From *Proc. AIAA Comp. in Aerospace 10 Conf.* , March 28-30, 1995, San Antonio, TX.

UltraSAN VERSION 3: ARCHITECTURE, FEATURES, AND IMPLEMENTATION *

W. H. Sanders†, W. D. Obal II‡, M. A. Qureshi†, and F. K. Widjanarko‡

† Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801
{whs, qureshi}@crhc.uiuc.edu

‡ Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721
{obal, krisnadi}@ece.arizona.edu

Abstract

This paper describes version three of *UltraSAN*, a software environment for the performance, dependability, and performability evaluation of computer networks and systems. The package makes use of both analytic- and simulation-based solution methods using a single, high-level, representation known as stochastic activity networks. Version three, just completed, contains significant enhancements, relative to the earlier versions, in model specification, construction, and solution.

Keywords: Stochastic Petri Nets, Stochastic Activity Networks, Model-Based Evaluation, Performance, Dependability, Performability.

1 Introduction

Development of tool-based techniques for performance/dependability evaluation, and in turn, the tools themselves, is an active research area. This paper describes version three of *UltraSAN*, one such tool with this aim. In *UltraSAN*, models are specified using a variant of stochastic Petri nets (SPNs) known as stochastic activity networks (SANs), and solution techniques include many analytic- and simulation-based approaches. Other SPN-based tools with this aim include, among many, DSPNexpress [1], FIGARO [2], GreatSPN [3], METASAN [4], RDPS [5], SPNP [6], SURF-2 [7], TimeNET [8], and *UltraSAN* [9].

Version three of *UltraSAN* has many enhancements relative to the initial version, which is described in [9]. In particular, with respect to the user interface, a parameterized model specification method that facilitates solution of models with multiple sets of input parameter values has been developed. Furthermore, three new analytical solvers, as well as a terminating simulation solver based on importance sampling, have been added to the package.

The first new analytic solver solves models that have deterministic as well as exponential activities. The two remaining new analytic solvers compute the probability distribution function, and mean, respectively, of reward accumulated over a finite interval. The importance sampling simulator provides, in many cases, an efficient solution for models that contain rare events which are significant relative to a measure in question. These enhancements have significantly improved *UltraSAN*'s usefulness to others, and illustrated its effectiveness as a test-bed for new modeling techniques. The package has now been at academic and industrial sites for about four years, and much has been learned from its use.

The remainder of the paper is organized as follows. Section II of this paper gives an overview of the modeling process, as embodied in *UltraSAN*, including a brief review of SANs and an example to illustrate the use of SANs in modeling a simple multiprocessor. The

*This work was supported in part by IBM, Motorola Satellite Communications, and US West Advanced Technologies.

intent here is to provide a high-level overview of the capabilities and features of the package, without cluttering the discussion with too many details of any single aspect. In Sections III–V, each step of the modeling process is described in detail. These sections provide more insight regarding the design choices that were made in each step of the process. Finally, Section VI summarizes the contributions and features of the tool, and suggests areas of future research and development.

2 Organization

The organization of *UltraSAN* follows directly from the goals of the package: the desire to create an environment (test-bed) for experimenting with new model construction and solution techniques, and the goal of facilitating large-scale evaluations of realistic systems using both analytic- and simulation-based techniques. The first goal dictates that the software be constructed in a modular fashion, so new construction and solution techniques can be easily added. The second goal dictates that many different solution techniques be available, since no single technique is ideal in all circumstances.

As shown in Figure 1, the package is constructed in a modular manner. All of the modules shown are coordinated by a central program called the control panel, that manages the interface between the user and the various modules, and provides basic consistency checking. In addition, the control panel offers several useful utilities, such as automatic documentation and convenient viewing of output data. The modules communicate via files which have well-defined formats. This configuration makes it easy to add new functionality, thus contributing to the goal of being a test-bed for developing new modeling techniques.

Figure 1 is divided into four sections, representing the four phases of the modeling process used in *UltraSAN*: *model specification*, *global variable assignment*, *study construction*, and *study solution*. These phases are briefly described here as an introduction to the package, with more elaborate descriptions of their functioning and underlying theory given in the following sections.

Model specification The first phase of the modeling process is model specification. In this phase, SANs are defined graphically using the SAN editor and then composed into a hierarchical model using the composed model editor. The performance variable editor is then used to define the performance measures of interest in terms of a reward structure at the SAN level. Once a model is specified, it is typically solved many times for differing values of the input parameters. To accommodate this activity, *UltraSAN* allows model specifications to be parameterized through the use of global variables in definitions of model components in the SAN editor.

Global variable assignment Global variable assignment, the second phase of the modeling process, is carried out by the study editor. In this phase, values are assigned to global variables. A model specification, together with the assignment of a single value to each global variable, is called an experiment. A group of related experiments is called a study. *UltraSAN* provides the study editor to make it easy to specify groups of related experiments.

Study construction The third phase of the modeling process, study construction, is the process of converting a specified model and a global variable assignment into a format amenable to computer solution via analytic or simulation-based techniques.

Analytical models are constructed by automatically generating a Markov process representation of the model. The size of the generated Markov process is determined in part by the definition of state. In standard stochastic Petri net models, the typical definition of the state space is the set of tangible reachable markings. Realistic models of modern systems usually have very large state spaces. This problem has been widely recognized, and various approaches have been devised to handle the explosive growth [10, 11]. *UltraSAN* uses an innovative largeness-avoidance approach called reduced base model construction [12]. This technique uses information about the performance measures and symmetries in the model to directly generate a stochastic process that is often much smaller than the set of reachable markings, and yet is sufficient to support the specified performance measures. Experience has shown that reduced base model construction is effective for medium to large systems that have replicated subsystems [13].

When the developed model is intended to be solved using simulation, *UltraSAN* does not generate a state-level representation. For this solution method, study construction is accomplished by linking the model description files generated in model specification and global variable assignment to the simulation library. The result is an executable simulation of the model. Essentially, simulation is accomplished by assigning events to activity completions and using the rules of SAN execution to determine the next state. The simulation is made more efficient through the use of data structures designed for reduced base model construction [14]. Symmetries in the model that allow lumping

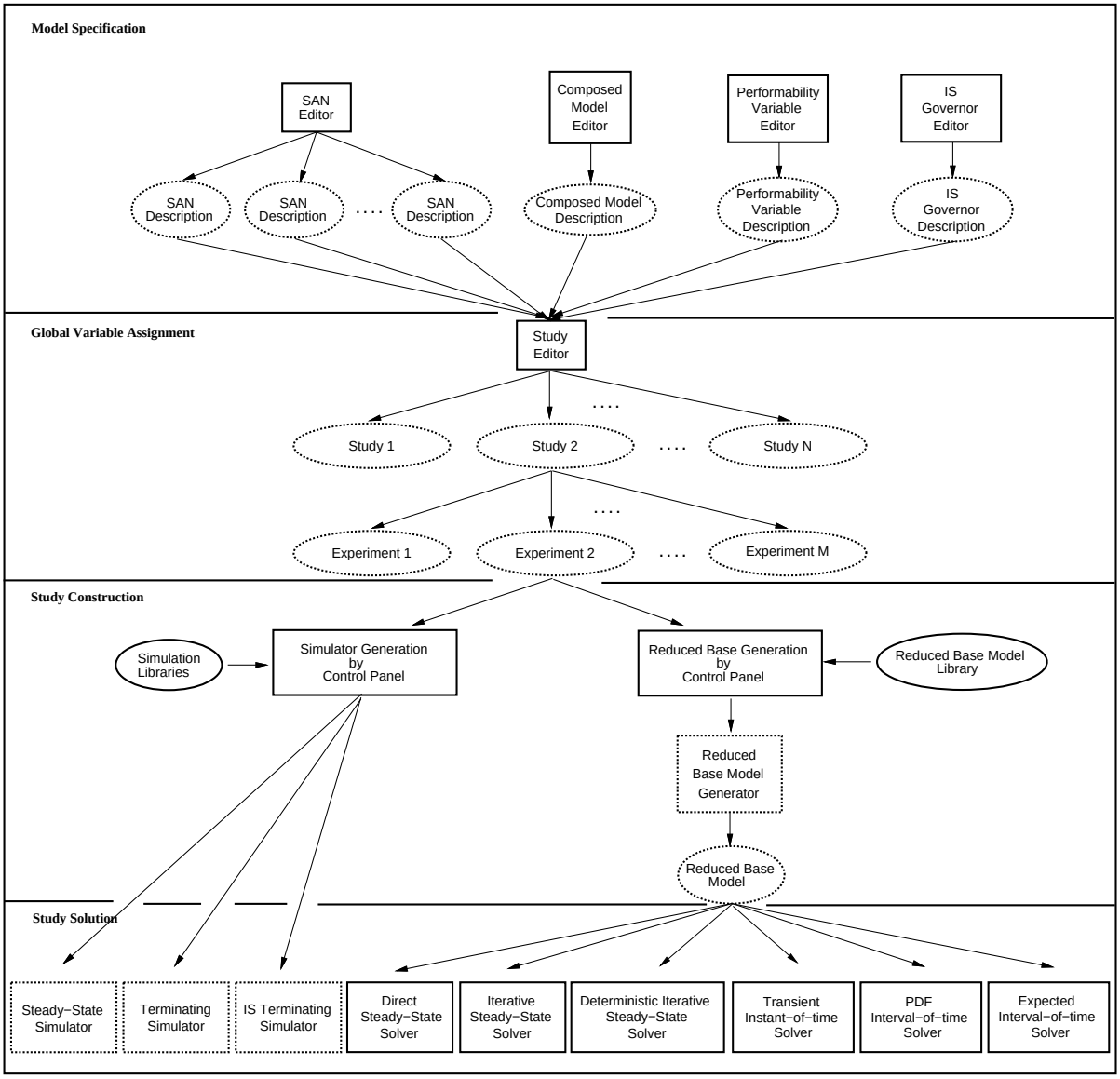

Figure 1: Organization of *UltraSAN*.

in state space construction are also useful for reducing the amount of future events list management that must be done for each state change.

Study construction is carried out using the control panel. After verifying that the model description is complete, the control panel builds the reduced base model generator for each experiment to be solved analytically, and builds executable simulators for experiments where simulation-based solution is desired. The control panel can then be used to run the reduced base model generator for all experiments that are to be solved analytically. In this case, the control panel can distribute the jobs to run on a user-specified list of other available machines. This "multiple runs" feature makes it easy to construct a study with little effort on the part of the user.

Study solution The last phase of the modeling process is study solution. If analytic solution is intended, the reduced base model generator was executed during study construction to produce the reduced base model. This representation of the model serves as input to the analytic solvers. Since a single solution method that works well for all models and performance variables is not available, *UltraSAN* utilizes a number of different techniques for solution of the reduced base model.

The package currently offers six analytic solution modules. The direct steady state solver uses LU-decomposition. The iterative steady-state solver uses successive overrelaxation (SOR). The deterministic iterative steady-state solver is used to solve models that have deterministic as well as exponential activities. It uses SOR and uniformization. The transient instant-of-time solver uses uniformization to evaluate performance measures at particular epochs. All of these solvers produce the mean, variance, probability density, and probability distribution function. The PDF interval-of-time solver uses uniformization to determine the distribution of accumulated reward over a specified interval of time. Similarly, the expected interval-of-time solver uses uniformization to compute the expected value of interval-of-time and time-averaged interval-of-time variables.

Three solution modules are available for simulation. The steady-state simulator uses the method of batch means to estimate the mean and variance of steady-state measures. The terminating simulator uses the method of independent replications to estimate the mean and variance of instant-of-time, interval-of-time, and time-averaged interval-of-time measures. The third simulator is useful when the model contains events that are rare, but significant with respect to the chosen performance variables. In this case, traditional simulation will be inefficient due to the rarity with which an event of interest occurs. For problems like this, *UltraSAN* includes an importance sampling simulation module [15] that estimates the mean of a performance measure. The importance sampling simulation module is flexible enough to handle many different heuristics, including balanced failure biasing, and is also applicable to non-Markovian models [16].

The study solution phase is also made easier by the control panel multiple runs feature. After study construction, the control panel may be used to run the reduced base model generator for all experiments that are to be solved analytically. After the state spaces for the experiments have been generated, all of the experiments can be solved by having the control panel distribute the solver or simulation jobs to run on all available machines.

These four phases, model specification, global variable assignment, study construction, and study solution are used repeatedly in the modeling process. As will be seen in the next section, all phases are supported by a graphical user interface, which facilitates specification of families of parameterized models, and solution by many diverse techniques. The following four sections detail the underlying theory and steps taken in each phase of the process.

3 Model Specification

Model specification in *UltraSAN* is done in three steps if importance sampling simulation is not done, and four if it is. The steps are, in order, specification of SAN models of the system, composition of the SAN models together to form a composed model, specification of the desired performance, dependability, and performability variables, and, if needed, specification of the "governor" to be used in importance sampling simulation. Completion of these three (or four, if importance sampling is used) steps results in a model representation that, once a value is assigned to each of its "global variables," can be used to generate an executable simulation or state-level model representation.

Input to each of the four editors is graphical and X-window based. In this section, we will briefly describe what must be specified in each of these steps, and how this specification is done in terms of the four editors. Detailed instructions concerning the operation of each editor cannot be given due to space limitations, but can be found in [17].

SAN model specification The SAN editor (see Figure 2) is used to create stochastic activity networks that represent the subsystems of the system being

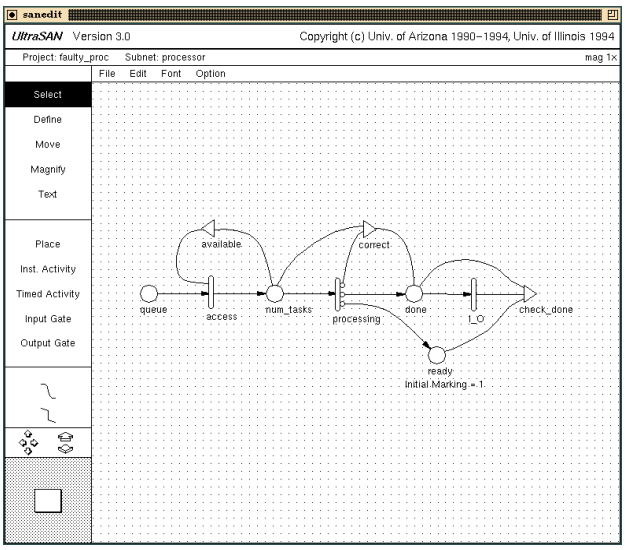

Figure 2: SAN *processor*.

studied. Before describing its operation, it is useful to briefly review the primitives that make up a SAN and how these are used to build system and component models. In particular, SANs consist of places, activities, and gates. These components are drawn and connected using the SAN editor, and defined through auxiliary editors that pop up on the top of the SAN editor.

The functions of the three model primitives are as follows. *Places* are as in Petri nets. Places hold tokens, and the number of tokens in each place is the *marking* of the SAN. Activities are used to model time delays in a system. *Timed* activities are used to represent delays whose time is important relative to a performance measure of interest. Associated with each timed activity is a marking dependent *activity time distribution*, a probability distribution function that describes the nature of the delay represented by the activity. If the time to complete an operation is deemed insignificant relative to performance measures of interest, then it can be modeled by an *instantaneous* activity.

Cases are used to model uncertainty about what happens upon completion of an activity. Each activity has one or more cases and a marking dependent *case distribution*. A case distribution is a probability distribution over the cases of an activity. Using cases, the possible outcomes of the completion of a task can be enumerated and the probability of each outcome can be specified. Gates serve to connect places and activities. *Input* gates provide flexibility in specifying enabling conditions for activities. *Output* gates provide flexibility in specifying how the marking is updated upon completion of an activity.

Table 1: Activity time distributions for the SAN *processor*.

Activity	*Dist*	*Rate*
I_O	expon	$GLOBAL\_D(io\_rate)$
access	expon	$GLOBAL\_D(access\_rate)$
processing	expon	$GLOBAL\_D(proc\_rate)$

Another important feature in the model specification tools is the incorporation of "global variable" support. In the SAN editor, global variables may be used in the definition of initial markings of places, activity time distributions, case distributions, gate predicates, and gate functions. Global variables can be thought of as constants (either integer or floating point) that remain unspecified during model specification, and are varied in different experiments. It is important to point out that the use of global variables is not limited to simple assignments. From its inception, *UltraSAN* has allowed SAN component definitions to make full use of the C programming language. The use of variables global to the model has been built on top of this capability. One ramification of this architecture is that the modeler is free to define SAN components in terms of complicated functions of one or more global variables or change the flow of control in a gate function, rather than simply assigning variables to component parameters.

For the sake of example, we introduce the following model of a faulty multiprocessor system. This model also serves as an example in the *UltraSAN* reference manual [17], and appears in a somewhat simpler form in [18]. The example is used throughout the rest of this paper to demonstrate the new capabilities and features of *UltraSAN*. The multiprocessor system is divided into two distinct submodels processor and buffer. Due to lack of space we are restrained to omit the SAN, *buffer*, for the submodel buffer. The interested readers are encouraged to consult reference [17]. Figure 2 shows the SAN for the submodel processor. Tables 1–3 show the definitions of the SAN components.

In the example system, jobs arrive for processing according to a Poisson process. Jobs arriving when the buffer is full are rejected. This portion of the multiprocessor system is represented by the SAN *buffer* (for details see reference [17]). Jobs entering the system are scheduled in FIFO order to available processors. If there are multiple available processors, each

processor is equally likely to receive the job. Each processor completes jobs at rate *proc_rate*. In addition, the processors in the system have a special feature that allows them to "piggyback" jobs. That is, while busy with one job, a processor may be scheduled for a second job. When a second job is piggybacked on the first job in this manner, the processor can complete both jobs in the time it normally takes for a single job. Both jobs are completed simultaneously at the single job processing rate, *proc_rate*. As a result of this special capability, a processor is available if it is idle or busy with a single job. Only one job can be piggybacked, so if a processor is busy with two jobs, it is unavailable. The processing delay is modeled by timed activity "processing" and the admission of a second job is modeled by timed activity "access" and input gate "available."

Furthermore, in the rush to market their "two-for-one" design, the system designers overlooked a fault in the dual job processing control logic. Therefore, when two jobs are processed at the same time they may interfere with each other. In this case, one or both of the jobs may be processed incorrectly. The probability that both jobs are correctly processed is *ok_prob*, while the probability that one of the jobs is in error is *one_error_prob*. The probability that both jobs are processed incorrectly is $1.0 - ok\_prob - one\_error\_prob$. Fortunately, the errors are detected (with probability one) upon completion of the two jobs. Jobs processed incorrectly are immediately returned to the processor for another try. Finally, if a processor is working on a single job, correct completion of the job occurs with probability one. These three possible outcomes and their probabilities are modeled through the three cases on activity "processing." As can be seen from Table 2, the case probabilities are marking dependent, since the erroneous processing events do not occur unless two jobs have been piggybacked. Jobs processed correctly are sent to output. Output from jobs is processed with rate *io_rate*, and the processor may not resume until I/O is complete. The I/O constraint on processor availability is modeled by activity "I/O," places "done" and "ready," and output gate "check_done."

After SAN models of the subsystems have been specified, the composed model editor is used to create the system model by combining the component SANs by using replicate and join operations.

Composed Model Specification In the second step of model specification, the composed model editor is used to create a graphical description of the composed model. A composed model consists of one or more SANs connected to each other through common places via replicate and join operations.

Table 2: Activity case probabilities for the SAN *processor*.

Activity	*C*	*Probabilities*
processing	1	*if (MARK(num_tasks) == 1)* *return(1.0);* *else return(GLOBAL_D(ok_prob));*
	2	*if (MARK(num_tasks) == 1)* *return(ZERO);* *else* *return(GLOBAL_D(one_error_prob));*
	3	*if (MARK(num_tasks) == 1)* *return(ZERO);* *else* *return(1.0 − GLOBAL_D(ok_prob)* *− GLOBAL_D(one_error_prob));*

Table 3: Gate definitions for the SAN *processor*.

Gate	*Definition*
available	Predicate *MARK(num_tasks) < 2*
	Function */* do nothing */* *;*
check_done	*/* when I/O done, reset ready */* *if (MARK(done) == 0)* *MARK(ready) = 1;*
correct	*/* put one or two tasks in done, clear num_tasks */* *MARK(done) = MARK(num_tasks) + 1;* *MARK(num_tasks) = 0;*

The replicate operation creates a user-specified number of replicas of a SAN. The user also identifies a subset of the places in the SAN as common. The places not chosen to be common will be replicated so they will be distinct places in each replica. The common places are not replicated, but are shared among the replicas. Each replica may read or change the marking of a common place. The join operation connects two or more SANs together through a subset of the places of the SANs that are identified as common.

As in the SAN editor, auxiliary editors pop up over the composed model editor to allow one to specify the details of replicate and join operations. For example, the replicate editor lets one specify the number of replicas and the set of places to be held common among the replicas.

The composed model for the multicomputer system is shown in Figure 3. As can be seen from the figure, a

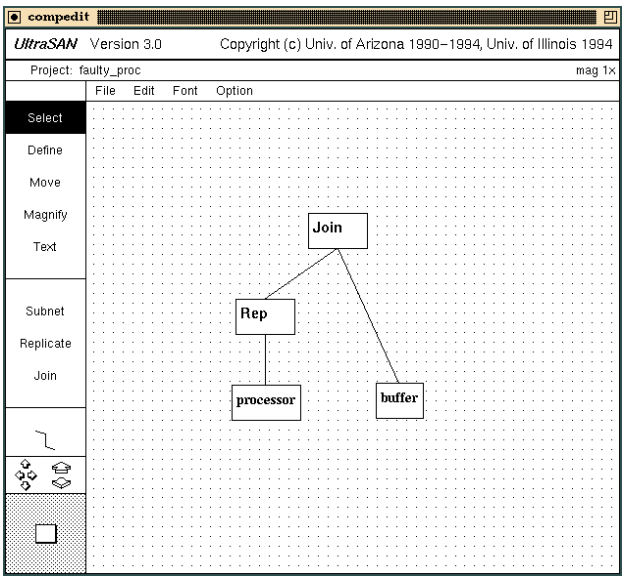

Figure 3: Composed model for multiprocessor, as specified in the composed model e ditor.

model of a multiprocessor is composed by joining any number of replicas of the SAN *processor* with the SAN *buffer*. The place *queue* is an example of a common place. It is held common at the system level, where it can be accessed by both submodels, so the submodel buffer can update it on an arrival while the submodel processor can update it on a job completion.

Performance, Dependability, and Performability Variable Specification The third step in model specification is to use the performability variable editor to build reward structures from which the desired performance measures may be derived. In short, to specify a performability variable one defines a reward structure and chooses a collection policy. Variables are typed and often referred to according to the collection policy. The three types of variables supported in *UltraSAN* are "instant-of-time," "interval-of-time," and "time-averaged interval-of-time." *Instant-of-time* variables are obtained by examining the value of the associated reward structure at a particular time t. *Interval-of-time* variables measure the reward accumulated in an interval $[t, t+l]$. *Time-averaged interval-of-time* variables divide the accumulated reward by the length of the interval. For a thorough discussion of the concept and application of these variables, see [19].

For simulation models, another type of variable is also available. These *activity variable* measures the time between completions of activities. Using an activity variable, one can measure the mean and variance of the time until first completion, the time between the tenth and twentieth completions, etc., using the terminating simulator. The steady-state simulator can be used to measure the mean and variance of the time between consecutive completions as time goes to infinity.

Importance Sampling Governor Specification If importance sampling is to be used in simulation, an importance sampling governor must be specified. The importance sampling governor [15] is used to specify importance sampling heuristics. The governor is similar to a finite state machine, where each state corresponds to a different set of activity definitions. The governor changes state based on the evolution of the simulation. Each state has a transition function consisting of a sequence of Boolean functions of the markings of the SANs in the composed model (called predicates), paired with a governor state name. When the marking of the model is updated, the transition function for the current governor state is checked. The first predicate that holds causes the governor to change to the associated governor state. The utility of the governor is that dynamic importance sampling heuristics may be experimented with, where the bias on the model is altered according to the evolution of a sample path. Additional flexibility is available through the use of marking-dependent governor state definitions.

After the model is specified, the next step is to assign values to the variables using the study editor.

4 Study Construction

Depending on model characteristics, the intended method for solving experiments in a study can be either analytic or simulation. Specifically, analytic solution is the choice when activity time distributions are exponential, reactivated often enough to ensure that their rates depend only on the current state, and the state space is not too large relative to the capacity of the machine. In addition, mixes of exponential and deterministic activities can be handled, as long as no more than one concurrently enabled deterministic activity is enabled in any reachable stable marking. Otherwise, simulation must be used.

If analysis is the intended solution method, the control panel links the experiment description files (generated and compiled during the model specification and global variable assignment phases) with the reduced base model construction library to create reduced base model generators for each experiment. As depicted in Figure 1, the constructed reduced base model generators are executed to generate "reduced base models."

Each *reduced base model* is the state-level representation of an experiment. The state-level representation consists of a set of states, either a rate (if the transition is caused by an exponential activity) or a probability (if the transition is caused by a deterministic activity) of transition between each pair of states, and a set of impulse and rate rewards for each state. (The impulse reward for a state is the impulse reward associated with the activity that completed to cause a transition to the state.) The notion of state employed is variable and depends on the structure of the composed model. One can think of the state as a set of impulse and rate rewards plus a *state tree* [12], where each node on the state tree corresponds in type and level with a node on the composed model tree. Furthermore, each node in a state tree has associated with it a subset of common places of the corresponding node in the composed model diagram. These places are those that are common at the node, but not at its parent node.

From a state, other states can be generated by executing each activity that may complete in the state, generating a new state tree and impulse reward corresponding to each possible next state that may be reached. For each possible next state, a non-zero rate or probability from the original state to the reached state is added to the list of rates to other states from the originating state. If the reached state is new, then it is added to the list of states which need to be expanded. Generation of the reduced base model proceeds by selecting states from the list of unexpanded states and repeating the above operations. The procedure terminates when there are no more states to expand. The resulting state-level representation serves as input to the analytic solvers, which determine the probabilistic nature of the selected performability variables.

If simulation is the intended solution method, the control panel links experiments (model descriptions), with the simulation library to generate a simulation program. As depicted in Figure 1, two types of simulation programs can be generated: a *steady state simulator* to solve long-run measures of performability variables, and a *terminating simulator* to solve instant-of-time or interval-of-time performability variables.

Unlike the analytic solvers, which require a reduced base model, the simulators execute directly on the assigned model specifications. Both simulators exploit the hierarchical structure and symmetries introduced by the replicate operation in a composed SAN-based reward model to reduce the cost of future event list management. More specifically, multiple future events lists are employed, one corresponding to each leaf on the state tree or, in other words, one corresponding to each set of replica submodels in a particular marking. These multiple future event lists allow the simulators to operate on "compound events," corresponding to a set of enabled replica activities, rather than on individual activities. By operating on compound events, rather than individual activities, the number of checks that must be made upon each activity completion is reduced. The details and precise algorithm can be found in [14]. This algorithm is the basis of the state-change mechanism for both steady-state and terminating simulators.

The next section details how the analytic and simulation solvers are used to determine the probabilistic behavior of the desired performability variables.

5 Study Solution

Solution of a set of base models, corresponding to one or more experiments within a study, is the final step in obtaining values for a set of performability variables specified for a model. The solution can be either by analytic (numerical) means or simulation, depending on model characteristics and choices made in the construction phase of model processing. This section details the solution techniques employed in *UltraSAN* to obtain analytic and simulative solutions.

5.1 Analytic Solution Techniques

Analytic solvers can provide steady-state as well as transient solutions for the reduced base model generated during study construction. Figure 1 illustrates the six solvers available for analytic solutions. Three of the solvers, specifically, the *direct steady-state solver*, *iterative steady-state solver*, and *deterministic iterative steady-state solver* provide steady-state solutions for the instant-of-time variables. The other three solvers, the *transient instant-of-time solver*, *PDF interval-of-time solver* and *expected interval-of-time solver*, provide solutions for transient instant-of-time and interval-of-time variables. Except for the two interval-of-time solvers, each solver calculates the mean, variance, probability density function, and probability distribution function of the variable(s) it is intended to solve for. The PDF interval-of-time solver calculates the probability distribution function of reward accumulated during a fixed interval of time $[0, t]$. Likewise, the expected interval-of-time solver calculates the expected reward accumulated during a fixed interval of time. All solvers use a sparse matrix representation appropriate for the solution method employed.

The direct steady-state solver uses LU decomposition [20] to obtain the steady-state solution for instant-of-time variables. As with all steady-state solvers, it is applicable when the generated reduced

base model is Markovian, and contains a single, irreducible, class of states. LU decomposition factors the transition matrix Q into the product of a lower triangular matrix L and an upper triangular matrix U such that $Q = LU$. Crout's algorithm [21] is incorporated to compute L and U in a memory efficient way. Pivoting in the solver is performed by using the improved generalized Markowitz strategy, as discussed by Osterby and Zlatev [22]. This strategy performs pivoting on the element for which the minimum fill-in is expected, subject to constraints on the magnitude of the pivot. Furthermore, the matrix elements that are less than some value (called the drop tolerance) are made zero (i.e., *dropped*) to retain the sparsity of the matrix during decomposition. In the end, iterative refinement is used to correct the final solution to a user-specified level of accuracy.

The iterative steady-state solver uses successive-overrelaxation to obtain steady-state instant-of-time solutions. Like the direct steady-state solver, it acts on the Markov process representation of a reduced base model. However, being an iterative method, it avoids fill-in problems inherent to the direct steady-state solver and therefore requires less memory. However, unlike LU decomposition, SOR is not guaranteed to converge to the correct solution for all models. In practice however, this does not seem to be a problem. The implementation solves the system of equations directly, without first adding the constraint that the probabilities sum to one, and normalizes only at the end, and as necessary to keep the solution vector within bounds, as suggested in [23]. Selection of the acceleration factor is left to the user, and defaults to a value of 1 (resulting in a Gauss-Seidel iteration).

The deterministic iterative steady-state solver provides a steady-state instant-of-time solution for models that contain a mix of exponential and deterministic timed activities. The method used in the solver was first developed for deterministic stochastic Petri nets [24, 25] with one concurrently-enabled deterministic transition and later adapted to SANs in which there is no more than one concurrently-enabled deterministic activity [26]. In this solution method, a Markov chain is generated for marking in which a deterministic activity is enabled, representing the states reachable from the current state due to completions of exponential activities, until the deterministic activity either completes or is aborted. The solver makes use of successive-overrelaxation and uniformization to obtain the solution.

The transient instant-of-time solver and expected interval-of-time solver both use uniformization to obtain solutions. Uniformization works well when the time point of interest is not too large, relative to the highest rate activity in the model. The required Poisson probabilities are calculated using the method by Fox and Glynn [27] to avoid numerical difficulties.

The PDF interval-of-time solver [28] also uses uniformization to calculate the probability distribution function of the total reward accumulated during a fixed interval $[0, t]$. This solver is unique among the existing interval-of-time solvers in that it can solve for reward variables containing both impulse and rate rewards. The solver calculates the PDF of reward accumulated during an interval by conditioning on possible numbers of transitions that may occur in an interval, and possible sequences of state transitions (paths), given a certain number of transitions have occurred. Since the number of possible paths may be very large, an implementation that considers all possible paths would require a very large amount of memory. This problem is avoided in *UltraSAN* by calculating a bound on the error induced in the desired measure by not considering each path, and discarding those with acceptably small (user specified) errors. In many cases, as illustrated in [28], selective discarding of paths can dramatically reduce the space required for a solution while maintaining acceptable accuracy. Calculation of the PDF of reward accumulated, given that a particular path was taken, can also pose numerical difficulties. In this case, the solver makes selective use of multiprecision math to calculate several summations.

5.2 Simulative Solutions

As shown in Figure 1, three simulation-based solvers are available in *UltraSAN*. All three simulators exploit the hierarchical structure and symmetries introduced by the replicate operation in a composed SAN-based reward model to reduce the cost of future event list management. More specifically, multiple future events lists are employed, one corresponding to each leaf on the state tree or, in other words, one corresponding to each set of replica submodels in a particular marking. These multiple future event lists allow the simulators to operate on "compound events," corresponding to a set of enabled replica activities, rather than on individual activities. By operating on compound events, rather than individual activities, the number of checks that must be made upon each activity completion is reduced. The details and a precise algorithm can be found in [14]. This algorithm is the basis of the state-change mechanism for all three simulators.

The steady-state simulator uses an iterative batch means method to estimate the means and/or variances of steady-state instant-of-time variables. The user se-

lects a confidence level and a maximum desired half-width for the confidence interval corresponding to each variable, and the simulator executes batches until the confidence intervals for every performability variable satisfy the user's criteria. The terminating simulation solver estimates the means and/or variances for performability variables at a specific instant of time or over a given interval of time. It uses an iterative replication method to generate confidence intervals that satisfy the criteria specified by the user.

The third simulation-based solver is a terminating simulator that is designed to use importance sampling to increase the efficiency of simulation-based evaluation of rare event probabilities. The importance sampling terminating simulator estimates the mean of transient variables that are affected by rare events by simulating the model under the influence of the governor (described in Section 3), so that the interesting event is no longer rare. The simulator then compensates for the governor bias by weighting the observations by the likelihood ratio. The likelihood ratio is calculated by evaluating the ratio of the likelihood that a particular activity completes in a marking, and that the completion of this activity results in a particular next marking, in the original model to the likelihood of the same event in the governed model. The likelihood calculation is based on the work in [29], adapted for SAN simulation and implemented in *UltraSAN* [15]. It works for models where all activities have closed-form activity time distribution functions. The probability distribution function is needed to handle the reactivations of activities when their definitions are changed by the governor.

6 Conclusion

We have described version three of *UltraSAN*, a software package designed to facilitate the specification, construction, and solution of performability models represented as composed stochastic activity networks. Earlier versions of the package have been in use for a period of about four years at several industrial and academic sites. Feedback from these users, and the development of new solution methods, has motivated the development of the new version of the software. As described herein, version three embodies significant enhancements, relative to the earlier versions, in both model solution capabilities and ease of use.

In particular, in the area of model specification, the user interface has been unified under the control panel, which provides a simpler interface to the package than in earlier versions, where the user was asked to run the various editors and solvers from the UNIX command line. In addition, several new utilities are also available from the control panel, including an automatic documentation facility. Furthermore, the generation and solution of large numbers of production runs for a model has been simplified through the introduction of support for model specifications parameterized by global variables. A large part of the implementation of support for parameterized models is the newly added study editor, which expedites the assignment of values to the global variables parameterizing the model. Finally, using the control panel, model construction and solution can be distributed across a network of workstations and carried out in parallel.

In the area of model construction, support has been added to generate reduced base models for composed SANs that contain a mix of exponential and deterministic timed activities. Three new solvers have also been added to *UltraSAN*. The first computes the probability distribution function of the reward accumulated over an interval of time. The second computes the expected value of transient interval-of-time variables. The third solver computes expected value, variance, probability density function, and probability distribution function of steady-state instant-of-time variables in SAN models with deterministic and exponential activities. Moreover, an importance sampling terminating simulator has also been added to speed up the simulation of systems with rare events.

These enhancements have resulted in a tool that is easier to use, contains more model solution options, and, as evidenced by its application (e.g., [13, 30, 31, 32]), capable of evaluating the performance, dependability, and performability of realistic, real-world, systems. In addition to this capability, the tool serves as a test-bed in the development of new model construction and solution algorithms.

Acknowledgments

The success of this project is the result of the work of many people, in addition to the authors of this paper. In particular we would like to thank Burak Bulasaygun, Bruce McLeod, Aad van Moorsel, Bhavan Shah, and Adrian Suherman, for their contributions to the tool itself, and the members of the Performability Modeling Research Lab who tested the pre-release versions of the software. We would also like to thank the users of the released versions who made suggestions that helped improve the software. In particular, we would like to thank Steve West of IBM Corporation, Dave Krueger, Renee Langefels, and Tom Mihm of Motorola Satellite Communications, Kin Chan, John

Chang, Ron Leighton and Kishore Patnam of US West Advanced Technologies, Lorenz Lercher of the Technical University of Vienna, and John Meyer, of the University of Michigan in this regard.

References

[1] C. Lindemann, "DSPNexpress: A software package for the efficient solution of deterministic and stochastic Petri nets," in *Proceedings of the Sixth International Conference on Modelling Techniques and Tools for Computer Systems Performance Evaluation*, (Edinburgh, Great Britain), pp. 15–29, 1992.

[2] M. Bouissou, H. Bouhadana, M. Bannelier, and N. Villatte, "Knowledge modeling and reliability processing: The FIGARO language and associated tools," in *Proceedings of the Twenty-Third Annual International Symposium on Fault-Tolerant Computing*, (Toulouse, France), pp. 680–685, June 1993.

[3] G. Chiola, "A software package for analysis of generalized stochastic Petri net models," in *Proceedings of the International Workshop on Timed Petri Net Models*, (Torino, Italy), pp. 136–143, July 1985.

[4] W. H. Sanders and J. F. Meyer, "METASAN: A performability evaluation tool based on stochastic activity networks," in *Proceedings of the ACM-IEEE Computer Society 1986 Fall Joint Computer Conference*, pp. 807–816, 1986.

[5] G. Florin, P. Lonc, S. Natkin, and J. M. Toudic, "RDPS: A software package for the validation and evaluation of dependable computer systems," in *Proceedings of the Fifth IFAC Workshop Safety of Computer Control Systems (SAFECOMP'86)* (W. J. Quirk, ed.), (Pergamon, Sarlat, France), pp. 165–170, October 1986.

[6] G. Ciardo, J. Muppala, and K. S. Trivedi, "SPNP: Stochastic Petri net package," in *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models*, (Kyoto, Japan), pp. 142–151, December 1989.

[7] C. Béounes, M. Aguéra, J. Arlat, S. Bachmann, C. Bourdeau, J.-E. Doucet, K. Kanoun, J.-C. Laprie, S. Metge, J. M. de Souza, D. Powell, and P. Spiesser, "SURF-2: A program for dependability evaluation of complex hardware and software systems," in *Proceedings of the Twenty-Third International Symposium on Fault-Tolerant Computing*, (Toulouse, France), pp. 668–673, June 1993.

[8] R. German, C. Kelling, A. Zimmermann, and G. Hommel, "Timenet: A toolkit for evaluating non-markovian stochastic petri-nets," tech. rep., Technische Universitat Berlin, Germany, 1994. Submitted for publication.

[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, pp. 69–80, September 1991.

[10] M. Siegle, "Reduced Markov models of parallel programs with replicated processes," in *Proceedings of the Second Euromicro Workshop on Parallel and Distri buted Processing*, (Malaga), pp. 1–8, January 1994.

[11] K. S. Trivedi, B. R. Haverkort, A. Rindos, and V. Mainkar, "Techniques and tools for reliability and performance evaluation: Proble ms and perspectives," in *Computer Performance Evaluation Modelling Tools and Techniques* (G. Haring and G. Kotsis, eds.), (New York), pp. 1–24, Springer-Verlag, 1994.

[12] 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.

[13] 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.

[14] W. H. Sanders and R. S. Freire, "Efficient simulation of hierarchical stochastic activity network models," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 3, pp. 271–300, July 1993.

[15] W. D. Obal II and W. H. Sanders, "Importance sampling simulation in *UltraSAN*," *Simulation*, vol. 62, pp. 98–111, February 1994.

[16] W. D. Obal II and W. H. Sanders, "An environment for importance sampling based on stochastic activity net works," in *Proceedings of the 13th Symposium on Reliable Distributed Systems*, (Dana Point, California), pp. 64–73, October 1994.

[17] Center for Reliable and High Performance Computing, Coordinated Science Laboratory, University of Illinois, *UltraSAN Reference Manual*, 1995.

[18] J. F. Meyer and W. H. Sanders, "Specification and construction of performability models," in *Second International Workshop on Performability Modelling of Computer and Communication Systems*, (Le Mont Saint-Michel, France), June 1993. Paper number 15.

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

[20] J. E. Tvedt, "Matrix representations and analytical solution methods for stochastic activity networks," Master's thesis, The University of Arizona, 1990.

[21] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1988.

[22] O. Osterby and Z. Zlatev, *Lecture Notes in Computer Science: Direct Methods for Sparse Matrices*. Heidelberg: Springer-Verlag, 1983.

[23] U. R. Krieger, B. Müller-Clostermann, and M. Sczittnick, "Modeling and analysis of communication systems based on computational m ethods for Markov chains," *IEEE Journal on Selected Areas in Communications*, vol. 8, no. 9, pp. 1630–1648, 1990.

[24] C. Lindemann, "An improved numerical algorithm for calculating steady-state solutions of deterministic and stochastic Petri net models," in *Proceedings of the Fourth International Workshop on Petri Nets and Performance Models*, (Melbourne, Australia), pp. 176–185, 1991.

[25] M. A. Marsan and G. Chiola, "On Petri nets with deterministic and exponentially distributed firing times," in *Advances in Petri Nets 1986* (G. Rozenberg, ed.), Lecture Notes in Computer Science 266, pp. 132–145, Springer, 1987.

[26] B. P. Shah, "Analytic solution of stochastic activity networks with exponential and deterministic activities," Master's thesis, The University of Arizona, 1993.

[27] B. L. Fox and P. W. Glynn, "Computing Poisson probabilities," *Communications of the ACM*, vol. 31, pp. 440–445, 1988.

[28] M. A. Qureshi and W. H. Sanders, "Reward model solution methods with impulse and rate rewards: An algorit hm and numerical results," *Performance Evaluation*, vol. 20, pp. 413–436, August 1994.

[29] 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*, (Newcastle upon Tyne, United Kingdom), pp. 491–498, June 1990.

[30] L. M. Malhis, W. H. Sanders, and R. D. Schlichting, "Analytic performability evaluation of a group-oriented multicast protocol," Tech. Rep. PMRL 93-13, The University of Arizona, June 1993. Submitted for publication.

[31] B. D. McLeod and W. H. Sanders, "Performance evaluation of N-processor time warp using stochastic activity networks," Tech. Rep. PMRL 93-7, The University of Arizona, March 1993. Submitted for publication.

[32] 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, pp. 172–193, August 1993.