Proceedings of the 1996 Winter Simulation Conference
ed. J. M. Charnes, D. J. Morrice, D. T. Brunner, and J. J. Swain

ELIMINATING CANCELING EDGES FROM THE SIMULATION GRAPH MODEL METHODOLOGY

Ricki G. Ingalls

Manufacturing Strategy Group
Corporate Operations
Compaq Computer Corporation
20555 SH 249
Houston, TX 77070

Douglas J. Morrice

MSIS Department
The University of Texas at Austin
Austin, TX 78712
End to End Simulation Department
Schlumberger Austin Research
Austin, TX 78720

Andrew B. Whinston

MSIS Department
The University of Texas at Austin
Austin, TX 78712

ABSTRACT

Event Graphs and Simulation Graph Models provide a powerful and general modeling framework for discrete event simulation. Within this framework it has been shown that an event cancellation construct in the form of a canceling edge is a modeling convenience rather than a necessary modeling tool. As a result, very little work on the formal development of Event Graphs and Simulation Graph Models directly considers the canceling edge construct. However, to the simulation practitioner the modeling convenience and functionality provided by canceling edges is important. This is clearly demonstrated by the presence of event canceling edges in SIGMA, the commercial software implementation of Event Graphs. This paper proposes an extension of the Event Graph methodology that will completely eliminate canceling edges without loosing the functionality that canceling edges provide. Since this extension is formally developed within the Simulation Graph Model framework, it provides a general approach to handling event cancellation.

1 INTRODUCTION

Event Graphs (EGs) have included event cancellation constructs in the form of event canceling edges since their introduction (Schruben 1983). Subsequent papers also include the canceling edge construct (Sargent 1988, Som and Sargent 1989). After EGs were formalized and extended into Simulation Graphs (SGs) and Simulation Graph Models (SGMs), it was shown theoretically that event canceling edges were a modeling convenience but not a necessary modeling tool (Yücesan 1989, Yücesan and Schruben 1992). As a result, very little of the formal development work on EGs and SGMs directly addresses the issue of event cancellation. Recently, Savage and Schruben (1995) demonstrated that it is practically possible to model without event canceling edges.

One might conclude that event canceling edges, in particular, and event cancellation, in general, should be eliminated from the EG methodology. However, simulation practitioners clearly find the modeling convenience and functionality provided by event canceling edges, beneficial. This is demonstrated by the fact that the commercial software implementation of EGs, SIGMA (Schruben 1995, page 75) includes the event canceling edge construct.

Since event canceling has some benefit in terms of modeling convenience this paper does not advocate its elimination altogether. Instead, the event canceling edge construct is eliminated and replaced by a more general event canceling construct in the form of an edge execution condition. Essentially, the edge execution condition is tested at the moment the event is scheduled to be executed. If the condition is true, the event is executed; if the condition is false the event is discarded without execution. The edge execution condition retains the functionality and modeling convenience of canceling edge.

This paper extends EGs to include the edge execution condition. In addition, this construct is formally developed within the SG framework to provide an extension to SGMs. Its generality is demonstrated within this framework. The complete simulation execution sequence of the SGM extension is provided in algorithmic form. The results are demonstrated on an example from Savage and Schruben (1995).

The remainder of the paper is organized as follows. Section 2 provides a description of EGs and SGMs. Section 3 introduces the new EG construct that includes an edge execution condition, extends SGMs to include this new construct, and provides the algorithm that demonstrates the complete simulation execution sequence. Section 4 demonstrates the methodology on an example from Savage and Schruben (1995). The example is described using the canceling edge approach, the approach developed by Savage and Schruben (1995), and the approach developed in this

paper. Section 5 provides a preliminary run-time comparison of computation effort required by the three approaches on the example provided in Section 4. Section 6 contains some concluding remarks.

2 ORIGINAL IMPLEMENTATION

Figure 1 depicts the basic EG construct. This construct is called a *scheduling edge*. The nodes labeled A and B represent events. The edge specifies that there is a relationship between the two events. More specifically, the construct can be interpreted as follows "if condition *(i)* is true at the instant event A occurs, then event *B* will be scheduled to occur *t* time units later" (Schruben 1983). The quantity *t* may assume the value zero, in which case *B* happens at the same instant as *A*. Note that it is possible (and often necessary) to specify an edge with no condition. The scheduling edge schedules the event vertex at the head of the edge in accordance with the time specification with the edge

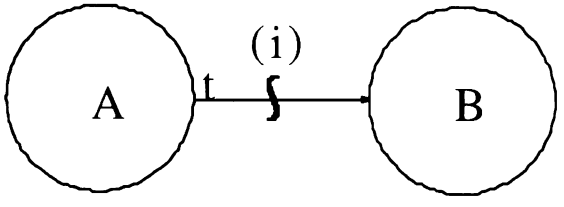

Figure 1: Scheduling Edge

Figure 2 depicts a second basic construct in EGs referred to as a *canceling edge*. The canceling edge has the following interpretation: "if condition *(i)* is true at the instant event A occurs, then the currently scheduled event *B* will be canceled *t* time units later" (Schruben 1983). In most instances, *t* equals zero for the canceling edge (Schruben 1995, page 76). The canceling edge is used to delete scheduled transactions that execute the vertex that is at the head of the edge. The canceling edge was added to the event graph framework to handle disruptions such as machine breakdowns.

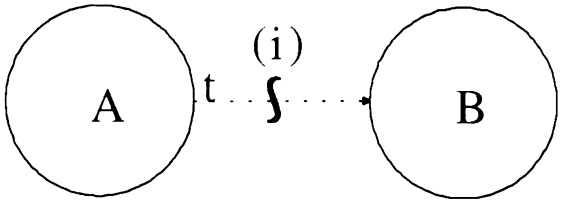

Figure 2: Canceling Edge

In the event graph, it is also possible to parameterize the event vertices and thus extend the basic constructs in Figures 1 and 2. Parameterization is accomplished through vertex parameters and edge attributes. A vertex parameter list is a string of state variables associated with a particular vertex. An edge attribute list is a string of expressions associated with a particular edge. These lists are used in scheduling or canceling specific instances of events. Using these parameters is analogous to passing values in subroutines in high-level programming languages. For example, Figure 3 provides an extension of Figure 1. The construct in Figure 3 is interpreted as follows: "if condition *(i)* it true at the instant event A occurs, then event *B(j)* will be scheduled to occur *t* time units later with parameter string *j*, equal to *k*" (Schruben 1995, page 79).

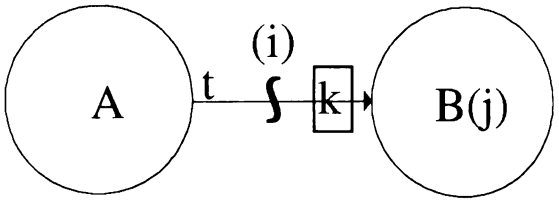

Figure 3: Vertex Parameters and Edge Attributes on a Scheduling Edge

Yücesan and Schruben (1992) and Schruben and Yücesan (1993) extend EGs to SGs and SGMs. This extension provides a formal graph theoretic framework for EGs. Within this framework, Yücesan (1989) shows that canceling edges and parameterization are modeling conveniences and do not augment the modeling capabilities of these graphs. As a result, very little of the formal development in SGs has focused directly on the canceling construct.

3 THE EDGE EXECUTION CONDITION

The basic construct in the modified event graph framework is shown in Figure 4. As before, the nodes labeled A and B represent events and the edge specifies that there is a relationship between the two events. However, the construct is now interpreted as follows: "if condition (i) is true at the instant event A occurs, then event B will be scheduled to occur *t* time units later. Event B will be executed *t* time units later with the state variables in array n set equal to the values in array k if condition (j) is true *t* time units later." This simple modification is the basis for eliminating canceling edges in the simulation graph methodology.

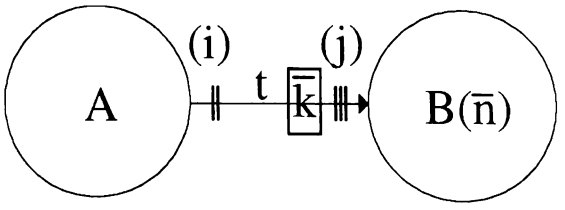

Figure 4: Scheduling Edge with an Execution Condition

3.1 Simulation Graphs Extension

Using the notation of Yücesan and Schruben (1992) and Schruben and Yücesan (1993), let $G=(V(G),E(G),\psi_G)$ be a directed graph where $V(G)$ is the set of vertices and $E(G)$ is the set of edges. G can be a multi-arc graph, meaning that more than one edge can connect the same two vertices flowing in the same direction. Vertices and edges can have functions and attributes assigned to them.

The function that is assigned to the vertices is:

(i) $\mathcal{F}=\{f_v : v \in V(G)\}$, the set of state transitions functions associated with vertex v.

The functions assigned to the arcs are:

(i) $C=\{C_e : e \in E(G)\}$, the set of *scheduling edge conditions*.

(ii) $\mathcal{X}=\{X_e : e \in E(G)\}$, the set of *execution edge conditions*.

(iii) $\mathcal{T}=\{t_e : e \in E(G)\}$, the set of *edge delay times*, and

(iv) $\Gamma=\{\gamma_e : e \in E(G)\}$, the set of *event execution priorities*.

With the modifications described above, the Simulation Graph Model (SGM) is now defined as $\mathcal{L}=\{\mathcal{F},C,\mathcal{X},\mathcal{T},\Gamma,G\}$.

In contrast to Schruben and Yücesan (1993), there are no canceling edges in the graph. Instead, the set $\mathcal{X}$, which is evaluated at the time that the event comes off the calendar to be executed, replaces the function of the canceling edge. If the condition X_v is true, then the vertex associated with the event notice is executed. If X_v is false, then the event notice is discarded. This implementation provides a precise way for canceling scheduled events in the methodology and in practice.

Although Schruben and Yücesan (1993) specifically detail the function of canceling edges in their terminology, their canceling edge step is not straightforward. The function of canceling edges is to search for and remove one or more *already scheduled* event notices from the event calendar, $\mathcal{L}$. The edge execution edge conditions permits the same functionality, i.e., the ability to not execute an event notice already scheduled on the calendar, without requiring a search of the events calendar. The edge execution approach does require some extra overhead associated with carrying more information on the event calendar and perhaps the need for additional state variables. However, in many instances, the overhead can be justified if it eliminates a costly search procedure.

3.2 Simulation Graph Execution Sequence

As with the framework in Schruben and Yücesan (1993), this framework also needs additional mechanisms to facilitate the execution of the model. The symbol τ will be used to represent the global simulation clock and $\mathcal{L}$, the event calendar. However, our definition of $\mathcal{L}$ will be different from the one specified in Schruben and Yücesan (1993). Our definition of $\mathcal{L}$ is an ordered set, $\mathcal{L}=\{(t_1,\gamma_1,v_1,e_1,a_1),(t_2,\gamma_2,v_2,e_2,a_2),\ldots\}$ where t_i represents the execution time, γ_i represents the execution priority, v_i is the vertex to be executed, e_i is the index of the edge that is being scheduled, and a_i are the values of the edge attributes. Each (t,γ,v,e,a) is an *event notice*. It is important to note that Schruben and Yücesan (1993) did not include the values of the edge attributes as part of the event notices. Edge attributes are included here since the parameter passing mechanism is intended to pass the values of the variables and not the references.

We also define the following sets:

(i) S_v : the set of state variables that can be changed at vertex v. (Since S is the set of all state variables, $S_v \subset S$).

(ii) P_v : the set of state variables in the parameter list of vertex v.

(iii) Φ_e : the set of state variables involved in the scheduling conditions on edge e.

(iv) ϑ_e : the set of state variables involved in the execution conditions on edge e.

(v) A_e : the set of state variables used to determine the values of a_e on edge e.

In addition to these sets, there is one conditional statement, ω, which is evaluated to determine if the simulation should stop. With these definitions, the execution of the Simulation Graph model is carried out as follows.

Initialize (Run Initialization)

Step 1. Initialize the global simulation clock. $\tau \leftarrow 0$.

Step 2. Insert one or more event notices into the event calendar: For simplicity, we will assume that each of these event notices could be executed at time 0. $\mathcal{L} = \mathcal{L} \cup \{(0, \gamma_1, v_1, e_1, a_1), (0, \gamma_2, v_2, e_2, a_2), \ldots\}$.

Execute (execution of the model implementation)

Step 1. Remove the first event notice from $\mathcal{L}$. $\mathcal{L} = \mathcal{L} \setminus \{\ell \mid \ell = (t_1, \gamma_1, v_1, e_1, a_1)\}$. Event notice ℓ is removed from the calendar. This implies $t_t \equiv t_1, \gamma_t \equiv \gamma_1, v_t \equiv v_1, e_t \equiv e_1$, *and* $a_t \equiv a_1$.

Step 2. If the execution edge condition, $X_{e_t}(\vartheta_{e_t}, a_t) = FALSE$, then go to Step 1 of execute.

Step 3. Update the simulation clock. $\tau \leftarrow t_t$.

Step 4. Assign the attributes to the parameters of the vertex. $P_{v_t} \leftarrow a_t$. If Y is the i^{th} state variable in the vertex parameter list, i.e. $(P_v)_i = Y$, then $Y \leftarrow (a_t)_i$.

Step 5. Evaluate the state change. $S_{v_t} \leftarrow f_{v_t}(S)$.

Step 6. Schedule further events. For each edge, e_{ij}, emanating from v_t, if $C_{e_{ij}}(\Phi_{e_{ij}}) = TRUE$ then evaluate $A_{e_{ij}}$ and assign the attribute value of the new event notice, k, $a_k \leftarrow A_{e_{ij}}$. Generate the inter-event time, $t_k = f(t_{e_{ij}})$, and schedule the event notice where $\mathcal{L} = \mathcal{L} \cup \{(\tau + t_k, \gamma_{e_{ij}}, v_j, e_{ij}, a_k)\}$.

Step 7. Terminate the execution of the simulation if any of the following conditions is satisfied:

(i) τ exceeds T_{stop}.

(ii) The simulation stopping condition, ω, evaluates TRUE.

(iii) $\mathcal{L}$ is empty.

Otherwise, go to Step 1 of Execute.

4 EXAMPLE

Savage and Schruben (1995) present an example in which they demonstrate a procedure to model canceling edges out of EGs. The same example will be used to demonstrate the edge execution condition approach.

The example is one of a single server system that has two types of customers. Of the two types of customers, customer type 1, has the highest priority. If a type 2 customer is being served when a type 1 customer arrives, the type 2 customer is preempted by the type 1 customer and will wait until all type 1 customers have been served. Figure 5 shows how this system can be modeled with a canceling edge (Savage and Schruben 1995).

4.1 With a Canceling Edge

For this system, the state variables are:

Q1 and Q2	The number of customers in each queue.
S	Server Status: 1 = available, 0 = busy, -1 = preempted

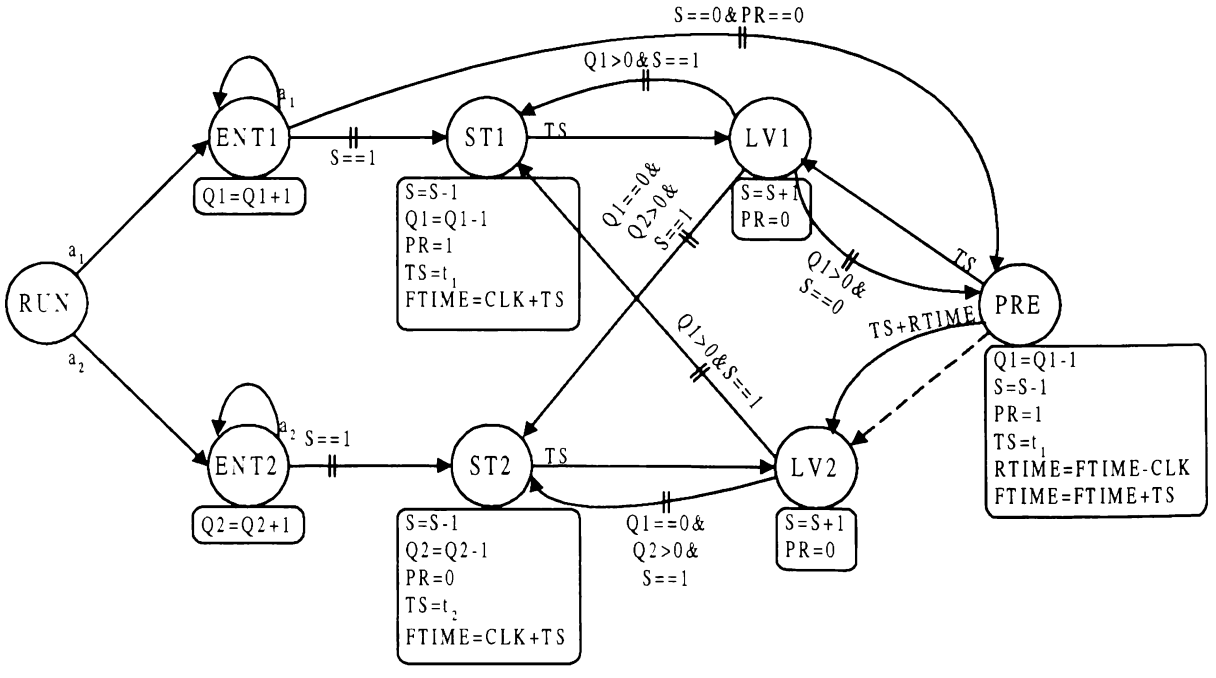

Figure 5: Single Server Preemptive System with a Canceling Edge (Savage and Schruben 1995)

PR	If =1, then a priority customer is being served.
TS	The length of time for service.
FTIME	The finishing time of service (if not preempted).
RTIME	The remaining time in the service.

In this system, each type of customer has its own arrival distribution, a_i. When a type 1 customer enters the system (ENT1), if there is a server available, the customer starts his service (ST1) and then finishes his service (LV1) after TS time has elapsed. When a type 2 customer enters the system (ENT2) and a server is available, then the type 2 customer starts his service (ST2) and is *scheduled* to finish his service TS time units later. However, if a type 1 customer enters the system while the server is busy with a type 2 customer, the type 1 customer preempts (PRE) the type 2 customer by taking the type 2 customer off of the future events list if the event the type 2 customer is scheduled to execute is LV2. The type 1 customer then is served for TS time units and then finishes his service (LV1). At the same time, the type 2 customer is scheduled to finish his service TS+RTIME units later. When a type 1 customer finishes service, one of three things can occur. First, if a type 2 customer was preempted (S==0) and there is another type 1 customer in the queue (Q1>0), then the type 2 customer that was previously preempted is preempted again and service is given to the new type 1 customer. Second, if there are new type 1 customers in the queue (Q1>0) and there was not a previous preemption (S==1), then the service is given to the next type 1 customer in the queue. Third, if there are no type 1 customers (Q1==0) and there are type 2 customers (Q2>0) and the server is available (S==1), then the service is given to the first type 2 customer in the queue.

4.2 Without a Canceling Edge

Figure 6 contains the Savage and Schruben (1995) model without canceling edges. They eliminate canceling edges by putting a check event (CHK) in the model that will not allow a type 2 customer to leave the system (LV2) until no type 1 customers are in the system (S==0). The check event is processed whenever a type 1 customer is scheduled to leave the system. The actual execution of this model as intended is problematic due to the number of zero-time events and the lack of priorities on the arcs. Regardless, this method of eliminating the canceling edge should work with minor modifications.

There are three basic problems with this solution. The first is that the modeler, and not the methodology, is responsible for eliminating the canceling edge. In many instance, this could be a challenging modeling exercise by itself. The second problem is that there is additional overhead in terms of scheduling the check event. The third problem is more philosophical, but one

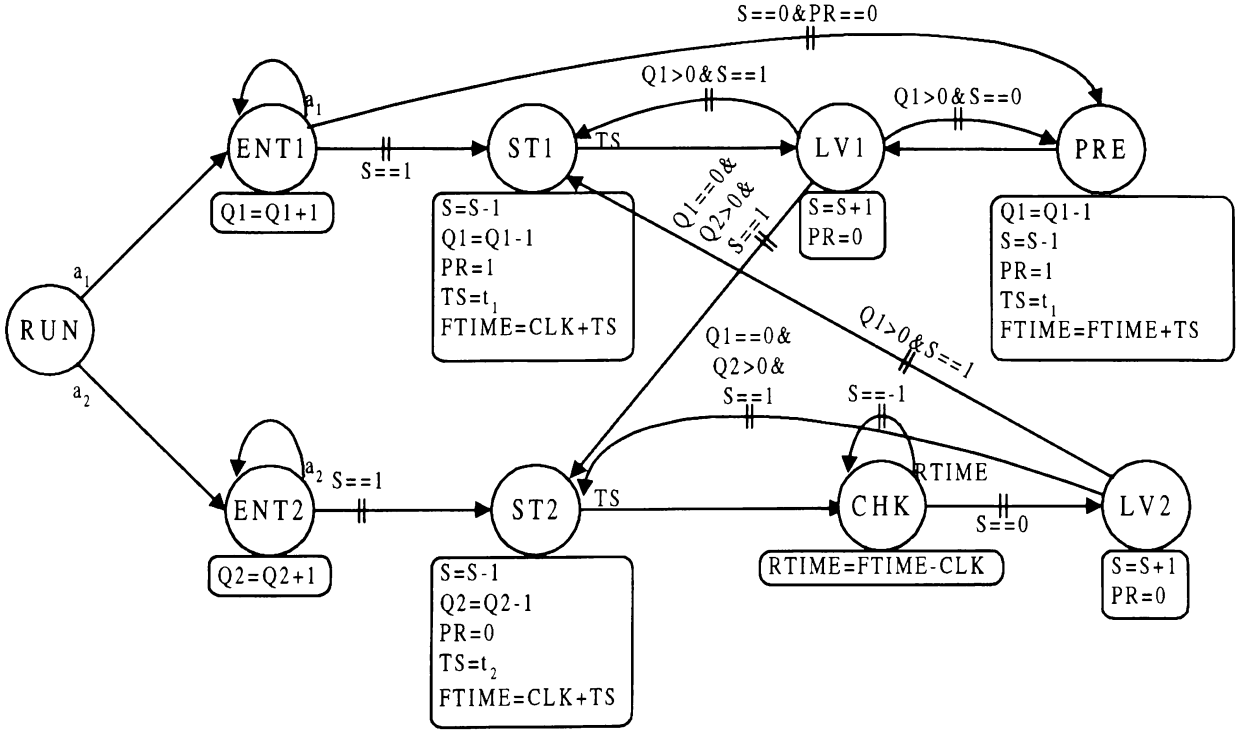

Figure 6: Single Server Preemptive System without a Canceling Edge (Savage and Schruben, 1995)

of the strengths of event graphs is that the events in the model usually correspond to events that occur in the actual system being modeled. A check event never actually occurs in the physical system.

4.3 With Execution Conditions

The model with execution conditions, as they are outlined in this paper, is very straightforward. Figure 7 shows the model as it would be with execution conditions.

In this system, the state variables are:

Q1 and Q2	The number of customers in each queue
S	Server status
P	The number of preemptions that have occurred
T	The type of customer being serviced
ST	The starting time in the service
RT	The remaining time in the service

In this system, each type of customer has its own arrival distribution, a_i. When a type 1 customer enters the system (ENT1), if there is a server available (S==1), the customer starts his service (ST1) and then finishes his service (LV1) after t_1 time has elapsed. When a type 2 customer enters the system (ENT2) and a server is available (S==1), then the type 2 customer starts his service (ST2) and is *scheduled* to finish his service TS time units later. When the event is scheduled, the number of preemptions that have occurred (P) is stored in attribute 1 (A(1)). When the event is scheduled to occur, if there have been no more preemptions (P==A(1)), then the type 2 customer leaves the system (LV2). However, if a type 1 customer enters the system while the server is busy (S==0) with a type 2 customer (T==2), the type 1 customer preempts (ST1P) the type 2 customer by incrementing P, the number of preemptions and storing the remaining service time for the type 2 customer (RT). This will not allow the type 2 customer to finish because P>A(1). In any case, when a type 1 customer finishes his service (LV1), then one of three things can occur. First, if there is a type 1 customer in the queue, then a new type 1 service is started. Second, if there is no one in the type 1 queue (Q1==0) and a type 2 customer has been preempted (RT>0), then the type 2 customer is restarted with completion scheduled for RT time units later. Again, the leave event (LV2) for the type 2 customer will only be executed if his is not preempted (P==A(1)). Third, if there is no one in the type 1 queue (Q1==0) and no one has been preempted (RT==0) and there is a type 2 customer waiting, then a new type 2 customer starts service (ST2).

5 RUN-TIME CONSIDERATIONS

In order to determine which of the three approaches is

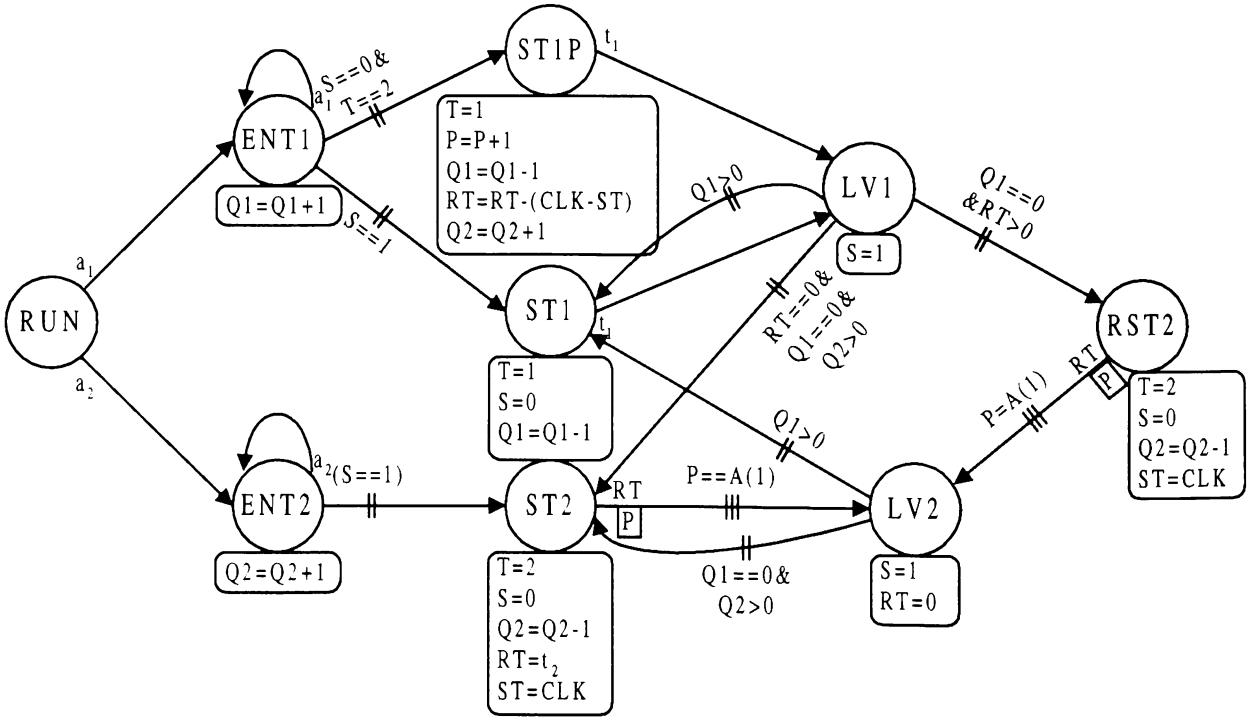

Figure 7: Single Server Preemptive System with Execution Conditions

more efficient, consider the run-time complexity of each of the three models. First, we define complexity as the number of times that an event is schedule or canceled since the management of the calendar is easily the most time consuming aspect of any discrete-event simulation.

To best gauge the run-time, let us pick a state that will be revisited, and thus could be called a cycle, in the simulation. Let us assume that the simulation is about to execute an ST2 event. If the simulation is about to execute an ST2 event, we know that there are no type 1 customers in the system. The complex part of this simulation is when type 1 customers preempt type 2 customers. Let us assume that n type 1 customers and m type 2 customers will enter the system before the next ST2 event is executed. Under these conditions, we can calculate the number of events that will executed between ST2 events. The different calculations on the number of events for each type of model are in Figure 8.

Events Scheduled or Canceled	**Number of Occurrences**
With Canceling Edge	
ST2	1
ENT2	m
ENT1,PRE,2*LV2,LV1	n
LV2	1
TOTAL	*5n+m+2*
Without Canceling Edge	
ST2,CHK	1
ENT2	m
ENT1,PRE,LV1,CHK	n
LV2	1
TOTAL	*4n+m+3*
With Execution Conditions	***(Worst Case)***
ST2	1
ENT2	m
ST1P,LV1,RST2	n
LV2	1
TOTAL	*3n+m+2*
With Execution Conditions	***(Best Case)***
ST2	1
ENT2	m
ST1P	1
LV1	n
ST1	n-1
RST2,LV2	1
TOTAL	*2n+m+3*

Figure 8: Complexity Calculations for Different Canceling Edge Elimination Methods

Assuming that scheduling events is the most time consuming task in a simulation, then the model with execution conditions should be able to outperform equivalent models that use either canceling edges or the approach suggest by Savage and Schruben (1995). At this point it is difficult to make a more definitive statement but these preliminary results look promising. A more thorough analysis of the computational requirements of these methods is the subject of ongoing research.

6 CONCLUSION

The implementation of execution conditions in the simulation graph methodology provides a standard, complete methodology for modeling interrupts. The implementation of execution conditions should be straightforward in any simulation software and be very useful for the simulation practitioner.

ACKNOWLEDGMENTS

We gratefully acknowledge the financial support of the CBA/GSB Faculty Research Committee of the College of Business Administration of the University of Texas at Austin. The second author gratefully acknowledges additional support from the End to End Simulation Department at Schlumberger Austin Research.

REFERENCES

Sargent, R. G. 1988. Event Graph Modeling for Simulation with an Application to Flexible Manufacturing Systems. *Management Science* 34 (10): 1231-1251.

Savage, E. L. and L. W. Schruben. 1995. Eliminating Event Cancellation in Discrete Event Simulation. In *Proceedings of the 1995 Winter Simulation Conference*, ed. C. Alexopoulis, K. Kang, W. R. Lilegdon, and D. Goldsman, 744-750. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Schruben, L.W. 1983. Simulation Modeling with Event Graphs. *Communications of the ACM*, 26(11): 957-963.

Schruben, L. W. 1995. *Graphical Simulation Modeling and Analysis Using Sigma for Windows.* Danvers, Massachusetts: Boyd & Fraser Publishing Company.

Schruben, L. W. and E. Yücesan. 1993. Modeling Paradigms for Discrete Event Simulation. *Operations Research Letters* 13: 265-275.

Som, T. K. and R. G. Sargent. 1989. Formal Development of Event Graphs as an Aid to Structured and Efficient Simulation Programs. *ORSA Journal on Computing* 1 (2):107-125.

Yücesan, E. 1989. Simulation Graphs: A Mathematical Framework for The Design and Analysis of

Discrete Event Simulations. Ph.D. Dissertation, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, New York.

Yücesan E. and L. W. Schruben. 1992. Structural and Behavioral Equivalence of Simulation Models. *ACM Transactions on Modeling and Computer Simulation* 2 (1): 82-103.

AUTHOR BIOGRAPHIES

RICKI G. INGALLS is the Manager of Business Analysis in the Manufacturing Strategy Group at Compaq Computer Corporation in Houston, Texas. He has been involved in the application and development of operational modeling tools and techniques in the electronics industry for over 12 years. He has a B.S. in Mathematics from East Texas Baptist College, a M.S. in Industrial Engineering from Texas A&M University and is currently a Management Science Ph.D. candidate at the University of Texas at Austin. Prior to re-joining Compaq, he was on the technical staff of the Operational Modeling Group at SEMATECH, Manager of the Operations Analysis Group at Compaq, a consultant with the Electronics Automation Application Center of General Electric Co. and an Industrial Engineer with Motorola, Inc. His research interests include simulation methodologies, qualitative simulation, and simulation applications. He is a member of the Society for Computer Simulation.

DOUGLAS J. MORRICE is an Associate Professor in the Department of Management Science and Information Systems at The University of Texas at Austin. He is currently on sabbatical leave in the End to End Simulation Department at Schlumberger Austin Research in Austin, Texas. Dr. Morrice received his undergraduate degree in Operations Research at Carleton University in Ottawa, Canada. He holds an M.S. and a Ph.D. in Operations Research and Industrial Engineering from Cornell University. His research interests include discrete event and qualitative simulation modeling and the statistical design and analysis of large scale simulation experiments. Dr. Morrice is a member of the The Institute for Operations Research and Management Science (InfORMS) and the Council of Logistics Management. He served as the Secretary for the InfORMS College on Simulation (1994-1996) and is Co-Editor of the *Proceedings of the 1996 Winter Simulation Conference.*

ANDREW B. WHINSTON is a professor of both business and computer science at the University of Texas at Austin, where he is also a fellow of the IC2 Institute and director of the Center for Information Services Management, and holds the Hugh Roy Cullen Centennial Chair in Business Administration. His research deals with decision support systems theory, distributed AI, organization modeling and qualitative modeling. He received his Ph.D. in management from Carnegie Mellon University, Pittsburgh, PA, and is a member of the Institute of Management Science.