

## A MODELING TOOL FOR A NEW DEFINITION OF STOCHASTIC ACTIVITY NETWORKS<sup>\*</sup>

M. ABDOLLAHI AZGOMI<sup>\*\*</sup> AND A. MOVAGHAR

Dept. of Computer Engineering, Sharif University of Technology, Tehran, I. R. of Iran  
Email: azgomi@mehr.sharif.edu

**Abstract**– Stochastic activity networks (SANs) are a powerful and flexible extension of Petri nets. Modeling and analysis with Petri nets and their extensions need a software tool to help to construct and analyze the models. The original definition of SANs has been used as a modeling formalism in several tools. In this paper, we introduce a modeling tool called SharifSAN, which is based on a new definition of SANs. This modeling tool is useful for both the verification and performance evaluation of systems. The functional aspects of a system are verified by model checking branching temporal logic formulas, whereas its performance aspects are evaluated using some analytic and simulative methods. The nondeterministic setting of SANs is used for verification, while their stochastic setting is employed for performance evaluation. In this paper, after a brief introduction to a new definition of SANs, we introduce the SharifSAN tool and its verification and evaluation techniques. We will also present some examples of its application.

**Keywords**– Modeling tool, Petri nets, stochastic activity networks, verification, analysis, simulation

### 1. INTRODUCTION

Stochastic activity networks (SANs) [1], introduced in 1984, are a stochastic extension of Petri nets [2]. SANs are more powerful and flexible than most other stochastic extensions of Petri nets including stochastic Petri nets (SPNs) [3] and Generalized Stochastic Petri nets (GSPNs) [4].

Modeling and analysis with Petri nets and their extensions needs a software tool to help to construct and analyze the model. The original definition of SANs [1] has been used as a modeling formalism in several modeling tools such as METASAN [5], UltraSAN [6] and Möbius [7]. All of these tools are intended for the evaluation of operational aspects (such as performance, dependability or performability) of systems and not for the verification of functional aspects (i.e. correctness).

In this paper, we introduce a modeling tool called SharifSAN, which is based on a new definition of SANs [8]. SharifSAN provides features for both verification and performance evaluation. Using this tool, one can verify the correctness of the specification of a concurrent and reactive system and then evaluate its operational aspects on the same model. By having the SAN model of a system, one can verify its correctness using the nondeterministic setting of the model and then evaluate its performance using the stochastic setting of the same model. To verify a system, an equivalent transition system will be obtained and the correctness of a few user-defined branching temporal logic (BTL) [9, 10] formulas will be checked automatically. BTL formulas will be used as a formal method to define the specification of a system.

For performance evaluation, SharifSAN is able to solve a finite-state Markovian model using a numerical solver and an infinite-state or non-Markovian model using a discrete-event simulator. SharifSAN provides an integrated development environment (IDE) for graphical construction, translation, verification, animation, simulation and analytic solution of SAN models.

---

<sup>\*</sup>Received by the editors November 10, 2003 and in final revised form December 12, 2004

<sup>\*\*</sup>Corresponding author

This paper is organized as follows. Section 2 gives a brief introduction to a new definition of SANs. In Section 3, the use of SANs for verification is described and some features of SharifSAN for this purpose are mentioned. In Section 4, the use of SANs for performance evaluation is studied and some related features of SharifSAN are presented. In Section 5, the software architecture of SharifSAN is illustrated. Finally, in Section 6, some examples of the application of this tool are presented.




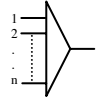
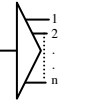
## 2. A NEW DEFINITION OF STOCHASTIC ACTIVITY NETWORKS

In a new definition of stochastic activity networks (SANs) [8], a SAN model is composed of the following elements:

1. Place: Is similar to a place of Petri nets and is shown as a circle.
2. Timed activity: Timed activities represent activities of the modeled system whose durations impact the system's ability to perform. A timed activity has  $m$  inputs and  $n$  outputs, where  $m+n > 0$ . Each input can be a place or an input gate and each output can also be a place or an output gate. To each timed activity is associated an activity distribution function, an enabling rate function (i.e. a computable partial function, which is the activity execution speed), and an  $n$ -ary computable predicate called the reactivation predicate.
3. Instantaneous activity: Instantaneous activities represent system activities, which, relative to the performance variable in question, are completed in a negligible amount of time. An instantaneous activity has  $m$  inputs and  $n$  outputs. To each instantaneous activity is associated a case probability function (i.e. a computable partial function that returns a value in  $[0, 1]$ ). Case probabilities associated with instantaneous activities permit the realization of spatial uncertainty. Uncertainty about which activities are enabled in a certain state is realized by case probabilities associated with intervening instantaneous activities.
4. Input gate: Gates are introduced to permit greater flexibility in defining enabling and completion rules. An input gate has a finite set of inputs and one output. To each such input gate is associated an  $n$ -ary computable predicate called the enabling predicate, and a computable partial function called the input function.
5. Output gate: An output gate has a finite set of outputs and one input. To each such output gate is associated a computable function called the output function.

The graphical representation of the elements of SANs is shown in Table 1.

Table 1. Graphical notations of SAN elements

| Element            | Place   | Timed activity  | Instantaneous activity  | Input gate   | Output gate   |
|--------------------|---|---|---|--|---|
| Graphical notation |  |  |  |  |  |

Structurally, a SAN model is an interconnection of a finite number of primitives, subject to the following connection rules:

1. Each input of an input gate is connected to a unique place and the output of an input gate is connected to a single activity.
2. Different input gates of an activity are connected to different places.
3. Each output of an output gate is connected to a unique place and the input of an output gate is connected to a single activity.
4. Different output gates of an activity are connected to different places.
5. Each place and activity is connected to some input or output gates.

In order to facilitate the use and to increase the understandability of SAN models, the following conventions are used in their graphical representation:

- An input gate with one simple place input, enabling predicate  $e(x): x \geq 1$ , and input function  $f$  such that  $f(x) = x-1$ , is shown as a directed line from its input to its output.
- An output gate with one simple place output and output function  $f$  such that  $g(x) = x+1$ , is shown as a directed line from its input to its output.
- An input gate with one input, enabling predicate  $e(x): x = 0$ , and identity input function is shown as a directed line from its input to its output crossed by two short parallel lines.

A new definition of SANs is based on a unified view of the system in three settings: nondeterministic, probabilistic, and stochastic. In a nondeterministic setting, nondeterminacy and parallelism are represented in a nondeterministic manner. In a probabilistic setting, nondeterminacy is specified probabilistically, but parallelism is treated nondeterministically. In a stochastic setting, both nondeterminacy and parallelism are modeled probabilistically [8].

The nondeterministic setting of SANs is referred to as activity networks, which are nondeterministic models for representing concurrent and reactive systems. Application of this setting is on the analysis of logical aspects or verification of concurrent and reactive systems. Disregarding the timing related information of the model and viewing it in a nondeterministic setting accomplish this. The activity network model is then translated into a transition system and verification is done.

For evaluating the operational aspects of systems such as performance, dependability and performability, the stochastic setting of SANs is used. In this setting, both nondeterminacy and parallelism are represented probabilistically [8]. Based on the probability distribution of timed activities will result in Markovian or non-Markovian SAN models. Application of the of the other setting of SANs, namely, probabilistic setting, is on probabilistic verification.

The original definition of SANs, as appeared in 1984, includes extra primitives, called "cases," for modeling nondeterminacy, which, with the new definition, can equivalently be replaced by some instantaneous activities. A model based on the original definition of SANs must be checked for well-behavedness. The well-behavedness check is, in general undecidable and is computationally complex for most models. Some solutions, have recently have recently been proposed to alleviate this problem [11]. However, in a new definition of SANs, no such check will ever be necessary. Because the syntax of the model has been defined free from well-behavedness checking. For more information about the differences between these two definitions of SANs and their formal definitions and properties, please see [1, 8, 12].

### 3. VERIFICATION WITH ACTIVITY NETWORKS

As we mentioned in the previous section, the nondeterministic setting of SANs referred to as activity networks, is used for the verification of concurrent and reactive systems. The verification system of SharifSAN provides the following functionalities:

- a) Graphical model construction; b) Model translation and compilation; c) Full state space generation;
- d) Specifying the system properties as BTL formulas; e) Transformation of an activity network to a transition system, f) Translation of BTL formulas to expression trees; g) Model checking.

More specifically, the above steps are described in the following subsections.

#### **a) Graphical model construction**

SharifSAN provides a graphical user interface (GUI) for edition, checking the syntax and semantics, interpretation, state space generation and verification activity network models. SAN editor is a component of this GUI which provides features for the graphical edition of SAN models. Each SAN model is composed of some primitives of SANs such as place, activity or gate, and connected by arcs. A view of the user interface of SharifSAN while a SAN model is being edited is shown in Fig. 1.

#### **b) Model translation and compilation**

The input/output function of the input/output gates of SANs are general computable functions. As an example, see the definition of an input gate in Fig. 2. The same is correct for predicates, enabling rate functions and case probability functions of activities. In SharifSAN, an interpreter is implemented for translating these definitions in order to generate the executable model. This is one of the differences between SharifSAN and other modeling tools based on SANs. The compilers of high-level languages is used by other tools for SANs for compiling and building the executable models from SAN definitions. In SharifSAN, however, we use an interpretation method. The interpretation method is slower than the compilation technique. However, it enables us to implement some features like interactive model checker or token-game animator, which are useful for constructing and understanding SAN models.

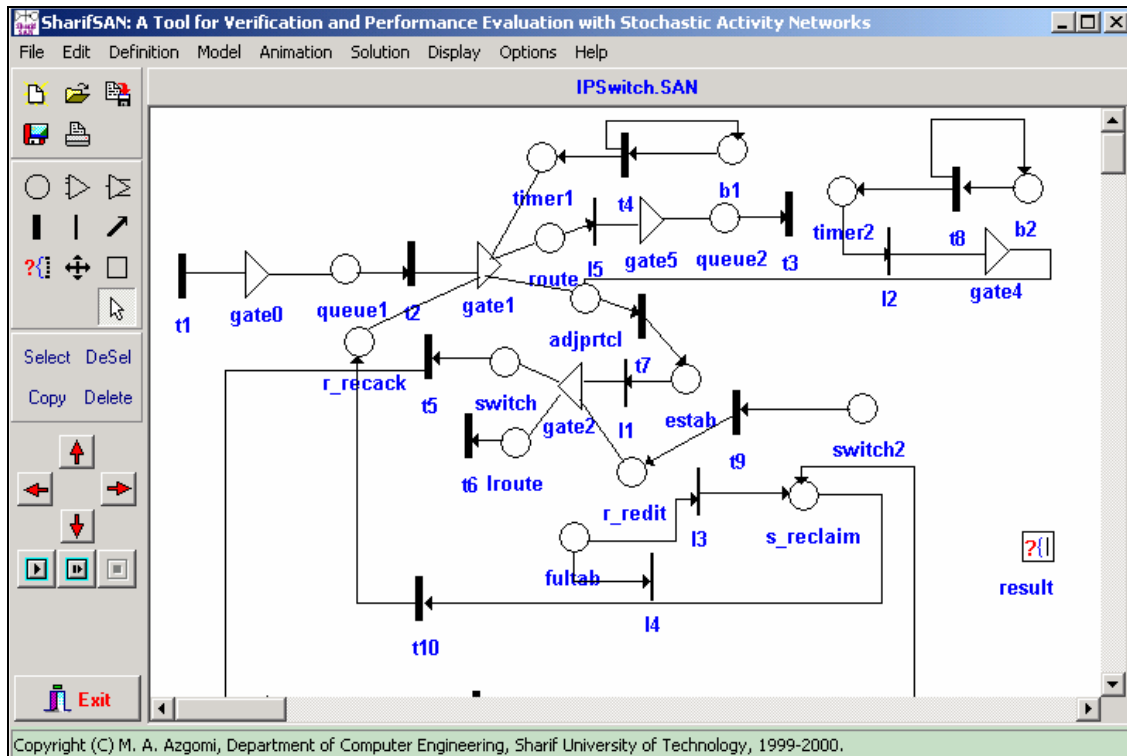


Fig. 1. A view of the user interface of SharifSAN

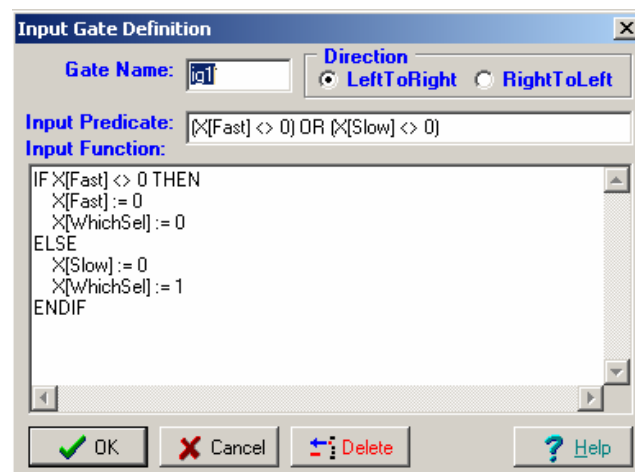


Fig. 2. Definition of an input gate and its input predicate and function

### c) Full state space generation

For verification, SharifSAN generates the full state space of the activity network model as a labeled reachability graph (LRG). A full reachability graph includes both stable and unstable states. In an unstable state, one or more instantaneous activities are enabled.

The state space generation algorithm of SharifSAN is shown in Fig. 3. This algorithm is divided into two parts. The first part is the main state generation algorithm (Fig. 3a) which determines all states of a model in a breadth-first manner. The second part specifies precisely how to determine the next states when timed and/or instantaneous activities are enabled in a specific state. The set of generated states ( $S$ ) contains all unique states that have been produced at any point in time by the state-space generator. Generated states are those states that have been found using the GenerateNS algorithm. The unexplored set contains all states that have not been used by GenerateNS. The set of next states found by GenerateNS is stored in the NS set. Algorithm 2 (GenerateNS) in Fig. 3b is used to determine all next states of a certain state. GenerateNS is a recursive algorithm that takes as input a state and a probability. Initially, the probability value is set to 1.0.

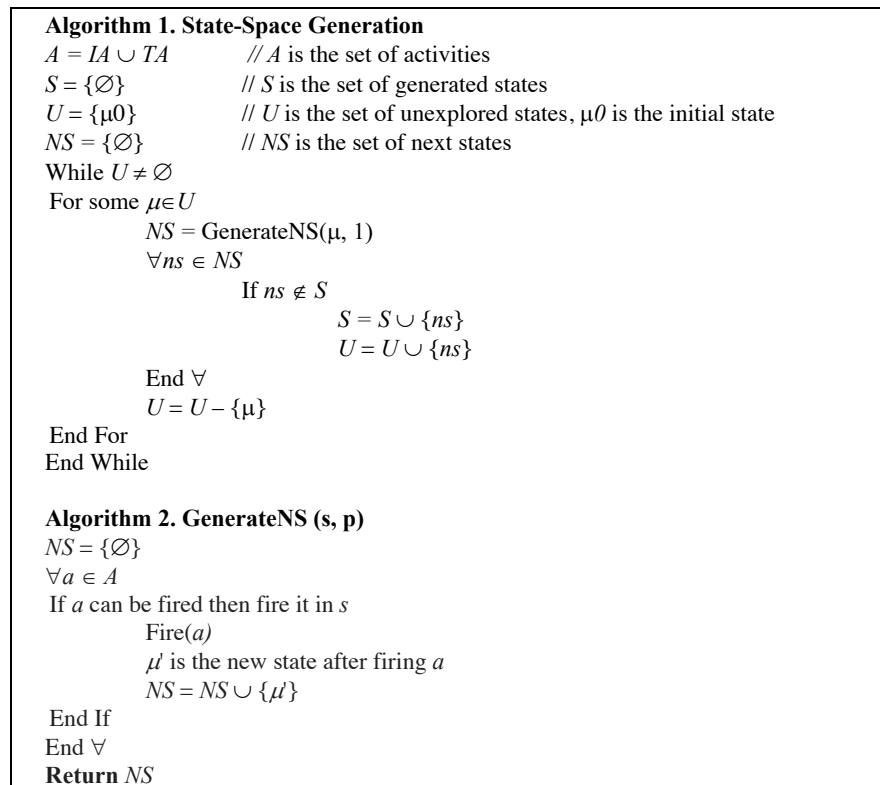


Fig. 3. General state space generation algorithm

#### d) Specifying the system properties as BTL formulas

Temporal logic [13] is one of the most interesting and powerful methods for the specification of the properties of concurrent and reactive systems. In SharifSAN, we use branching temporal logic (BTL) [9] for this purpose. So, the modeler can specify a few BTL formulas after building the activity network model of the system. SharifSAN then makes the expression tree for the input BTL formulas and finally checks for the correctness of the formulas.

A temporal formula is constructed from state formulas to which we apply temporal operators, Boolean operators (NOT, AND, OR), and quantification ( $\exists$ ,  $\forall$ ). The temporal operators are partitioned into two groups: the future operators and the past operators [13]. The future operators include Next, Henceforth, Eventually, Until, and Unless (Waiting-for). The past operators include Previous, Has-always-been, Once, Since, and Back-to.

In defining a system of temporal logic, there are two possible views regarding the underlying nature of time [9]. One is that the course of time is linear: at a moment there is only one possible future moment. The other is that time has a branching, tree-like nature: at each moment, time may split into alternative courses representing different possible futures. Depending upon which view is chosen, we classify a system of temporal logic as either a linear-time temporal logic (LTL) or a system of branching-time temporal logic

(BTL). In the logic of branching time, the modalities reflect the branching nature of time by allowing quantification over possible futures.

In SharifSAN, we use BTL with the future operators, including always (G), Eventually (F), and Until (U) for the specification of concurrent systems. These operators, in combination with quantification ( $\exists$ ,  $\forall$ ), yield AG, EG, AF, EF, AU, and EU operators, which together with Boolean operators (NOT, AND, OR) could be used in BTL formulas.

#### ***e) Transformation of an activity network to a transition system***

An LRG can be viewed as a transition system. Labels on the edges of an LRG specify the next state of the model after firing an activity. The initial state of LRG is the initial number of tokens in places of the activity network model, which is specified by the user at the time of editing the model. Each state is either stable or unstable and contains enough information to specify the possible next states. In a stable state, there is no enabled instantaneous activity. Otherwise, the state is unstable. For performance evaluation, where steady state solutions are mainly of interest, state space should be translated to stable state space. This is not, however, necessary for verification where BTL formulas are checked for both stable and unstable states. If we consider the unstable states, the LRG will be larger and the model checking will be more time-consuming.

#### ***f) Translation of BTL formulas to expression trees***

For checking the correctness of BTL formulas, they should be translated to suitable formats. Similar to state space generation, we use interpretation instead of compilation for translating BTL formulas to expression trees. Accordingly, we have developed a recursive-descent parser for translating formulas internally, which checks the syntax and semantics of formulas and generates expression trees for them. Expression trees are binary trees with internal nodes and leaves, which are the operators and operands of BTL formulas, respectively.

#### ***g) Model checking***

Model checking algorithms used in SharifSAN are similar to those considered in [10]. For verifying the correctness of a BTL formula, first all sub-formulas with a length of 1 are checked, then all sub-formulas with a length of 2, and so on. In an LRG, all correct sub-formulas in a given state are associated to that state as labels. If the algorithm is at the end of step  $i$ , all sub-formulas  $f_i$  with length  $i$  have been checked. A formula  $f$  is correct, if for all states  $s$  and for all  $i$ ,  $f_i$  is correct, namely,  $f_i \in \text{label}(s)$ . It is clear that this algorithm is of order  $O(L*(n+e))$ , where  $L$  is the average length of formulas,  $n$  is the number of nodes and  $e$  is the number of edges. Since  $L$  is limited and small, it can be considered as a constant factor. Therefore, the total order of the model-checking algorithm is  $O(n+e)$ .

### **4. PERFORMANCE EVALUATION WITH STOCHASTIC ACTIVITY NETWORKS**

The stochastic setting of SANs is used for performance evaluation. Assigning certain parameters to timed activities and viewing the model in a stochastic setting accomplish this. General probability distributions are assigned to timed activities, which may result in Markovian or non-Markovian models. In general, discrete-event simulation can be used to analyze a SAN model. However, if all timed activities are exponentially distributed and the model has a finite state space, the model will be Markovian, and may therefore be solved analytically using standard Markovian solution techniques.

In addition to steps (a) through (c) which are explained in the previous section for the verification system of SharifSAN, the following functionalities are also implemented for the evaluation purposes in the tool

- a) Tangible state space generation, b) Steady state analytic solution, c) Steady state simulation.

The above steps are described in the following subsections.

### a) Tangible state space generation

For the steady-state analytic solution of the SAN model that is used for the performance evaluation, only tangible states of the state space are required to obtain the transition-rate matrix. A tangible state is a state that has no instantaneous activity enabled in it and the net is stable. Therefore, by making some changes in GenerateNS algorithm in Fig. 3b, it is possible to delete the unstable states of the state space and obtain the tangible state space. The resulting algorithm (Algorithm 3) is shown in Fig. 4.

**Algorithm 3. GenerateNS (s, p)**  
 $NS = \{\emptyset\}$   
 If  $s$  is not stable  
 $\forall ia \in IA$   
 If  $ia$  can be fired then fire it in  $s$   
 $\mu'$  is the new state after firing  $ia$   
 $\mu'.p = ia.p$   
 $NS = NS \cup \{\mu'\}$   
 End If  
 End  $\forall$   
 $\forall \mu \in NS$        $\mu.p = \frac{\mu.p}{\sum_{\mu \in NS} \mu.p} \times s.p$   
 End If  
 If  $s$  is stable  
 $\forall ta \in TA$   
 If  $ta$  can be fired then fire it in  $s$   
 Fire  $ta$   
 $\mu'$  is the new state after firing  $ta$   
 $\mu'.p = ta.rate \times s.p$   
 $NS = NS \cup \{\mu'\}$   
 End If  
 End  $\forall$   
 End If  
 $NS' = \{\emptyset\}$   
 $\forall \mu \in NS$   
 If  $\mu$  is stable       $NS' = NS' \cup \{\mu\}$   
 Else       $NS'' = GenerateNS(\mu, \mu.p)$   
 $NS' = NS' \cup NS''$   
 End If  
 End  $\forall$   
**Return**  $NS'$

Fig. 4. Next tangible states generation algorithm

### b) Steady state analytic solution

If all timed activities of a SAN model are exponentially distributed, it can be represented by a continuous-time Markov chain (CTMC). The resulting CTMC will be ergodic if its reachability graph (RG) is finite and fully connected. An ergodic CTMC can then be solved analytically using standard Markovian solution techniques. Accordingly, the global balance equations of a CTMC may be solved to compute the steady state probabilities of its states. Since the state-transition-rate matrices of such CTMCs are usually sparse, numerical methods such as Gauss-Seidel and Gauss-elimination [14] could be helpful. Most performance measures of interest, as classified in [15], such as responsiveness (including waiting time, processing time, queue length, etc.) and usage-level (including throughput and utilization factor) may be evaluated from such steady state probabilities.

### c) Steady state simulation

If a SAN model includes non-exponential timed activities or its RG is infinite, it may not be solved analytically. In such cases, simulation may be employed to solve the model. Using SharifSAN, SAN models can be simulated interactively or automatically. In an interactive simulation or model animation, the modeler can trace the execution of the SAN model. It is possible to see the effects of the single-step execution on the graphical representation of the model. This is similar to single-step debugging of programs, which provides

a way for checking whether the model works as expected or not. This feature is useful on the process of constructing a new model and learning an existing model. Automatic simulation is similar to program execution. The purpose on automatic simulation is to be able to execute the SAN model as fast and efficiently as possible, without detailed human interaction and inspection.

Discrete-event simulation is used in most modeling tools based on Petri nets and their extensions for steady state simulation. There are a few methods for organizing the simulation run such as independent replications, single-run and regenerative methods [16]. The first two methods can be used for simulating SANs. The discrete-event simulation algorithm of SharifSAN (Algorithm 4) is shown in Fig. 5.

SharifSAN automatically computes the maximum and average number of tokens for all places, number of firings, throughput, total enabling time and enabling probability for all timed activities and the number of firings and throughput for all instantaneous activities for all models. In addition, it allows the user to define his/her interested markings and queries that will be computed at the end of simulation or even while the simulation is in progress.

**Algorithm 4. Discrete-Event Simulator**

1. Determine the set of enabled activities in the current marking of the model.
2. Reactivate those disabled activities whose reactivation predicates are true in the current marking.
3. Generate the activity execution time for newly enabled or reactivated activities.
4. Apply the corresponding enabling rates on the remaining time of the enabled timed activities.
5. If more than one instantaneous activity is enabled in an unstable marking, select one of them probabilistically. The probability of selection of each enabled instantaneous activity  $a$  in marking  $\mu$  is computed by the following formula introduced in [8]:
 
$$\alpha = \frac{C(\mu, a)}{\sum_{a' \in A'} C(\mu, a')}$$
- Where  $A'$  is the set of all enabled instantaneous activities and  $C$  is the *case probability* of  $a$  in marking  $\mu$ .
6. In a stable marking (no instantaneous activity is enabled), considering the remaining time of all enabled timed activities and their respective enabling rates, select the next timed activity for completion. In the case of two equal completion times, select one of the corresponding timed activities, probabilistically.
7. Fire the selected instantaneous or timed activity:
  - Remove a token from all input places,
  - Execute the function of all input gates,
  - Add a token to all output places, and
  - Execute the function of all output gates.
8. Disable those enabled activities whose enabling predicate are not true in the current marking.
9. Set the simulation clock to the time of the most eminent event.
10. Collect the aggregated statistics and update user-defined queries.
11. If the specified confidence level (or fixed replications/time interval) has not been achieved, go to step 1.

Fig. 5. Discrete-event simulator of SAN models

## 5. SOFTWARE ARCHITECTURE OF SHARIFSAN

The overall software architecture of SharifSAN, including its main modules and their relations, has been depicted in Fig. 6.



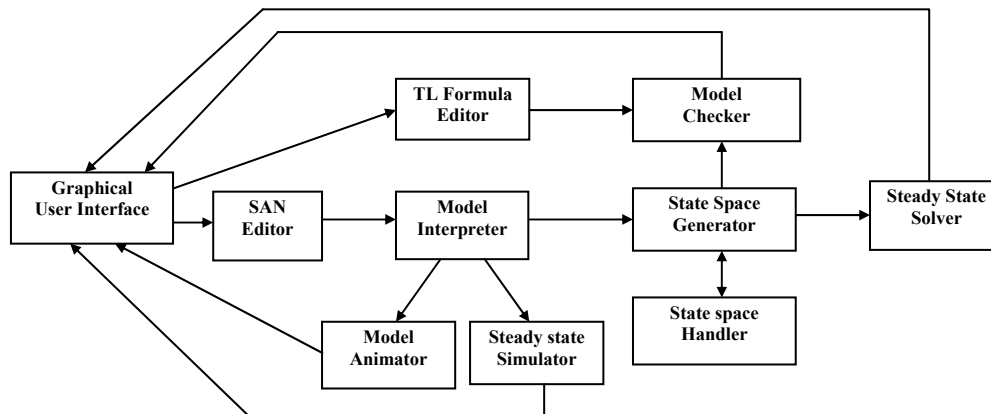


Fig. 6. Software architecture of SharifSAN

Currently, SharifSAN includes the following modules:

- **Graphical user interface:** SharifSAN provides a graphical user interface and an integrated development environment (IDE) for edition, checking the syntax and semantics, interpretation, animation, state space generation, verification, simulation and analytic solution of SAN models. This integration makes SharifSAN more user-friendly than most other tools. SharifSAN runs under Windows and benefits its GUI facilities.
- **SAN editor:** For editing a SAN model, a user can select primitives of SANs such as place, input gate, output gate, timed activity and instantaneous activity and put them as separate objects on the design page and connect them by arcs to compose the whole model. The parameters of each object such as predicate, function and other values can be entered directly by clicking on the object.
- **Model interpreter:** This module is for translating the definitions of SANs and to execute them by interpretation. Because the input/output functions of the input/output gates of SANs are general computable functions, we have developed an interpreter for a subset of a high-level language similar to Pascal that contains the most usual programming constructs. This interpreter can communicate with user-defined functions, which are implemented directly in the definition of the model or in Dynamic Link Library (DLL) files.
- **Model animator:** SharifSAN can animate SAN models. This capability is known as token-game animation, and is useful for constructing and understanding SAN models. For animation, it shows step-by-step or continuous execution of the network by firing enabled activities and changing the states. Animation can be in a nondeterministic or stochastic manner. In stochastic animation, the result of the discrete-event simulation of SANs will be animated. Debugging is also a useful feature of this animator, enabling users to define queries about the model and trace its results while animation is in progress. This feature helps the modeler verify the model behavior.
- **State space generator:** It executes SAN models for generating state space until no new state is added and then builds RG by calling the state space handler module.
- **State space handler:** This module manages RG. Currently, this module uses the main memory and can handle graphs with at most one million nodes.
- **TL formula editor:** This editor enables the user to enter a few BTL formulas and check their syntax and semantic. If there is no error in the formula, the editor will generate expression tree and pass it to the model checker module.
- **Model checker:** Its main function is verification by applying the model checking algorithms to a given LRG and checking for the correctness of a given BTL formula. The result of this check, which is TRUE or FALSE, will be displayed on the user interface.
- **Steady state solver:** This module will be used if all timed activities of the models are exponentially distributed. If so, the corresponding Markov chain (MC) will be generated and if finite, its ergodicity will also be checked. If it is ergodic, its global balance equations will be solved to compute its steady state probabilities. The results will be displayed on the user interface. The user can then use the statistical analyzer module to compute the queries and desired performance measures.

- **Discrete-event simulator:** If the steady state solver cannot be used, simulation can be utilized for performance evaluation. This module simulates SANs using discrete-event simulation and collects the statistics specified by the user and displays them when simulation is in progress or after it is done. The user can use the statistical analyzer module to compute his/her queries and the desired performance measures. Model debugging is also possible. This feature enables a modeler to define queries about the model and trace their results while simulation is in progress.

## 6. APPLICATIONS OF SHARIFSAN

SharifSAN has been designed to be useful for both functional and operational analyses of computer and communication systems. In this section, we present some applications of this tool on verification of concurrent systems and performance evaluation of computer and communication systems.

### a) Verification of concurrent systems

The integration of BTL with SANs provides capabilities for modeling and the specification of concurrent and reactive systems, as well as verifying their properties. We have used this feature of SharifSAN for the verification of some sample concurrent systems. However, model checking with this tool could only be done for models with finite state spaces and not for models with very large or infinite state spaces.

The following is an example of verification with SharifSAN:

**Example 1.** Consider a concurrent system with two producers and two consumers. There are two input buffers and a common channel for the transmission of messages from producers to consumers. This channel is a critical resource, which should be accessed mutually exclusive by producers. The capacity of input buffers is limited to 10 places. Assume the first producer has non-preemptive priority over the second one on accessing to the common channel. We want to model this system and check for the following properties:

1. Mutual exclusion on access to the common channel,
2. The existence of starvation for producers.

The activity network model for the above system has been constructed using SharifSAN and is shown in Fig. 7. In this model, *Prod1* and *Prod2* are timed activities which model producers. *Buf1* and *Buf2* places model the buffers with capacity 10 for two producers, and *CS* place models in the common channel. The gate table for the model is also shown in the figure.

In SharifSAN, we use a syntax that is similar to the Pascal programming language for the predicates and functions of the input and output gates. We also use "X[PlaceName]" instead of "MARK(PlaceName)" to indicate the marking of a place identified as "PlaceName".

Now, we can verify some properties of the system by entering and checking some BTL formulas:

**Property 1.** Access to the channels should be mutually exclusive.

The mutual exclusion could be specified with a BTL formula like (1)

$$\text{NOT (EF ((X[InChan1]=1) AND (X[InChan2]=1)))} \quad (1)$$

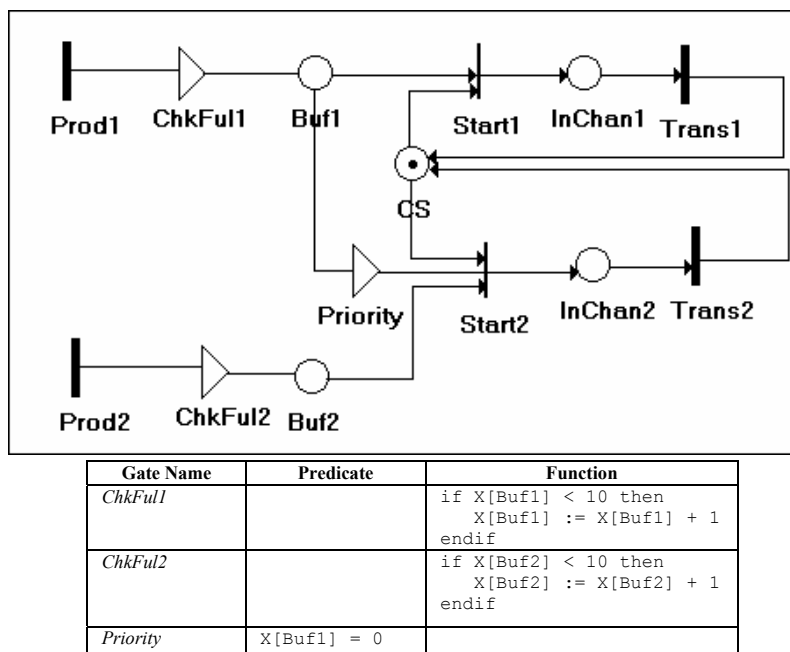


Fig. 7. A SAN model for a system with two producers and two consumers

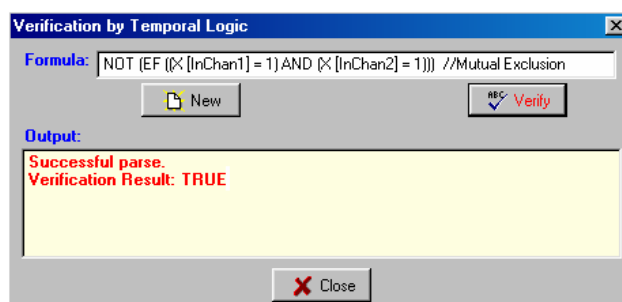


Fig. 8. Entering BTL formula and viewing verification result

The meaning of formula (1) is as follows: There is not any state in which two producers are transmitting a message through a common channel simultaneously. In this formula, EF is a BTL operator and means "∃ eventually" and "X [InChan1] = 1" means there is a token in place InChan1.

After entering the above formula, SharifSAN will generate the state space for the model and convert the BTL formula to an expression tree. Then the correctness of the formula on the state space will be automatically checked and the result, which is TRUE, will be displayed on the user interface of SharifSAN (see Fig. 8). This answer proves that access to the common channel is always mutually exclusive.

**Property 2.** Starvation should be avoided in the system.

The starvation is not a desired property in a concurrent system. For this example, starvation is a situation in which one producer will no longer be able to send a message through the common channel. Starvation for producer 2 could be described as BTL formula (2):

$$\text{NOT (AG (AF (X[InChan2]=1)))} \tag{2}$$

The meaning of formula (2) is as follows: There is a state in which producer 2 will no longer be able to send any message. In this formula, AG and AF are BTL operators and stands for "∀ henceforth" and "∃ eventually", respectively. After entering this formula, the result, which is TRUE, will be displayed on the user interface of SharifSAN. This answer proves that there is a starvation situation in the system. Based on this result, the modeler can change the model and construct a starvation-free model. Using the model animation feature of SharifSAN, the user can observe that the above result is correct. This is because after a

few steps, there will be some tokens in Buf1 and because of the priority of Prod1 over Prod2, Prod2 will no longer be able to access the common channel and send any message.

### b) Performance evaluation of computer systems

SharifSAN is also useful for performance evaluation purposes using analytic or simulative methods. While the analytic method has some limitations, the simulation of SAN models could be used for a wide range of systems. The following is an example of performance evaluation with SharifSAN:

**Example 2.** Consider a computer system with two processors, in which one of them is faster than the other. Tasks enter the system based on a Poisson process and route to a processor for execution. The fast processor has priority over the slow one on processor allocation. If both processors are busy, tasks wait in a bounded buffer with a capacity equal to 3. If this buffer is full, tasks will be rejected. Service time of different tasks is exponentially distributed.

A SAN model for this system has been constructed with SharifSAN and is presented in Fig. 8. In this model, *Entrance* is a timed activity, which is exponentially distributed with a rate of 7.0. This activity models the process of entering tasks to the system. *SlowProc* and *FastProc* timed activities, which are exponentially distributed with rates 5.0 and 10.0, respectively, model two processors of the system. Because, all timed activities of this system are exponentially distributed, this model is Markovian and could be solved analytically.

The results of its analytic solution and simulation are shown in Table 2. Simulation has been run for 10000 and 100000 units of time for before and after steady state, respectively. As it is shown in the table, results for the analytic solution and simulation are close to each other. We have defined some variables to compute some of the statistics about the behavior of system. The *arrivals* variable represents the number of tasks arrived to the system and *Rejected* variable represents the number of rejected tasks from system. We have defined another variable *RejectProb*, which is equal to  $Rejected/Arrivals$  and represents the probability of task rejection in the system. As it is shown in the table, its computed value is 0.01566, which is near the value of  $Pr \{X [Jobs] = 3\} = 0.01589$  or 0.01590, resulting from the analytic solution and simulation, respectively.

Table 2. Results of analytic solution and simulation

| Data  | Analytic Results | Simulation Results |
|---|------------------|--------------------|
| Average no. of tokens in place <i>Jobs</i>      | 0.18876          | 0.18891            |
| Average no. of tokens in place <i>SlowAlloc</i> | 0.40961          | 0.41018            |
| Average no. of tokens in place <i>FastAlloc</i> | 0.48407          | 0.48512            |
| Average no. of tokens in place <i>WhichSel</i>  | 0.31169          | 0.31161            |
| Average no. of tokens in place <i>Fast</i>      | 0.51593          | 0.51633            |
| Average no. of tokens in place <i>Slow</i>      | 0.59039          | 0.58981            |
| Probability of 0 tokens inside <i>Jobs</i>      | 0.87707          | 0.87679            |
| Probability of 1 tokens inside <i>Jobs</i>      | 0.07297          | 0.07333            |
| Probability of 2 tokens inside <i>Jobs</i>      | 0.03405          | 0.03396            |
| Probability of 3 tokens inside <i>Jobs</i>      | 0.01589          | 0.01590            |
| Probability of 0 tokens inside <i>SlowAlloc</i> | 0.59039          | 0.58981            |
| Probability of 1 tokens inside <i>SlowAlloc</i> | 0.40960          | 0.41018            |
| Probability of 0 tokens inside <i>FastAlloc</i> | 0.51592          | 0.51487            |
| Probability of 1 tokens inside <i>FastAlloc</i> | 0.48407          | 0.48512            |
| Probability of 0 tokens inside <i>WhichSel</i>  | 0.68830          | 0.68838            |
| Probability of 1 tokens inside <i>WhichSel</i>  | 0.31169          | 0.31161            |
| Probability of 0 tokens inside <i>Fast</i>      | 0.48407          | 0.48512            |
| Probability of 1 tokens inside <i>Fast</i>      | 0.51592          | 0.51487            |
| Probability of 0 tokens inside <i>Slow</i>      | 0.40960          | 0.41018            |
| Probability of 2 tokens inside <i>Slow</i>      | 0.59039          | 0.58981            |
| Total number of jobs entered to the system      | -                | 701221             |
| Total number of rejected jobs                   | -                | 10988              |
| Rejection probability of incoming jobs          | -                | 0.01566            |

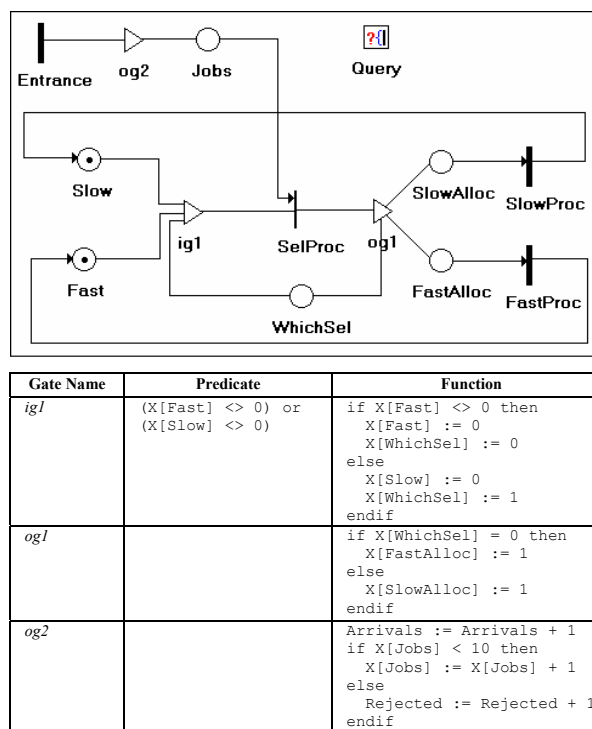


Fig. 9. An example of a computer system with two processors

### c) Performance evaluation of communication systems

As a design goal of SharifSAN, we intend to make this tool more useful in the area of communication systems and high-speed networking. The integration of verification and performance evaluation makes SharifSAN very useful in such areas. In these applications, the verification of protocol correctness is as important as the evaluation of measures of the quality of service (QoS) such as delay or throughput.

So far, SharifSAN has been employed in two research projects in this area. In [17], a model for the structure of the IP switch [18] has been constructed and evaluated using the simulative method of SharifSAN. Also, in [19], a model for the structure of the tag switch [20] has been constructed and analyzed. The purposes of these researches were to evaluate performance measures such as delay, cell-loss probability, and the optimal size of input and output queues. Since the resulting SAN models in both researches were non-Markovian, they have been simulated.

The application of SharifSAN in these researches demonstrated the usefulness of this tool for modeling and analysis of communication systems and high-speed networks.

## 7. CONCLUSIONS

In this paper, we introduced an integrated tool for modeling and analysis of computer and communication systems. This modeling tool, which is called SharifSAN, is based on a new definition of stochastic activity networks.

SharifSAN has features for both verification and performance evaluation. For verification, first the nondeterministic setting of SANs, called activity networks, is used to model the system. Then, the branching temporal logic is used for specifying of system properties. For performance evaluation, SharifSAN provides a steady state solver for Markovian models. Also, a discrete-event simulator is provided, which can be used for both Markovian and non-Markovian systems. Model animation (or token-game animation) is another feature of SharifSAN. This feature is useful for constructing, debugging and understanding SAN models.

We are currently working to add techniques for transient analysis to SharifSAN, in order to make it useful for dependability and performability evaluation purposes. We will also improve the state space handler of the tool for handling large state spaces.

## REFERENCES

1. Movaghar, A. & Meyer, J. F. (1984). Performability modeling with stochastic activity networks. *Proc. of the 1984 Real-Time Systems Symp., Austin, TX*, 215-224.
2. Peterson, J. L. (1981). *Petri nets theory and the modeling of systems*. Prentice-Hall.
3. Molloy, M. K. (1982). Performance analysis using stochastic petri nets, *IEEE Trans. on Computers*, C-31, 913-917.
4. Ajmone Marsan, M., Balbo, G. & Conte, G. (1986). *Performance models of multiprocessor systems*. MIT Press.
5. Sanders, W. H. & Meyer, J. F. (1986). METASAN: a Performability evaluation tool based on stochastic activity networks. *Proc. ACM/IEEE-CS Fall Joint Computer Conference*, 807-816.
6. Sanders, W. H., Obal II, W. D., Qureshi, M. A. & Widjanarko, F. K. (1995). The UltraSAN modeling environment. *Performance Evaluation*, 24, 1-33.
7. Deavours, D. D., et al. (2002). The Möbius framework and its implementation. *IEEE Trans. on Soft. Eng.* 28(10), 956-969.
8. Movaghar, A. (2001). Stochastic activity networks: A new definition and some properties. *Scientia Iranica*, 8(4), 303-311.
9. Emerson, E. (1990). *Temporal and modal logic*. J. van Leeuwen (ed.), Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics, Elsevier Science Publishers B.V.
10. Clarke, E., Emerson, E. & Sistla, A. (1986). Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Prog. Lang. and Systems*, 8(2), 244-263.
11. Deavours, D. D. & Sanders, W. H. (1999). An efficient well-specified check. *Proc. of 9th Int. Workshop on Petri Nets and Performance Models (PNPM'99)*, Zaragoza, Spain.
12. Sanders, W. H. & Meyer, J. F. (2001). Stochastic activity networks: Formal definitions and concepts. In Brinksma, E., Hermanns, H., and Katoen, J. P. (eds.), LNCS 2090, Springer-Verlag, 315-343.
13. Manna, Z. & Pnueli, A., *The temporal logic of reactive and concurrent systems*. Specifications, Springer-Verlag, Berlin.
14. Rice, J. R. (1993). *Numerical methods*. Software and Analysis, Academic Press.
15. Kant, K. (1992). *Introduction to computer system performance evaluation*, McGraw-Hill.
16. Banks, J., Carson II, J. S. & Nelson, B. L. (1996). *Discrete-event system simulation*. 2nd Edition, Prentice-Hall.
17. Kohandani, F. & Valaee, S. (2001). IP switch structural design and relevant analysis based on SAN model. *Proc. of the 9th Iranian Conference on Electrical Engineering (ICEE'01)*, Tehran, Iran, 7.1-7.10 (In Persian).
18. Newman, P., Minshall, G., Lyon, T. & Huston, L. (1997). IP switching and gigabit routers, *IEEE Communications Magazine*, 35, 64-69.
19. Azimdoost, B. (2000). Structural design of interfaces for transmitting IP over ATM network using tag switching approach. *MS. Thesis*, Department of Electrical Engineering, Sharif University of Technology (In Persian).
20. CISCO Systems Inc. Tag switching-uniting routing and switching for scalable, High-Performance Services, URL: <http://www.cisco.com>.

## یک ابزار مدل‌سازی برای تعریف جدیدی از شبکه‌های فعالیت تصادفی

محمد عبداللهی ازگمی و علی موقر

دانشکده مهندسی کامپیوتر، دانشگاه صنعتی شریف، تهران، جمهوری اسلامی ایران

**چکیده:** شبکه‌های فعالیت تصادفی (SANs: stochastic activity networks) یکی از بسط‌های قوی و منعطف شبکه‌های پتری (Petri nets) است. چندین ابزار مدل‌سازی برای مدل‌های SANs وجود دارد که همگی آنها مبتنی بر تعریف اولیه این مدل بوده و بیشتر برای مقاصد ارزیابی (evaluation) مورد استفاده قرار گرفته‌اند. در این پژوهش یک ابزار مدل‌سازی به نام SharifSAN طراحی و پیاده‌سازی شده که مبتنی بر تعریف جدیدی از مدل‌های SANs بوده و هم برای مقاصد درستی‌یابی (verification) و هم ارزیابی کارایی (performance evaluation) قابل استفاده است. با استفاده از این ابزار، جنبه‌های وظیفه‌مندی با روش بررسی مدل (model checking) و منطق زمانی انشعابی (BTL: branching temporal logic) انجام می‌شود. در حالیکه، جنبه‌های عملیاتی با استفاده از روش‌های تحلیلی و شبیه‌سازی ارزیابی می‌شوند. تنظیم غیرقطعی مدل‌های SANs برای کاربرد اول و تنظیم تصادفی آنها، برای کاربرد دوم مورد استفاده قرار می‌گیرد. در این مقاله، پس از ارائه مقدمه‌ای بر تعریف جدید شبکه‌های فعالیت تصادفی، مشخصات، طرح و کاربردهای نرم‌افزار SharifSAN معرفی می‌شود.