the next time event A

occurs. You can think of an edge as nonexistent or broken unless its edge condition is true. If the condition for an edge is always true, the condition is left off the graph. We will call edges with conditions that are always true *unconditional* edges. Zero time delays for edges are not shown on the graph. Note that we use the suggestive symbol (//) for an edge condition like [Askin and Standridge] rather than the tilde in [Schruben].

3.2. Event Graph Behavior

An Event Graph is executed by a main control program that operates on a master appointment list of scheduled events. This list is called the *future events list* and contains all of the events that are scheduled to occur in the future.

The main control program is the same used in most discrete event simulation codes. The control program

1. advances time to the next event,
2. executes the state change for the event, and
3. schedules events where exiting edges are true.

Once this event has finished executing, the event is removed from the future events list. The control program then returns to step 1 and will again advance time to the next scheduled event and execute the corresponding event procedure. The simulation operates in this way, successively calling and executing the next scheduled event procedure until some condition for stopping the simulation run is met or there are no more scheduled events.

3.3. An Example

To model our G/G/N queue we will use the simulation Event Graph in Figure 3. Using the same notation as in the Petri Net of Figure 1, S is the number of idle servers and Q is the number of customers waiting for service. The "RUN" vertex simply initializes the model with the initial system state of N idle servers and a calling population of A customers.

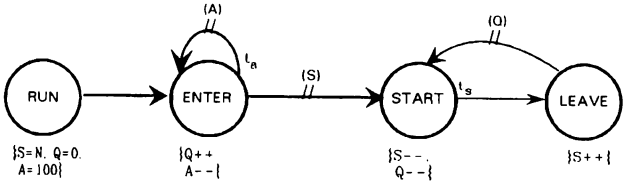

Figure 3: An Event Graph of a G/G/N Queue

Event vertices carry descriptive labels and state changes associated with each event are enclosed in braces below the respective vertices. When the meaning is clear, we will sometimes label the vertices with their state change.

An Event Graph model can be read by simply describing each edge in the graph. There is a single edge in the graph for each of the following sentences.

At the start of the simulation RUN, the first customer will ENTER the system. Successive customers ENTER the system every t_a *minutes. If an ENTERing customer finds an idle server* (S>0), *they START service immediately. Customers who START service can LEAVE after a service delay of* t_s *minutes. Whenever a customer LEAVEs and the queue is not empty* (Q>0), *the server will START with the next customer.*

This graph represents a completely defined simulation model once we specify a rule for breaking time ties for simultaneously scheduled events.

It is worth noting that while there is no universally accepted definition of an "event", the customary notion of a system event will typically correspond to a subgraph of event vertices connected by edges with zero delay (a particular vertex may belong to more than one such system event). As mentioned earlier, if the arrival and service times for the above queueing system are exponentially distributed, the event graph can be collapsed into a single vertex representing the departure of a customer (i.e., representing the embedded Markov chain at customer departure times).

4. EXTENSIONS

There are numerous extensions of the standard Stochastic Petri Net model defined here as well as to our definition of a Event Graph model. It is interesting to note that the enrichments of Stochastic Petri Nets have been motivated primarily by a need to increase their modeling power [Baccelli, et. al.]. On the other hand, the enrichments of Event Graph models have been motivated by a desire for better analytical tools for these models [Sargent and Som].

Extensions to Stochastic Petri Nets include inhibitor arcs, randomized markings, and colored tokens. An inhibitor arc will prevent a transition from firing if the input place is marked with a token. Randomized markings will move tokens in output places according to some probability law.

In Colored Petri Nets, tokens of different "colors" represent different classes of objects (e.g.: different types of parts being produced in a factory or different types of messages in a communication system). The colors of tokens that may occupy each place is specified.

Also specified are the numbers and colors of tokens in each input place required for a transition to fire. There are transition input (output) functions that determine the numbers and colors of tokens removed from (deposited in) input (output) places when a transition fires. There are many other enrichments of Petri Nets, some that are specifically designed for discrete event simulation [Torn, 1990].

Extensions of standard Event Graph models include passing parameter values between events and the introduction of an event cancelling edge. Neither of these extensions are believed to actually increase the modeling power of Event Graph models but certainly make it easier to represent certain types of systems [Yucesan]. Parameterized vertices and edge attributes are particularly significant in that they permit a basic event graph to be used to represent different instances of similar subsystems. These graphs can be linked into a model of a larger system. For example: an Event Graph of a single generic machine cell can be parameterized to represent different types of cells that simulate a large factory.

5. MAPPING STOCHASTIC PETRI NETS INTO EVENT GRAPHS

In the mapping of a Stochastic Petri Net into an Event Graph model, edges become vertices and vertices become edges. For simplicity, the mapping presented here is specific to the subclass of Stochastic Petri Nets where each place has a single entering and exiting edge (also called an "event graph" Stochastic Petri Net in [Baccelli, et. al.]).

5.1 The Mapping

Specifically, all incoming edges to a Stochastic Petri Net transition vertex become a single "start transition" vertex for the Event Graph model. Similarly, all outgoing edges from a transition of a Stochastic Petri Net become a single "end transition" vertex in a Event Graph model. All Stochastic Petri Net transition vertices and place vertices become edges in the Event Graph model. A transition vertex will become an unconditional edge with an edge delay equal to the transition firing time. A place vertex will be come a zero-delay edge conditioned by the token counts of all places preceding the next transition. Finally, a single "Run" vertex is appended to the event graph that specifies the initial state of the system.

5.2 Some Observations

As there are two types of vertices in a Stochastic Petri Net, there will be two types of edges in the corresponding Event Graph model. Transition vertices of the Stochastic Petri Net correspond to unconditional edges (usually with a delay) in the Event Graph model, and place vertices in the Stochastic Petri Net become (usually conditional) zero-delay edges in the Event Graph model. The Event Graph model is "edge bipartite" in that no two conditional or delayed edges are separated by only a single vertex.

The mapping that we present here often produces vertices corresponding to elements that are needed for the Petri Net paradigm but superfluous for the Event Graph model. We can reduce such Event Graph models by removing unnecessary events using some of the rules in [Yucesan and Schruben] and the procedure in [Sargent and Som].

5.3. Our G/G/N Example

We start with the Stochastic Petri Net in Figure 1. By making all edges entering and exiting each transition into vertices and all Stochastic Petri Net vertices into edges, we have the Event Graph model in Figure 4. The token counts for places B and E are never tested as false on any edge, so the are omitted from the graph in Figure 4. Here we label each vertex with their corresponding edges in the Stochastic Petri Net of Figure 1.

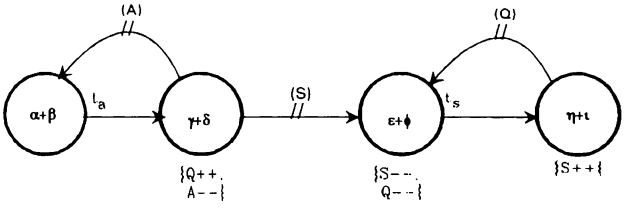

Figure 4: Mapping of our Stochastic Petri Net into an Event Graph Model

Eliminating the null event at the far right of Figure 4 and adding the RUN initialization vertex completes the transformation from Figure 1 into Figure 3.

5.4. An Example using Inhibitor Arcs

An example of a Stochastic Petri Net with an inhibitor arc is a model of the failure and repair process of three machines shown in Figure 5. Here failure and repair times are subscripted with f and r respectively.

The inhibitor arc has a small circle near its head. The current marking of this net with tokens indicates that two machines are working (place D) and one machine is under repair (place C). The marking shown is for a machine just about to fail. The timed transition at the far left models the generation of machine failures. Here it is necessary to assume that a failure may "queue" in

place B if all machines are broken when the failure occurs. This is fine if the times between failures have an exponential distribution. Figure 5 shows many of the basic graphical objects in a Stochastic Petri Net: timed and instantaneous transitions, multiple tokens, and an inhibitor arc.

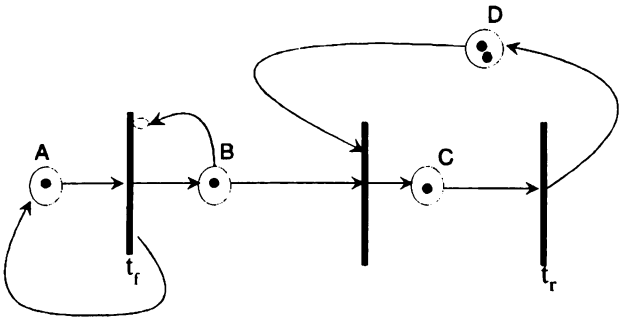

Figure 5: Stochastic Petri Net Machine Failure Model

The Event Graph model resulting from the transformation specified in section 5.1 appears in Figure 6. Again, the variables A, B, C, and D in the Event Graph of Figure 6 are the token counts of the four places in the Petri Net model in Figure 5. The model is initialized (with 3 working machines). Note that A will be equal to -1 if a machine failure has to "queue" for a working machine. Recall that we are using standard C notation for the state changes and edge conditions (the edge condition (A) means that place A is marked, (!B) means that B is not marked). The inhibitor arc corresponding to place B in the Petri Net behaves like any other event scheduling arc except a "not" (!) logical operator is attached to its token count in the Event Graph.

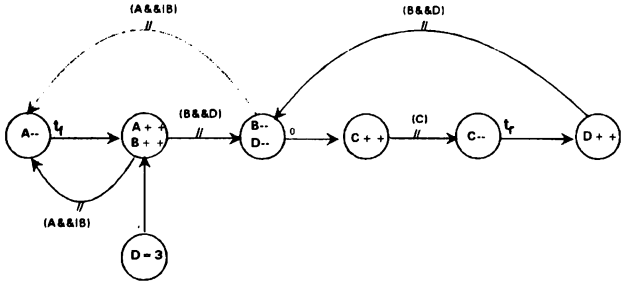

Figure 6: Event Graph of the Petri Net in Figure 5

The Event Graph in Figure 6 can be reduced to the simple event graph in Figure 7 by eliminating edges that can never be true and null vertices. A reduced Event Graph for any number of machines, N, is shown in Figure 7.

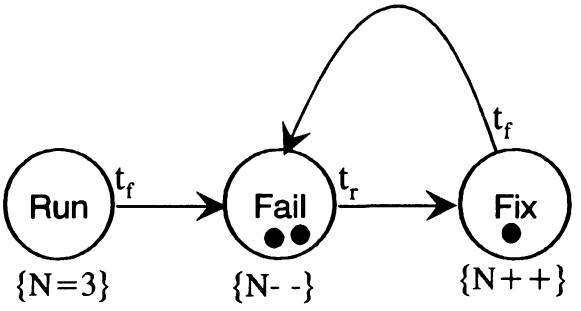

Figure 7: Event Graph Model for Machine Failures

For the Event Graph model in Figure 7, the machine failure times are not assumed to be exponential. Obvious advantages of the Event Graph representation in Figure 7 over the Petri Net in Figure 5 are its generality and simplicity.

5.5 Event Tokens

To show the state of the future events list for an event graph, we can mark each vertex with *event tokens* that indicate how many instances of each event are scheduled to occur in the future. The total count of the tokens gives the number of events that are currently scheduled to occur in the future.

The marking rule for event tokens is simple: after an event vertex is executed (it "times out"), a token is removed from that vertex and tokens are deposited in all successor vertices connected by edges with true conditions. To completely specify the state of the future event calendar, the event tokens could also carry labels with the times that the corresponding events are scheduled.

In Figure 7 there are 3 machines, two are working (waiting for their next scheduled failures) and one is broken (waiting for its next scheduled repair). The RUN initiation vertex has already been executed and removed from the list of scheduled events.

6. DISCUSSION

A simple mapping of a class of Stochastic Petri Nets into Event Graph models has been presented. There is an enormous literature on Petri Nets while the development of Simulation Event Graph models has just begun. Event Graph models have become widely used only in the United States, primarily for education. There are certain advantages to Event Graph models in terms of simplicity and efficiency in simulation and research. It is hoped that this article encourages persons familiar with Petri Net modeling to become acquainted with Event Graph modeling and vice versa.

ACKNOWLEDGMENT

I am grateful to the National Science Foundation for sponsoring a research project of which this paper is a part.

REFERENCES

Askin, R. G. and C. R. Standridge, (1993). *Modeling and Analysis of Manufacturing Systems,* John Wiley and Sons.

Baccelli, F.,G. Cohen,G.L. Olsder, and J.-P. Quadrant, 1992, Synchronization and Linearity: An Algebra for Discrete Event Systems, J. Wiley and Sons, Chichester, England.

Hoover, S. and R. Perry, (1990). *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., (1986). *Introduction to SIMAN, 2nd ed.*, Systems Modeling Corp.

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

Sargent, R. G., (1988). "Event Graph Modeling for Simulation with an Application to Flexible Manufacturing Systems." *Management Science* 24(10), 1231-1351.

Schruben, L. (1983). "Simulation Modeling with Event Graph Models." *Communications of the Association of Computing Machinery* 26(11), 957-963.

Schruben, L., and E. Yucesan, (1988). "Duality in Simulation Graphs." *Proc. 1989 Winter Simulation Conference,* Washington D.C.

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.

Törn, 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., (1993). "On the Modeling Power of Simulation Graphs", Technical Report, INSEAD, Fontainebleau, France.

Yucesan, E. and L. W. Schruben, (1992). "Structural and Behavioral Equivalence of Simulation Models." *ACM Transactions on Modeling and Computer Simulation* 2(1).

AUTHOR BIOGRAPHIES

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. He is also one of the developers of the SIGMA simulation modeling and analysis system (the 3rd edition has recently been published in The Scientific Press series from boyd&frasier).

ENVER YUCESAN is on the faculty at INSEAD in Fontaineblau, France. He received his undergraduate degree from Purdue University and his PhD from Cornell. His research interests include simulation modeling paradigms and structures and the design and analysis of simulation experiments. He has also been involved in various application areas. Recently his wife, Jae, and he had their first child, Elliot.

Prof. Lee Schruben
School of Operations Research
and Industrial Engineering
Room 219 Engineering Theory Center
Cornell University
Ithaca, NY 14853-3801
phone: (607) 255-9133.
Fax: (607) 255-9129
e-mail: lee@orie.cornell.edu

Prof. Enver Yucesan
INSEAD
Boulevard de Constance
77305 Fontainebleau Cedex
FRANCE
phone: (33) 1 60 72 40 00
Fax: 91 011 (33) 1 60 72 40 49
e-mail: yucesan%FRE1BA51.bitnet@cunyvm.cuny.edu