From *Simulation*, vol. 62, number 2, February 1994, pp. 98–111.

IMPORTANCE SAMPLING SIMULATION IN *UltraSAN* *

W. Douglas Obal II and William H. Sanders
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721 USA
(602) 621-6181

obal@ece.arizona.edu and whs@ece.arizona.edu

ABSTRACT

Traditional simulation techniques perform poorly when estimating performance measures based on rare events. One solution to this problem is the use of importance sampling. However, two problems that have limited the use of importance sampling are the lack of a formal framework for specifying importance sampling strategies, and the fact that in most cases the simulations must be hand-coded—a very time consuming process. This paper presents a software tool that facilitates experimentation with importance sampling by addressing these two problems. First, the tool is based on a flexible framework for specifying importance sampling simulations in terms of stochastic activity networks. Second, once specified, the importance sampling simulation program is automatically generated by the tool, freeing the researcher to focus on the modeling problem. The effectiveness of the software is demonstrated through the solution of a machine-repairman model with Weibull distributed failure times and a delayed group repair policy. Orders of magnitude reduction in the CPU time required to obtain a specified relative accuracy were achieved.

Key Words: Discrete event simulation, Importance sampling, Rare event simulation, Stochastic activity networks, Stochastic Petri nets.

This work was supported in part by The University of Arizona Advanced Telecommunications Research Project and the Digital Equipment Corporation Faculty Program: Incentives for Excellence.

I Introduction

Discrete event modeling and simulation plays an important role in the development of many modern systems. In the design phase, model-based system evaluation has become a popular alternative to the relatively costly and time consuming process of prototyping and testing. When planning changes to an existing system, model-based evaluation can help identify the most beneficial modifications, and provide an early warning against changes that may reduce performance. Moreover, model-based evaluation can usually be carried out without affecting the operation of the system under study, whereas testing activities often impact system performance by increasing response times or disrupting service.

Typically, a model of a discrete event system is created to solve for one or more performance measures of interest. We use the term "performance measure" in a very broad sense, meaning whatever metric the modeler feels is important. The performance measure may be reliability, transactions per second, response time, availability, etc.

The characteristics of a model determine which solution methods may be employed to obtain the desired performance measure. If the model satisfies the Markov property, and the state space is not too large relative to available computing resources, analytical techniques can be utilized. When model characteristics preclude an analytical solution, discrete event simulation is an effective alternative. In many cases, simulation is more flexible than analytical techniques and less expensive than prototyping and testing.

The primary drawback of simulation is that it can take an unacceptably long time to produce an accurate estimate of the performance measure. Often, the source of inefficiency in simulation is a performance measure based on a rare event. Such performance measures are often required in system validation. For example, the specification for a highly dependable system might require the unreliability of the system over some interval of time to be less than 10^{-6}. When estimating a performance measure based on a rare event, traditional simulation is inefficient because most of the effort is spent processing events that have little or no impact on the performance measure. Intuitively, we want to make the interesting, or important, events occur more often, so that we obtain a higher rate of relevant samples.

Importance sampling [1, 2] is one technique that can be used to increase the relative frequency of rare events. Originally developed as a variance reduction technique for Monte Carlo simulations [3], the crux of importance sampling is to alter the probability measure governing events so that the sample variance is reduced. The resulting observations are

then weighted to compensate for the use of the alternative probability measure. For a rare event simulation, this means causing the formerly rare event to occur more often, thereby achieving a smaller variance in the weighted observations of the performance measure.

One obstacle to the effective application of importance sampling to discrete event simulations is the difficulty of deciding how to alter the probability measure. Calculating the optimal change of probability measure is hard, since the calculation requires knowledge of the quantity to be estimated. Researchers have, therefore, focused on finding good heuristics for particular types of models, such as models of dependable systems [4, 5] and certain classes of queuing networks [6, 7]. For an alternative approach to the rare event problem, see Van Moorsel, et al. [8].

Two problems hindering importance sampling researchers are that there is no simple representation for importance sampling heuristics, and that there is no easy way to experiment with new heuristics. To date, most researchers have hand-coded simulation programs for each problem. The main benefit of this approach is the high degree of flexibility available to the simulation programmer. On the other hand, it is very time consuming to program, validate, debug, etc., for each new problem or importance sampling heuristic.

We have designed and implemented an importance sampling facility for the *UltraSAN* [9] modeling package that addresses these problems. *UltraSAN* is a software tool for model-based performance, dependability, and performability evaluation of systems. The tool is based on stochastic activity networks (SANs) [10, 11], a stochastic extension to Petri nets [12, 13]. The *UltraSAN* importance sampling simulation facility comprises two parts. The first is a formal framework for specifying importance sampling heuristics in terms of the random elements of SANs [14]. The second part of the facility is a simulation engine that accepts the original SAN model and the importance sampling strategy as input, simulates the model under importance sampling, weights the observations to compensate for the altered model, and yields an unbiased estimate of the performance measure.

The primary advantage of our approach is the level of automation made possible by the use of SANs and a formal specification framework for importance sampling. SANs have very precise execution rules, allowing a single simulation engine to handle any given SAN. By combining SANs with an equally precise specification of an importance sampling strategy, the facility produces a specification of the overall simulation precise enough to be automatically translated into a procedural programming language and compiled. *UltraSAN* automatically generates an executable simulation by linking the model and importance

sampling heuristic specification with the simulation engine. Thus, by using the *UltraSAN* facility, the modeler is freed from the problem of designing, coding, validating, and debugging the simulation program. Instead, he may focus on his model. Furthermore, changes to the model or importance sampling heuristic are expedited, since the modeler does not have to alter a program—he simply alters his description of the model or the importance sampling strategy.

The remainder of this paper is organized as follows. First, the theory underlying the importance sampling specification framework and simulation engine is reviewed, and an example SAN model is introduced. Next, the *UltraSAN* implementation of the framework for importance sampling simulations is discussed. Finally, an example is used to demonstrate the utility of the *UltraSAN* importance sampling facility.

II Theory

In this section, we give an overview of the framework for importance sampling presented in [14].

A Introduction to Importance Sampling

As stated in the introduction, the main idea behind importance sampling is to alter the probability measure on the set of outcomes to enhance the probability of rare, but important, events. Consider a probability space $(\Omega, \mathcal{F}, \mathcal{P})$ and a random variable X defined on the space. The power of importance sampling arises from the following transformation:

$$E_P[X] = \int_\Omega X \, dP = \int_\Omega X \, \frac{dP}{dP'} \, dP' = E_{P'}[XL],$$

where P' is an alternative probability measure on $(\Omega, \mathcal{F})$ and $L = dP/dP'$ is the *likelihood ratio*. This transformation suggests that an estimate of $E_P[X]$ can be calculated by sampling XL under P' and forming the sample mean.

In a discrete event simulation, the performance measure is usually specified as a function of the sample path. We use the term "sample path" to refer to the sequence of events that occur as the simulation program executes. So, an event is rare if the sample paths in which it occurs are rare. Therefore, to increase the relative frequency of a rare event in a discrete event simulation, one must increase the probability of the set of sample paths in which it occurs.

However, the probability measure on the set of possible sample paths for a given discrete event simulation is usually not available in closed form. Instead, it is induced by the random elements built into the model. Therefore, to alter the probability measure on the set of sample paths one alters the random elements of the model. The random elements of SAN models are the probabilistic characteristics of SAN components, which are discussed in the next subsection.

B Stochastic Activity Networks

In the *UltraSAN* modeling package, models are specified as stochastic activity networks (SANs) [10, 11]. Since we wish to induce alternative probability measures by manipulating the random elements of models, a brief description of SANs is required.

A SAN is composed of *places*, *activities*, and *gates*. Places, as in Petri nets, hold *tokens*. The modeler is free to interpret numbers of tokens in places any way he wishes. The number of tokens in a place is called the *marking* of the place. A vector containing the marking of each place in a SAN is called the marking of the SAN.

Activities are the random elements in SANs. There are two types of activities. *Timed* activities represent system delays relevant to the performance measure of interest, while *instantaneous* activities are used for delays that have little or no impact on the performance measure. The delay associated with a timed activity is called the *activity time* and is described by a probability distribution function called the *activity time distribution function*.

Gates connect places and activities. There are two types of gates. Input gates have one or more *input places* and are connected to a single activity. Output gates are connected to a single activity, and have one or more *output places*. *Input* gates have a *predicate* and a *function*. *Output* gates have only a function. A predicate is a Boolean function of the marking of the places connected to the gate. When an input gate predicate is true, the gate *holds*. An activity is *enabled* when all of its input gates hold.

When an activity becomes enabled, it is *activated*. The time at which an activity is activated is called the *activation time* of the activity, and the marking in which the activity is activated is called the *activation marking* of the activity. At activation time, the activity time is sampled from the activity time distribution, which may depend on the activation marking. The *completion time* of the activity is calculated by adding the activity time to the activation time. The modeler has the option of specifying a *reactivation* function. For a given activity and activation marking, this function returns a set of markings that, if

reached before completion, causes the activity to be aborted and immediately activated in the new marking. By default, the set of reactivation markings is empty for all activation markings. If the marking of the SAN changes such that one or more of the input gates connected to the activity no longer hold, the activity becomes disabled and is aborted. If the activity remains enabled until the completion time is reached, the activity *completes*.

Cases represent uncertainty about what happens when an activity completes. Each activity has a *case distribution*, a discrete probability distribution over the cases of the activity that may depend on the marking in which the activity completes.

Once an activity completes and a case is chosen, the new marking is determined by the input gates connected to the activity and the output gates connected to the chosen case. Input gate functions are executed first, followed by output gate functions. The functions map from the markings of the connected places into the set of all possible markings for the connected places.

To help clarify the definitions of SAN components, the following subsection presents a SAN model of a machine-repairman system.

C Example SAN Model

Figure 1 shows a SAN model of a machine-repairman system. The system is composed of two types of components. There are two components of type-one, and four components of type-two. The components are represented in the SAN by places "comp_1_1," "comp_1_2," "comp_2_1," etc. A token in one of these places indicates that the component is working. Each component fails independently of the others, and the failure times of components of the same type are identically distributed.

The failure times of the components are modeled by separate timed activities "fail_1_1," "fail_2_1," etc. As shown in Table 1, the activities corresponding to the same type of component have the same activity time distributions. The failure times of the components have Weibull distributions with shape parameter α and scale parameters β_1 for type-one and β_2 for type-two components. For this example, $\alpha = 1$, $\beta_1 = 100$, and $\beta_2 = 50$. Later, we will vary α.

When a failure activity completes, the token is removed from its input place and deposited in "failed_1" or "failed_2," based on the type of the component.

Repair begins only after two components of the same type have failed. Places "failed_1" and "failed_2" are used to maintain a count of failed components of each type. The predicate

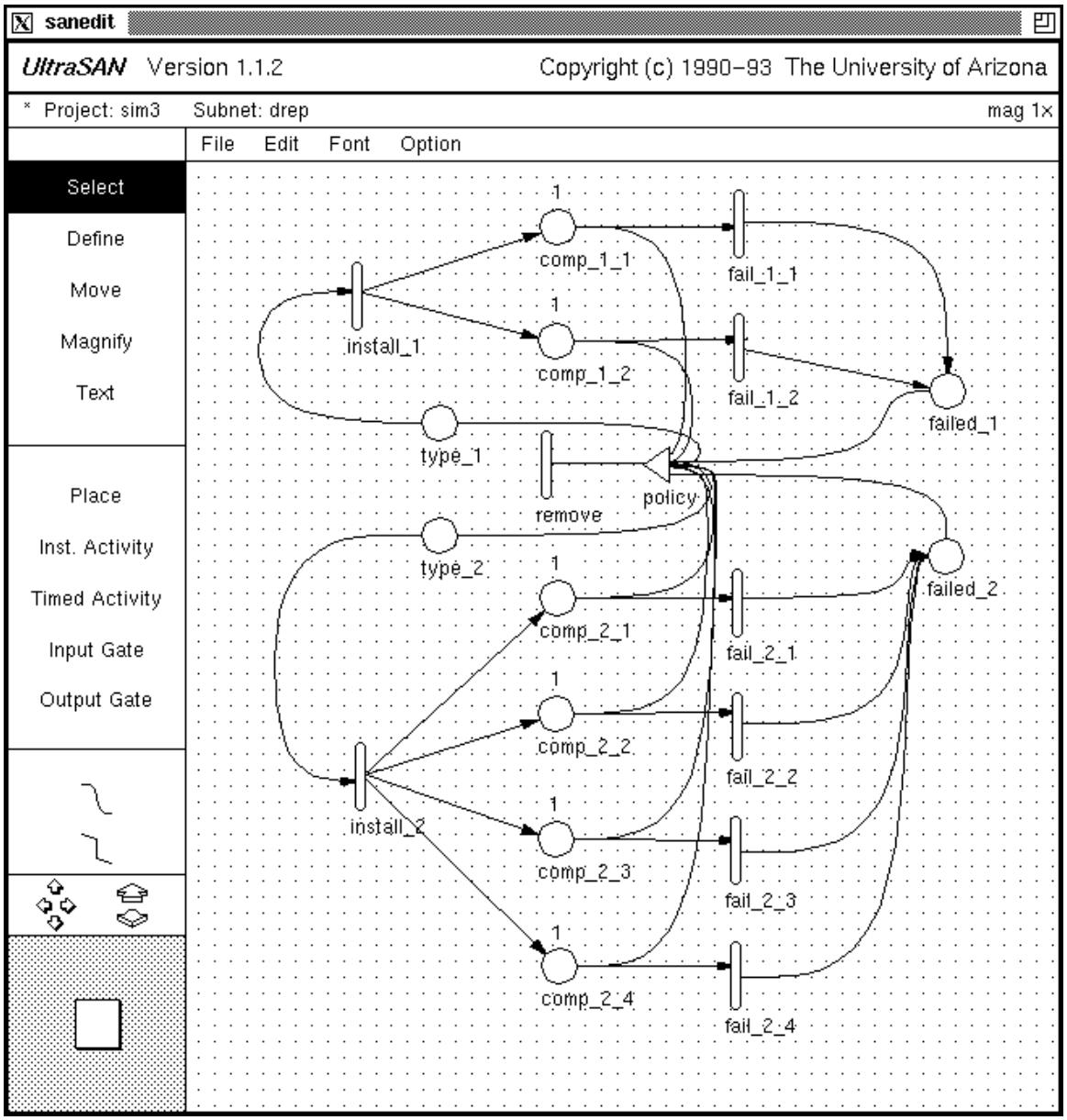

Figure 1: Machine-Repairman Model with Delayed Repair Policy

Table 1: Activity Time Distributions

Activity	**Distribution**
$fail\_1\_1$	Weibull(α,β_1)
$fail\_1\_2$	Weibull(α,β_1)
$fail\_2\_1$	Weibull(α,β_2)
$fail\_2\_2$	Weibull(α,β_2)
$fail\_2\_3$	Weibull(α,β_2)
$fail\_2\_4$	Weibull(α,β_2)
$install\_1$	exponential(3600.0)
$install\_2$	exponential(3600.0)
$remove$	hyperexponential(1.0,0.5,0.9)

of input gate "policy," shown in Table 2, enables timed activity "remove" only if "failed_1" or "failed_2" contains at least two tokens. Repair of type-one components has preemptive priority over repair of type-two components. This requirement is met by introducing a reactivation predicate for the "remove" activity. As shown in Table 3, the "remove" activity is reactivated (aborted and restarted) if it is activated in a marking with less than two type-one components failed, and the SAN reaches a marking in which two type-one components are failed before "remove" completes.

Repair consists of two stages. First, the failed components group must be removed and repaired or replaced. Then, the component group is reinstalled. The two-phase nature of the repair is modeled by two activities in series. Timed activity "remove" models the time to remove and repair a component group. As shown in Table 1, "remove" has a hyperexponential activity time. With probability 0.9, the removal time will be exponentially distributed with rate parameter 1, but with probability 0.1, the removal time will be exponentially distributed with rate parameter 0.5. Timed activities "install_1" and "install_2" model the time it takes to install components of type-one and type-two, respectively. Both are exponentially distributed with rate parameter 3600, the assumption being that most of the remove/repair time is spent on repair, so that the installation of the repaired component group is almost negligible.

When repair of a component type is complete, all components of that type are as good as new. Since the function of input gate "policy" (see Table 2) clears the component places upon completion of timed activity "remove," when the install activity completes and replaces the tokens, the failure activities are activated with new activity time distributions.

Table 2: Input Gate Predicates and Functions

Gate	Enabling Predicate	Function
policy	$(MARK(failed\_1) < 2 \;\&\&\; MARK(failed\_2) >= 2) \;\|\|$ $(MARK(failed\_1) == 2 \;\&\&\; MARK(failed\_2) < 4)$	$if\ (MARK(failed\_1) == 2)\ \{$ $\quad MARK(failed\_1) = 0;$ $\quad MARK(comp\_1\_1) = 0;$ $\quad MARK(comp\_1\_2) = 0;$ $\quad MARK(type\_1) = 1;\}$ $else\ \{$ $\quad MARK(failed\_2) = 0;$ $\quad MARK(comp\_2\_1) = 0;$ $\quad MARK(comp\_2\_2) = 0;$ $\quad MARK(comp\_2\_3) = 0;$ $\quad MARK(comp\_2\_4) = 0;$ $\quad MARK(type\_2) = 1;\}$

Table 3: Activation and Reactivation Predicates

Activity	Activation	Reactivation
remove	$MARK(failed\_1) < 2$	$MARK(failed\_1) == 2$

Thus, the components are as good as new.

When all components fail, the system fails, and all repair activity halts. The predicate of input gate "policy" (Table 2) ensures that the "remove" activity becomes disabled when the marking of the SAN is such that the marking of place "failed_1" is two and the marking of place "failed_2" is four. A SAN marking such as this, in which no activities are enabled, is called an *absorbing marking*.

D Performance Measures

Performance measures for SANs are specified using the concept of *reward variables* [15], extended to SANs [10, 16]. A reward variable is defined by a *reward structure* and a *variable type*. The reward structure associates *rate rewards* with SAN markings, and *impulse rewards* with activity completions. The instantaneous reward associated with a given SAN marking is the sum of the rate rewards associated with the marking and the impulse reward of the last activity that completed.

The variable type determines how the performance measure is calculated from the reward structure. Three variable types are *instant-of-time*, *interval-of-time*, and *time-averaged-interval-of-time*. An instant-of-time variable is assigned the instantaneous reward associated with the SAN marking at a specified time. Interval-of-time variables accumulate the reward earned over an interval of specified length, beginning at a specified time. Time-averaged-interval-of-time variables give the reward accumulated over an interval, divided by the length of the interval.

As an example, consider estimating the unreliability of the machine-repairman model. The unreliability is the probability that the system fails during some interval of time. Since the system failed state is an absorbing marking, we can find the probability that the system fails in an interval, given it was functioning at the beginning of the interval, by examining the state of the system at the end of the interval. If we assign a rate reward of one to the failed marking, and zero to all other markings, we can obtain the unreliability of the system as the expected value of an instant-of-time variable evaluated at the end of the interval.

E Framework for Importance Sampling

The application of importance sampling to a SAN simulation requires the ability to alter the probability measure governing the space of possible sample paths of the SAN. Since the

measure is induced by the definitions of the activities in the SAN, the most natural approach to specifying an alternative measure for a SAN is through modifications to the activities in the SAN. The *governed SAN* [14] provides a versatile mechanism for defining the alternative probability measure on the sample path space of a SAN.

A governed SAN consists of a pairing of a SAN with an *importance sampling governor*. The importance sampling governor, or "governor," is similar to a finite state machine. Each state of the governor is an alternative definition of the activities in the SAN, corresponding to a particular bias of the probability measure on the space of sample paths of the SAN. The governor changes state based on the evolution of a sample path. Given the current governor state, and a new marking of the SAN, the governor state transition function determines whether a governor state change should occur.

To demonstrate the utility of the importance sampling governor, we present the description of an importance sampling strategy for the machine-repairman SAN discussed in Subsection C, using a recently developed heuristic for dependable systems with delayed group repair [17]. The heuristic, called *modified failure biasing* (MFB), is derived from the *failure biasing* technique invented by Lewis and Böhm [18]. MFB was designed for regenerative simulation [19], where the observation period is the time between successive visits to a specified regenerative state. In terminating simulation, the observation period is specified in terms of the simulation clock. In [14] we found that in terminating simulation, the performance of the heuristic could be improved by augmenting MFB with an approximate forcing technique [5]. Since the time horizons under consideration are small, it is unlikely that the system will reach a state where repair is initiated before the time horizon. By decreasing the mean time to failure of components in the biased model, the probability that the system reaches a state where repair activity is initiated is increased. Thus, the system is forced out of the initial set of states and into the set of important states.

Figure 2 shows the state diagram for a governor implementing MFB augmented by approximate forcing. Table 4 contains the state definitions, while the governor state transition function is shown in Table 5.

F Simulating Governed SANs

Once a governed SAN has been constructed, it must be simulated to obtain the desired performance measure. SAN simulation is carried out at the network level [10, 20]. In this subsection, we give a brief description of the simulation of a governed SAN, using the

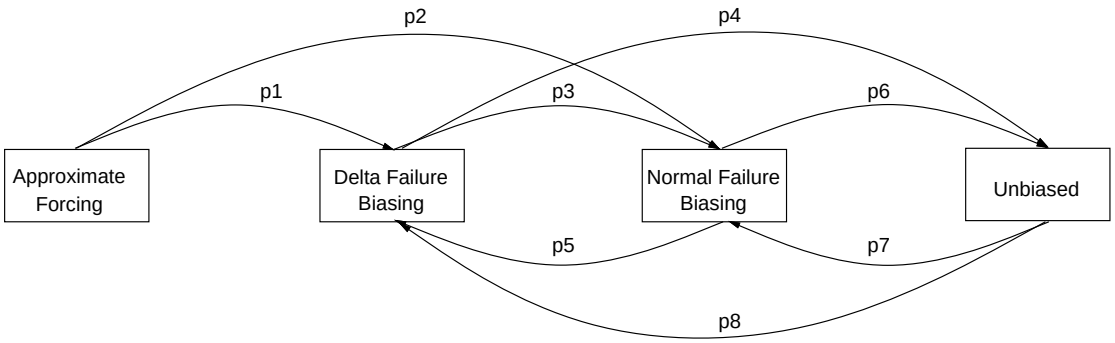

Figure 2: Governor for Modified Failure Biasing with Approximate Forcing

Table 4: State Specification for Governor for MFB with Approximate Forcing

State	**Bias**		
	Component	*Distribution*	*Parameters*
approximate forcing	type-one	Weibull	$\alpha = 1.0, \beta_1 = 25$
	type-two	Weibull	$\alpha = 1.0, \beta_2 = 12.5$
delta	type-one	Weibull	$\alpha = 1.0, \beta_1 = 10.0$
	type-two	Weibull	$\alpha = 1.0, \beta_2 = 5.0$
normal	type-one	Weibull	$\alpha = 1.0, \beta_1 = 2.0$
	type-two	Weibull	$\alpha = 1.0, \beta_2 = 1.0$
unbiased	unbiased		

Table 5: State Transition Function for Governor for MFB with Approximate Forcing

Current State	**Arc**	**Predicate**	**Next State**
approximate forcing	p1	$(MARK(failed\_1) == 1$ && $MARK(failed\_2) == 2)$ || $(MARK(failed\_1) == 2$ && $MARK(failed\_2) == 1)$	delta
	p2	$(MARK(failed\_1) == 0$ && $MARK(failed\_2) == 2)$ || $(MARK(failed\_1) == 2$ && $MARK(failed\_2) == 0)$	normal
delta	p4	$MARK(failed\_1) <= 1$ && $MARK(failed\_2) <= 1$	unbiased
	p3	$(MARK(failed\_1) == 2$ && $MARK(failed\_2) == 2)$ || $(MARK(failed\_1) == 1$ && $MARK(failed\_2) == 3)$	normal
normal	p6	$MARK(failed\_1) <= 1$ && $MARK(failed\_2) <= 1$	unbiased
	p5	$(MARK(failed\_1) == 1$ && $MARK(failed\_2) == 2)$ || $(MARK(failed\_1) == 2$ && $MARK(failed\_2) == 1)$	delta
unbiased	p8	$(MARK(failed\_1) == 1$ && $MARK(failed\_2) == 2)$ || $(MARK(failed\_1) == 2$ && $MARK(failed\_2) == 1)$	delta
	p7	$(MARK(failed\_1) == 0$ && $MARK(failed\_2) == 2)$ || $(MARK(failed\_1) == 2$ && $MARK(failed\_2) == 0)$	normal

machine-repairman model in Figure 1 to help clarify the main points.

Starting in the initial marking, enabled activities are activated. The initial marking of the machine-repairman model consists of a single token in each of the component places "comp_1_1," "comp_1_2," "comp_2_1," etc. Therefore, each of the corresponding failure activities is enabled in the initial marking. Upon activation, an activity time is sampled according to the activity time distribution function assigned to the activity by the importance sampling governor, and the calculated completion time is placed on the future events list. For the machine-repairman model, the future events list will be initialized with completion times for each of the failure activities. The activity with the earliest completion time is identified, and a case is chosen according to the case distribution for that activity in the current marking. Then the functions of each input gate connected to the activity are executed, followed by the functions of output gates connected to the chosen case, to determine the next marking. If the next marking has instantaneous activities enabled, they are completed immediately, cases are chosen, and gate functions are executed until a marking is reached in which there are no instantaneous activities enabled. The new marking is referred to as the *next stable marking*. There are no instantaneous activities in the machine-repairman model, so all of the markings are stable.

When the next stable marking is reached, the likelihood ratio is updated, and the governor state transition function is executed to determine whether the governor should change state. Hence, for the machine-repairman model with the governor in Figure 2, the governor will evaluate the transition function associated with initial governor state "approximate forcing," shown in Table 5. If one of the predicates is true, the governor will transition to the corresponding next state. Input gates connected to each activity are checked to see which activities become disabled, which activities remain enabled, and which activities are newly enabled. Disabled activities are aborted; their completion times are removed from the future events list. Activities that remain enabled are checked to see if they must be reactivated in the new marking. Activities must be reactivated if the governor has changed to a state that assigns a new activity time distribution function to the activity, or if a reactivation predicate holds. If reactivation is called for, the activity is aborted (the old completion time is removed from the future events list), and a new completion time is calculated based on a new sample from the activity time distribution specified by the governor for the activity in the new marking. Newly enabled activities are activated, and their completion times are added to the future events list. After determining the status of

the activities in the SAN, the activity with the earliest completion time is identified, and the process repeats.

As suggested in Subsection A, when simulating under an altered probability measure, one must compensate by weighting the resulting observations by the likelihood ratio. Otherwise, the estimator will be biased. Therefore, when simulating a governed SAN, one must calculate the ratio of the likelihood of the sample path under the probability measure induced by the original definition of the SAN to the likelihood of the path under the probability measure induced by the modifications imposed by the governor. A technique for iteratively updating the likelihood ratio upon each state transition was developed by Nicola, et al. [5] and adapted to governed SANs in [14]. The gist of the method is to focus on the likelihood that a particular activity completes to cause the next marking change, given the history of the sample path, and then to calculate the conditional probability that a particular next stable marking is reached, given that the transition was caused by that activity.

III Implementation in *UltraSAN*

The last section described the theory underlying the importance sampling facility in *UltraSAN*. In this section, the implementation of the theory and its incorporation into the *UltraSAN* modeling tool is presented. We begin with an overview of the tool.

A *UltraSAN*

UltraSAN [9] is a software tool for model-based evaluation of discrete event systems represented as stochastic activity networks. It runs on the UNIX operating system under the X Window System, and is currently available for the Convex, DECstation, IBM RS6000 and Sun/4 architectures. The effectiveness of the tool has been demonstrated through its use in several model-based evaluation studies, e. g., [21, 22, 23].

Figure 3 shows the organization of the tool. Models are specified through a graphical interface that runs on the X Window System. The three main components of the user-interface (SAN editor, composed model editor, and performability variable editor) appear at the top of Figure 3. The SAN editor allows one to specify SANs by simply drawing the SAN. Figure 1 is a screen dump showing the machine-repairman model as it appears in the SAN editor. Auxiliary editors are provided for defining each SAN component. For example,

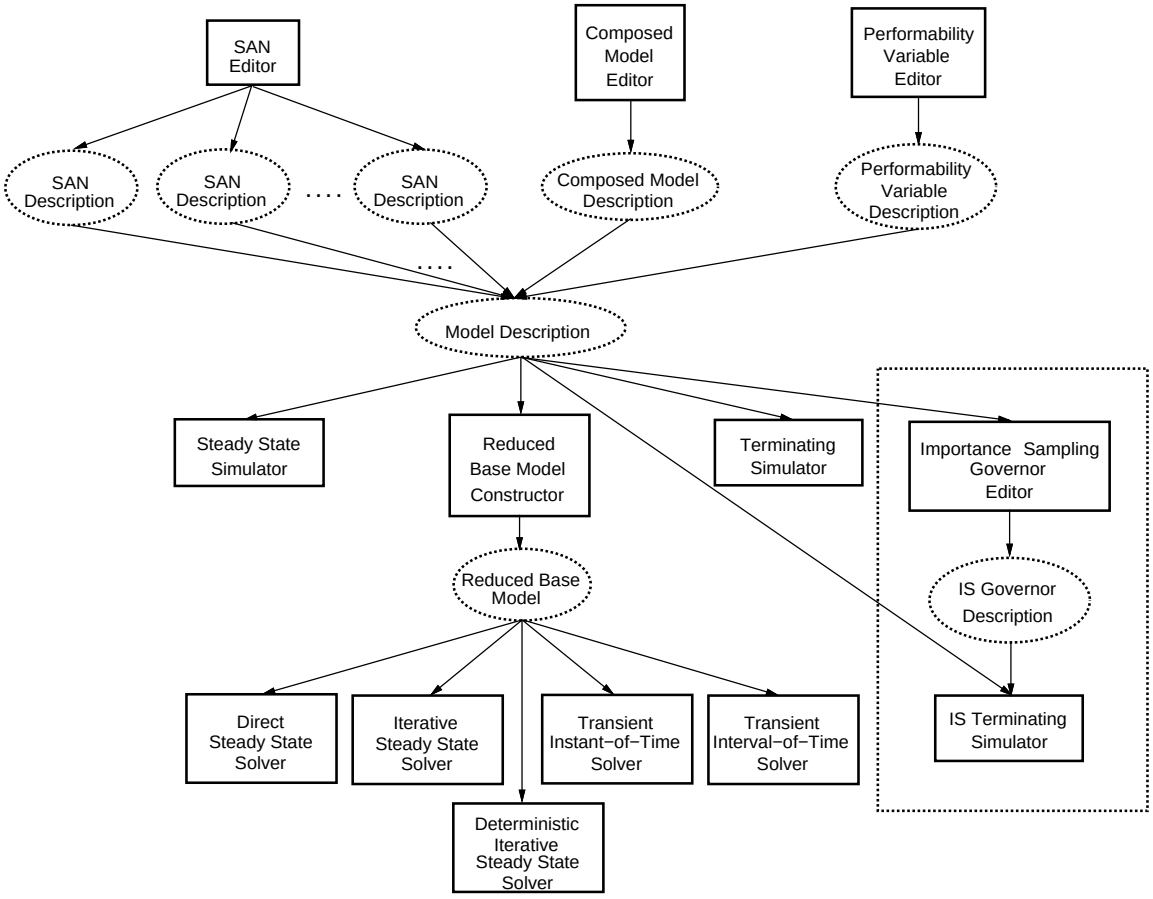

Figure 3: Organization of *UltraSAN*

an activity editor is used to select the activity time distribution function, its parameters, and its reactivation function. The composed model editor can be used to combine individual SANs into a hierarchical *composed model* using *replicate* and *join* operations [24]. Finally, the performability variable editor is used to specify reward variables for the desired performance measures.

The primary function of the user-interface modules is to generate a C language description of the model from its graphical specification. The generated C code description of the model is automatically compiled and linked with other libraries to produce the reduced base model construction program and the simulation programs.

The reduced base model construction program generates a stochastic process description of the model that is amenable to solution by standard analytical methods for continuous time discrete state space Markov processes. The generated stochastic process serves as input to the analytical solvers available in *UltraSAN* [9, 25, 26, 27]. Note that the model must satisfy the Markov property to be analytically tractable. SAN models satisfy the Markov property if all activity time distributions are exponential, and activities are reactivated often enough so that their rates depend only on the current marking [24].

Two different traditional simulation programs may be generated. The steady state simulator uses an iterative batch means [28] technique after a specified initial transient period to estimate the steady-state expected value and variance of instant-of-time reward variables. The terminating simulator uses the method of independent replications to estimate the expected value and variance of all three reward variable types described in section II.D for a specific time or time interval. Both simulators are designed to estimate reward variables to within a user-specified relative accuracy at a user-specified confidence level. Moreover, both simulators are designed to take advantage of symmetries in the composed model to increase the efficiency of the simulation [20].

B Importance Sampling Facility

The importance sampling framework detailed in [14] has been implemented and incorporated into the *UltraSAN* modeling package. Consistent with the rest of the package, the importance sampling facility has an X Window System-based user-interface that generates C code describing the importance sampling governor. This code is automatically compiled and linked with other *UltraSAN* libraries to produce an importance sampling simulation

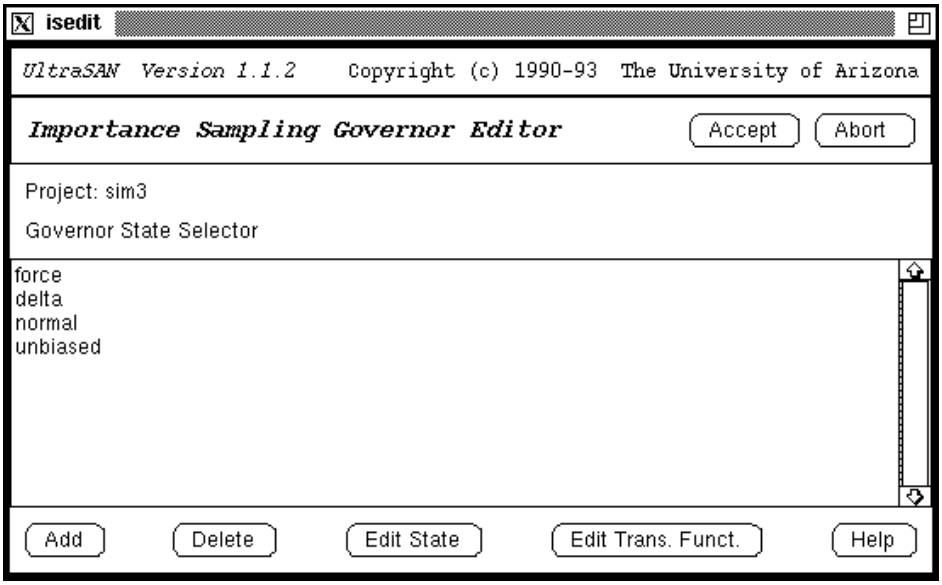

Figure 4: Importance Sampling Governor Editor

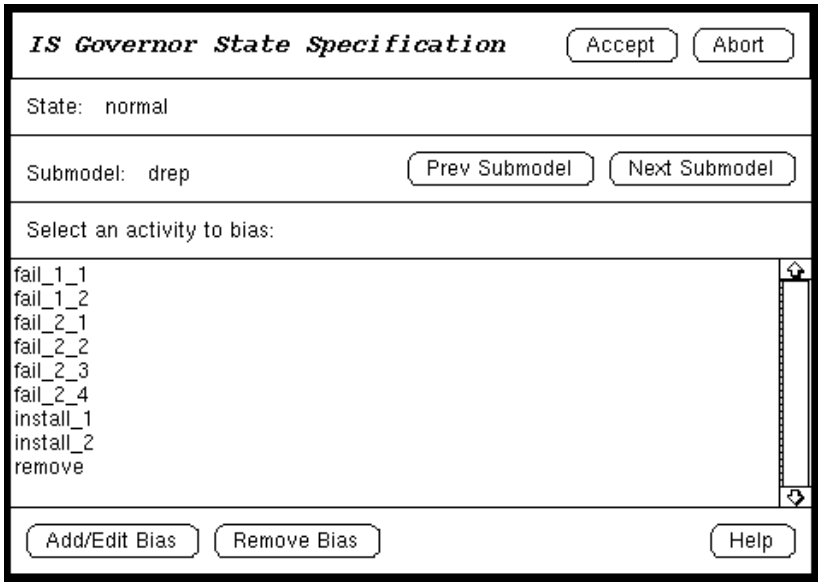

Figure 5: Governor State Editor

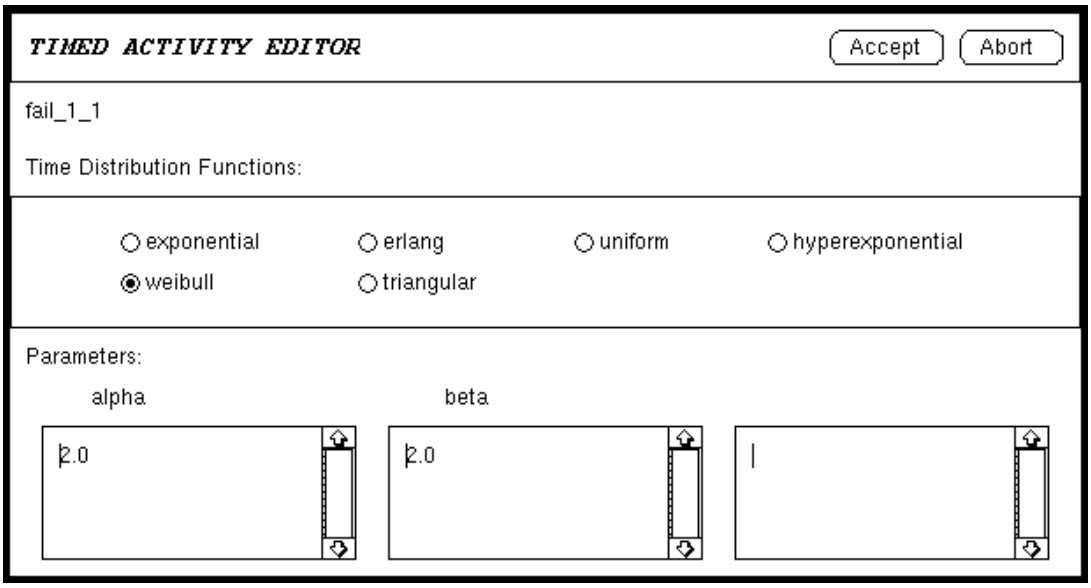

Figure 6: Activity Bias Editor

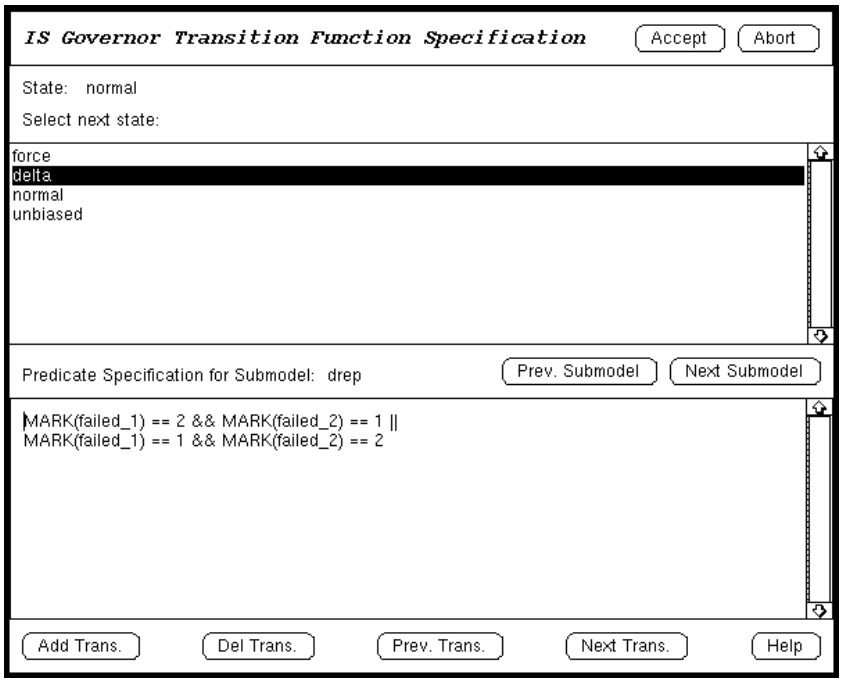

Figure 7: Governor State Transition Function Editor

program.

The user-interface to the *UltraSAN* importance sampling facility has three main components. The first component, shown in Figure 4, is the importance sampling governor editor. With this editor, the user can add, edit, or delete states from the importance sampling governor, and edit the governor state transition function. In Figure 4, the four states of the governor for the machine-repairman model are listed.

Recall that each governor state may contain new definitions of the timed activities in each SAN in the composed model. We refer to a new definition of an activity as a *bias* on that activity. The governor state editor, shown in Figure 5, allows the user to scroll through the list of SANs in the composed model, using the Prev. and Next Submodel buttons. For each SAN, the editor displays a list of the timed activities in the SAN. In Figure 5, state "normal" of the governor for the machine-repairman example is shown. One may choose to edit or remove a bias from any timed activity in the model. The activity bias editor, shown in Figure 6, allows the user to redefine the activity time distribution and case distribution. The example activity shown in Figure 6 is "fail_1_1" from the machine-repairman example. "Fail_1_1" has only one case, so the case distribution portion of the editor is not displayed.

Figure 7 shows the importance sampling governor state transition function editor. The transition function editor is used to specify when the governor should change state. The state transition function for the governor is defined state by state. For each governor state, one specifies arcs to other governor states. Associated with each transition arc is a predicate that defines conditions under which the transition will be taken. The predicate for the arc is defined in terms of predicates on the SANs in the composed model. The example in Figure 7 shows the predicate for the arc from governor state "normal" to governor state "delta."

When the user has finished specifying the governor, the user-interface converts his description to a series of C functions and header files and compiles them into a library. The importance sampling simulation program is generated by linking the library describing the governor, the library containing code for the importance sampling simulation engine, and a short C program that handles initializations and makes the initial call into the simulation code.

Table 6: Importance Sampling Strategies for the Machine-Repairman Model

Name	**Heuristic**	**State**	α	β_1	β_2
IS-1	MFB	normal	1	2	1
		delta	1	30	15
IS-2	MFB	normal	1	2	1
		delta	1	10	5
IS-3	MFB	normal	1	2	1
		delta	1	4	2
IS-4	MFB + approx. forcing	force	1	25	12.5
		delta	1	10	5
		normal	1	2	1

IV Results

The results obtained from the machine-repairman model are presented in this section. First, MFB is applied and tuned to achieve good performance. Then, approximate forcing is applied to further improve the performance. Finally, the shape parameter of the failure distributions is modified so that the model cannot be solved analytically, and the developed importance strategy is applied.

Table 6 shows four importance sampling strategies for the machine-repairman model. The first three are straightforward MFB, with different levels of bias. Plain MFB is tantamount to specifying the approximate forcing state of the governor in Figure 2 as unbiased. These strategies are used to demonstrate the effects of tuning the bias to improve the performance of a heuristic for a given model. The fourth strategy augments MFB with approximate forcing.

Table 7 shows the results obtained for the unreliability of the machine-repairman system during the interval $[0, 10]$. All results are at a 95% confidence level, and the confidence interval is reported as the point estimate and the estimated relative error. One can see from Table 7 that all of the applied strategies reduced by orders of magnitude the CPU time required to achieve the desired accuracy. The second strategy, IS-2, outperformed the other MFB candidates. Thus, IS-2 was chosen as the strategy to use in conjunction with approximate forcing, forming IS-4. The results for IS-4 in Table 7 show a significant improvement over IS-2.

Table 7: Unreliability in $[0, 10]$ of Machine-Repairman System

Strategy	Traditional	IS-1	IS-2	IS-3	IS-4
Point Estimate	3.3×10^{-7}	3.8×10^{-7}	3.4×10^{-7}	3.8×10^{-7}	3.5×10^{-7}
Relative Error	10%	10%	10%	10%	10%
Replications	1,184,152,657	4,593,933	574,091	2,162,765	39,870
CPU Time (sec.)	740,699	4,871	610	2,320	128

Table 8: Results for Modified Machine-Repairman Model

Strategy	Traditional	IS-4
Point Estimate	1.5×10^{-5}	1.7×10^{-5}
Relative Error	10%	10%
Replications	26,524,623	152,730
CPU Time (sec.)	30,344	521

Table 8 shows results of applying MFB augmented by approximate forcing to a version of the machine-repairman model with components whose failure times are Weibull distributed with shape parameter $\alpha = 2$. In order to obtain traditional simulation results in a reasonable period of time, we altered the scale parameters as well. In the modified model, type-one components have scale parameter $\beta_1 = 30$; type-two components have scale parameter $\beta_2 = 15$. The IS-4 strategy is shown in Table 8 to be effective despite these significant changes to the failure distributions of components of the model.

V Conclusions

Traditional simulation techniques perform poorly when the desired performance measure is based on a rare event. One solution to the inefficiency inherent in the rare event problem is the use of importance sampling. Research on importance sampling has shown promise in several areas, but researchers were hindered by two problems. Namely, there was no convenient framework for formally representing importance sampling strategies, and most studies were done via hand-coded simulations.

We have presented a tool that facilitates experiments with importance sampling by addressing these problems. The tool is based on a framework for specifying importance

sampling simulations in terms of stochastic activity networks, and has been implemented within the context of *UltraSAN*, a software tool for model-based evaluation of systems represented as stochastic activity networks. When the *UltraSAN* importance sampling facility is used to specify an importance sampling strategy for a stochastic activity network, the tool automatically generates the importance sampling simulation program. Thus, no programming is required of the user.

To demonstrate the effectiveness of the tool, an example model of a machine-repairman system with a delayed group repair policy was introduced and solved. Using the *UltraSAN* importance sampling facility, we obtained several orders of magnitude reduction in the CPU time required to attain a specified relative accuracy. Furthermore, the developed importance sampling strategy proved effective even after significant changes to component failure time distributions.

Acknowledgments

The authors would like to thank the past and present members of the University of Arizona Performability Modeling Research Lab for their contributions to the development of *UltraSAN*. Without the tool as a starting point, the implementation of these ideas would have been much more difficult.

REFERENCES

[1] J. M. Hammersley and D. C. Handscomb, *Monte Carlo Methods*, Methuen & Co., Ltd., London, 1964.

[2] H. Kahn and M. Marshal, "Methods of reducing sample size in Monte Carlo computations," *Journal of the Operations Research Society of America*, vol. 1, pp. 263–278, November 1953.

[3] H. Kahn, "Random sampling (Monte Carlo) techniques in neutron attenuation problems," *Nucleonics*, vol. 6, pp. 27–33,36,60–65, May 1950.

[4] A. Goyal, P. Shahabuddin, P. Heidelberger, V. F. Nicola, and P. W. Glynn, "A unified framework for simulating Markovian models of highly dependable systems," *IEEE Transactions on Computers*, vol. 41, no. 1, pp. 36–51, January 1992.

[5] V. F. Nicola, M. K. Nakayama, P. Heidelberger, and A. Goyal, "Fast simulation of dependability models with general failure, repair and maintenance processes," in *Proceedings of the Twentieth Annual International Symposium on Fault-Tolerant Computing*, pp. 491–498, Newcastle upon Tyne, United Kingdom, June 1990.

[6] S. Parekh and J. Walrand, "A quick simulation method for excessive backlogs in networks of queues," *IEEE Transactions on Automatic Control*, vol. 34, no. 1, , January 1989.

[7] J. Walrand, "Quick simulation of queueing networks: An introduction," in *Computer Performance and Reliability*, G. Iazeolla, P. J. Courtois, and O. J. Boxma, editors, pp. 275–286, North Holland, 1987.

[8] A. P. A. van Moorsel, B. R. Haverkort, and I. G. Niemegeers, "Fault injection simulation: A variance reduction technique for systems with rare events," in *Second International Working Conference on Dependable Computing for Critical Applications*, pp. 57–64, Tucson, Arizona, February 1991.

[9] J. Couvillion, R. Freire, R. Johnson, W. D. Obal II, M. A. Qureshi, M. Rai, W. H. Sanders, and J. Tvedt, "Performability modeling with *UltraSAN*," *IEEE Software*, vol. 8, no. 5, pp. 69–80, September 1991.

[10] W. H. Sanders, *Construction and Solution of Performability Models Based on Stochastic Activity Networks*, PhD thesis, University of Michigan, 1988.

[11] J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior, and application," in *Proceedings of International Workshop on Timed Petri Nets*, pp. 106–115, Torino, Italy, July 1985.

[12] M. K. Molloy, "Performance analysis using stochastic Petri nets," *IEEE Transactions on Computers*, vol. C-31, pp. 913–917, September 1982.

[13] S. Natkin, *Reseaux de Petri Stochastiques*, PhD thesis, CNAM-PARIS, June 1980.

[14] W. D. Obal II and W. H. Sanders, "A framework for importance sampling based on stochastic activity networks," Technical Report PMRL 93-8, The University of Arizona, April 1993.

[15] R. A. Howard, *Dynamic Probabilistic Systems. Vol II: Semi-Markov and Decision Processes*, Wiley, New York, 1971.

[16] W. H. Sanders and J. F. Meyer, "A unified approach for specifying measures of performance, dependability, and performability," in *Dependable Computing for Critical Applications, Vol 4: of Dependable Computing and Fault-Tolerant Systems*, A. Avizienis and J. C. Laprie, editors, pp. 215–238, Springer Verlag, Vienna, 1991.

[17] S. Juneja and P. Shahabuddin, "Fast simulation of Markovian reliability/availability models with general repair policies," in *Proceedings of the Twenty-Second Annual International Symposium on Fault-Tolerant Computing*, pp. 150–159, Boston, Massachusetts, July 1992.

[18] E. E. Lewis and F. Böhm, "Monte Carlo simulation of Markov unreliability models," *Nuclear Engineering and Design*, vol. 77, pp. 49–62, January 1984.

[19] M. A. Crane and A. J. Lemoine, *An Introduction to the Regenerative Method for Simulation Analysis*, volume 4 of *Lecture Notes in Control and Information Sciences*, Springer-Verlag, New York, 1977.

[20] W. H. Sanders and R. S. Freire, "Efficient simulation of hierarchical stochastic activity network models," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 3, no. 2/3, pp. 271–300, July 1993.

[21] B. D. McLeod and W. H. Sanders, "Performance evaluation of N-processor time warp using stochastic activity networks," Technical Report PMRL 93-7, The University of Arizona, March 1993.

[22] W. H. Sanders, L. A. Kant, and A. Kudrimoti, "A modular method for evaluating the performance of picture archiving and communication systems," *Journal of Digital Imaging*, vol. 6, no. 3, pp. 172–193, August 1993.

[23] W. H. Sanders and L. M. Malhis, "Dependability evaluation using composed SAN-based reward models," *Journal of Parallel and Distributed Computing*, vol. 15, pp. 238–254, July 1992.

[24] W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, pp. 25–36, January 1991.

[25] M. A. Qureshi and W. H. Sanders, "Reward model solution methods with impulse and rate rewards: An algorithm and numerical results," Accepted for publication in *Performance Evaluation*.

[26] B. P. Shah, *Analytic Solution of Stochastic Activity Networks with Exponential and Deterministic Activities*, Master's thesis, The University of Arizona, 1993.

[27] J. E. Tvedt, *Matrix Representations and Analytical Solution Methods for Stochastic Activity Networks*, Master's thesis, The University of Arizona, 1990.

[28] A. M. Law and W. D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, New York, 1982.