SPNP: Stochastic Petri Net Package

Gianfranco Ciardo[†], Jogesh Muppala, Kishor Trivedi
Department of Computer Science, Duke University, Durham, NC 27706, USA
[†] Now at Software Productivity Consortium

Abstract

We present SPNP, a powerful GSPN package developed at Duke University. SPNP allows the modeling of complex system behaviors. Advanced constructs are available, such as marking-dependent arc multiplicities, enabling functions, arrays of places or transitions, and subnets; in addition, the full expressive power of the C programming language is available to increase the flexibility of the net description.

Sophisticated steady-state and transient solvers are available; cumulative and up-to-absorption measures can be computed. In addition, the user is not limited to a predefined set of measures: detailed expressions reflecting exactly the measures sought can be easily specified.

We conclude comparing SPNP with two other SPN-based packages, GreatSPN and METASAN.

1 Introduction

Reliability block diagrams and fault trees are commonly used for system reliability/availability analysis [41]. These model types allow a concise description of the system under study and can be evaluated efficiently, but they cannot represent dependencies occurring in real systems [39, 45]. Markov models, on the other hand, are capable of capturing various kinds of dependencies that occur in reliability/availability models [13, 14, 44].

Task precedence graphs [21, 22, 26, 37, 38] can be used for the performance analysis of concurrent programs with unlimited system resources. System performance analysis using product-form queueing networks [23, 25] can instead consider contention for resources. The product-form assumptions are not satisfied, however, when behaviors such as concurrency within a job, synchronization, and server failures are considered. Once again, Markov models do provide a framework to address all these concerns.

The common solution for modeling reliability/availability or performance would then appear to be the use of Markov models, but one major drawback of Markov models is the

largeness of their state space. Generalized Stochastic Petri nets (GSPNs) can be used to generate a (large) underlying Markov process automatically starting from a concise description. Nevertheless, the combinatorial growth of their state space (reachability graph) constitutes a major limitation to applicability of GSPN(-reward) models in real-life problems.

If we can identify states with low probability then we can avoid the generation of such states. Such *state truncation* methods are an integral part of many reliability [4] and availability analysis tools [14]. The error resulting from state truncation has also been analyzed [24, 31].

An alternative way of avoiding the generation and solution of large one-level model is to decompose the GSPN into nearly independent subnets. The time for the generation, storage space and the solution time should reduce drastically in this way. However, since the subnets are dependent, their solutions need to iteratively communicate. Such fixed-point iterative schemes to solve large GSPNs are discussed in [8].

Traditionally, performance analysis assumes a fault-free system. Separately, reliability and availability analysis is carried out to study system behavior in the presence of component faults, disregarding different performance levels in different configurations. Several different types of interactions and corresponding tradeoffs have prompted researchers to consider combined evaluation of performance and reliability/availability [18, 27, 28, 42]. Most work on the combined evaluation is based on the extension of Markov processes to Markov reward processes [17], where a reward is attached to each state of the Markov process.

Markov reward processes have the potential to reflect concurrency, contention, fault-tolerance, and degradable performance; they can be used to obtain not only program/system performance and system reliability/availability measures, but also combined measures of performance and reliability/availability [3, 9, 27, 29, 42]. Since the Markov process is generated from a concise GSPN model, it is necessary to express the reward structure in terms of GSPN entities. In other words, the GSPN becomes a "GSPN reward process" which can be automatically transformed into a Markov reward process.

Steady-state analysis of GSPNs is often adequate to study the performance of a system, but time-dependent behavior is sometimes of greater interest: instantaneous availability, interval availability, and reliability (for a fault-tolerant system); response time distribution of a program (for performance evaluation of software); computational availability (for a degradable system). Except for a few instances [6, 10, 11, 40], transient analysis of (G)SPN models has not received much attention in the past.

Analytical models are often used to evaluate alternatives during the design phase of a system, when exact values for all the input parameters may not be known yet. It is then useful to estimate how the output measures are affected by variations in the value of these parameters. Other important applications of *sensitivity analysis* are system optimization and bottleneck analysis.

A GSPN model lends itself to sensitivity analysis at various levels: (1) changes in the

GSPN structure; (2) arbitrary changes in the initial marking; (3) changes in the initial number N of tokens in a place; (4) changes in a parameter μ involved in the definition of the rate or probability of one or more transitions. An "X-Y plot" is especially meaningful in (3) and (4). Unfortunately, (1), (2), and (3) require reevaluation of the entire model, since they modify the underlying reachability graphs (RG). In (4), the structure of the underlying RG is unaffected, only the rates and probabilities dependent on μ change; this can be reflected in the generator matrix Q of the underlying CTMC. The sensitivity measures are then computed by analyzing the underlying CTMC.

The Stochastic Petri Net Package (SPNP) described in this paper, allows the specification of GSPN reward models, the computation of steady-state, transient, cumulative, time-averaged, and "up-to-absorption" measures and sensitivities of these measures. Efficient and numerically stable algorithms employing sparse matrix techniques are used to solve the underlying CTMC. Parametric sensitivity analysis of GSPN models is also implemented in SPNP. Rates or probabilities of transitions and their derivatives need to be specified (SPNP is currently unable to compute the symbolic derivatives automatically). SPNP can then compute steady-state or transient output measures and their derivatives with respect to the chosen parameter.

Precursors to SPNP are DEEP [11, 12] and GSPNA [1].

2 Important Features of SPNP

The input language for SPNP is CSPL (C-based SPN language). A CSPL file is a C file [20]; it is compiled using the C compiler and then linked with the precompiled files constituting SPNP.

The full power of the C programming language is available, but most applications will only require a limited knowledge of the C syntax, since predefined functions are available to define GSPN objects.

2.1 Marking dependency

Perhaps the most important characteristic of CSPL is the ability to allow extensive marking dependency. Parameters such as the rate of a timed transition, the cardinality of an input arc, or the reward in a marking, can be specified as a function of the number of tokens in some (possibly all) places. Marking dependency can lead to more compact models of complex systems. The syntax to express marking dependency is a natural extension of the syntax to describe marking independent behavior. For example,

```
trans("t1"); rateval("t1",2.7);
```

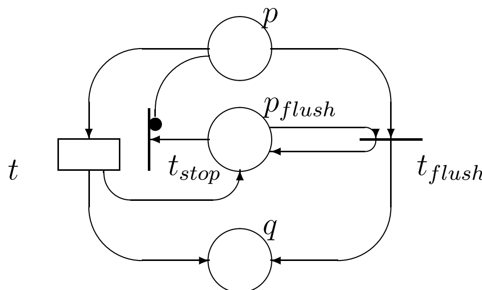

Figure 1: Flushing a place.

defines a transition `t1` with a constant firing rate of 2.7; if the rate of transition `t2` is the number of tokens in place `p1` times `MU1` (a constant) plus the number of tokens in place `p2` times `MU2` (another constant), we can define the marking dependent function

```
double   rate2() {
  return( mark("p1") * MU1 + mark("p2") * MU2 );
}
```

and then use it in the specification of the rate for transition `t2`:

```
trans("t2");  ratefun("t2",rate2);
```

2.1.1 Variable cardinality arc

In the standard PN and in most SPN definitions, the cardinality of an arc is a constant integer value [34]. If the cardinality of the input arc from place p to transition t is k, k tokens must be in p before t can be enabled and, when t fires, k tokens are removed from p. Often, *all* the tokens in p must be moved to some other place q [11]. A constant cardinality arc cannot accomplish this in a compact way. In GSPNs, we can use transition t to move only the first token from p to q and to deposit a control token in p_{flush}; then, immediate transition t_{flush} moves the remaining tokens one at a time until p is empty; finally, immediate transition t_{stop} removes the control token from p_{flush} (see figure 1).

The same behavior can be easily described by specifying the cardinalities of the input arc from p to t and of the output arc from t to q as $\#(p)$, the number of tokens in p. This representation is more natural, no additional transitions or places are required, and the execution time is likely to be shorter for two reasons: fewer transitions exists and no additional vanishing marking is generated.

The use of variable cardinality is somewhat similar to the conditional case construct of the Stochastic Activity Networks (SANs) [30]. We allow variable cardinality input, output, and inhibitor arcs.

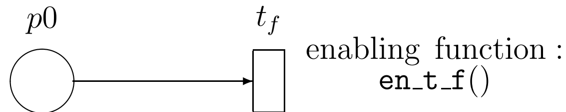

Figure 2: Truncation using Enabling Function.

When the cardinality of the arc is zero, the arc is considered absent. The user of SPNP must be aware of the difference between defining the cardinality of an input arc as "$\max\{1, \#(p)\}$" or as "$\#(p)$". The first definition disables t when p is empty, the second does not; the correct behavior depends on the particular application.

2.1.2 Enabling functions

Each transition t has an associated (boolean) enabling function e. The function is evaluated in marking M when "there is a possibility that t is enabled", that is, when (1) no transition with priority higher than t is enabled in M; (2) the number of tokens in each of its input places is larger than or equal to the (variable) cardinality of the corresponding input arc; (3) the number of tokens in each of its inhibitor places is less than the (variable) cardinality of the corresponding inhibitor arc. Only then $e(M)$ is evaluated; t is declared enabled in M iff $e(M) = \mathit{TRUE}$. The default for e is the constant function TRUE.

The ability to express complex enabling/disabling conditions textually is invaluable. Without it, the designer might have to add extraneous arcs or even places and transitions to the GSPN, to obtain the desired behavior. The logical conditions that can be expressed graphically using input and inhibitor arcs are limited by the following semantics: a logical "AND" for input arcs (all the input conditions must be satisfied), a logical "OR" for inhibitor arcs (any inhibitor condition is sufficient to disable the transition). A condition such as

$$(\#(p_1) \geq 3 \ \vee \ \#(p_2) \geq 2) \ \ \wedge \ \ (\#(p_3) = 5 \ \vee \ \#(p_4) \leq 1)$$

is difficult to represent graphically. An additional problem with the input arc is its double semantic. An input arc not only defines a precondition for the firing of transition t (at least n tokens must be present in p), but it also defines a postcondition (exactly m tokens are removed from p). In the standard definition, $n = m$ is the cardinality of the arc, but there are cases where the logic of the system being modeled requires $m < n$ ($m > n$ is meaningless). We can then set the cardinality of the input arc to n and add an output arc from t to p with cardinality $(n - m)$. Alternatively, though, we could choose to define an enabling function returning *FALSE* when $\#(p) < n$ and set to m the cardinality of the input arc from p to t.

Enabling functions can also be used to specify state truncation. For example, suppose t_f is a transition associated with the time to failure of some resource type as shown in figure 2. Further, let us assume that the probability of having more than *trunc* resources "down" at

any given time is very small. We could truncate the state space (reachability graph) of the GSPN by not considering states with more than *trunc* concurrent failures of the resource. This can be specified by associating the following enabling function with the transition t_f:

```
enabling_type en_t_f() {
  if ( mark("p0") > N - trunc )
    return(1);
  else
    return(0);
}
```

The transition t_f is enabled as long as the number of failures of resource is less than *trunc* ; i.e., the number of tokens in place $p0$ is more than $N - trunc$ where N is the initial number of tokens in the place (initial number of the resource available).

2.1.3 Assertions

A logical condition on the marking of the GSPN is useful for debugging purposes as well. A (single) boolean function a can be defined, expressing general assertion(s) that are believed to hold in any marking of the reachability set.

As soon as a marking M is generated, $a(M)$ is evaluated. M is considered "illegal" if $a(M) = FALSE$; the execution then halts, M is displayed, and the partially generated reachability graph is written to a file, as a debugging aid.

This feature is especially useful with large and complex GSPNs, where it greatly reduces the time to discover simple errors, such as a missing arc or an incorrect cardinality specification. This check is turned off by setting a identically equal to *TRUE*.

2.2 Run-time specification of the GSPN

An important feature of CSPL is the ability to exploit C language constructs to represent a large class of GSPNs within a single CSPL file. In contrast to other (G)SPN packages [6, 10] that allow the run-time specification of scalar parameters such as the initial number of tokens in a place or the value of a parameter used in the definition of a rate or probability, SPNP provides a function to input a value at run-time, before reading the specification of the GSPN. This input value can be used in SPNP to modify the value of a scalar parameter as well as the structure of the GSPN itself. For example, the statement

```
policy = input("policy (select 0 or 1)");
```

causes the message

```
Please type 'policy (select 0 or 1)' >
```

to be displayed at run-time. The value typed is then used to set the value of `policy` (a C variable), which can be used during the specification of the GSPN structure:

```
if (policy == 0) {
  trans("t_pol0"); rateval("t_pol0",3.0);
  iarc("t_pol0","pX"); oarc("t_pol0","pY");
}
```

If the run-time value provided for `policy` is 0, the GSPN contains a timed transition `t_pol0` with rate 3.0, having an input arc from place `pX` and an output arc to place `pY`; the transition and the arcs are absent otherwise.

Two features are especially useful to exploit this "structural parametrization": arrays of places or transitions and subnets. A single CSPL file is sufficient to describe any legal GSPN, since the user of SPNP can input at run-time the number of places and transitions, the arcs among them, and any other required parameter. In practice, the class of GSPNs described by a single CSPL file is more likely to represent only minor variations on a common structure, to increase compactness and consistency. A single file can be used to represent all the GSPNs corresponding to a given system under consideration, even when they differ somewhat in their structure. As a result, the numerical parameters used in the specification of rates and probabilities need to appear in only one file, decreasing the risk of having inconsistent definitions in different files.

2.2.1 Arrays of places and transitions

SPNP provides functions to group places or transitions into one- or two- dimensional arrays. For example,

```
place_1("cpu",7);
```

defines seven places, with names `cpu.0` through `cpu.6`. Assuming the value of `N` is three,

```
trans_2("move",N,2*N);
```

defines eighteen transitions, with names `move.0.0` through `move.2.5`. Since `N` can be set at run-time, a single CSPL file can be used, for any array size. The most complex aspect of managing arrays of places or transitions is the description of the arcs connecting them. We can consider a single place or transition as a zero-dimensional array, so an arc can connect zero-, one-, or two-dimensional objects:

```
iarc_2_0("move",ALL,0,"obj");
```

defines input arcs from place `obj` to transitions `move.0.0`, `move.1.0`, and `move.2.0`. Complex connections often require the individual specification of each arc. If the input arc is absent between `obj` and `move.2.0`, the following specification can be used:

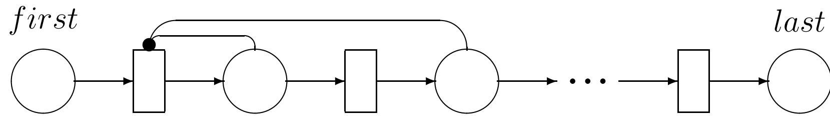

Figure 3: Erlang subnet.

```
for (i = 0; i < N; i++)
  if (i != 2)   iarc_2_0("move",i,0,"obj");
```

2.2.2 Subnets

A CSPL file is a legal C file, so it can contain function calls. If function `subn` contains the definition of a subnet, each call to `subn` generates a new instance of the corresponding subnet. The following function definition can be used to generate Erlang subnets with arbitrary number of stages and arbitrary timing (figure 3):

```
void   erlang(first,last,n,time,paux,taux)
char   *first,*last;/* 1st, last places in subnet */
int    n;                /* number of stages (> 1) */
double time;   /* E[time] to traverse subnet (> 0)*/
char   *paux,*taux;    /* used for internal names */
{
  int i;
  place_1(paux,n-1);       /* n-1 internal places */
  trans_1(taux,n);      /* n internal transitions */
  rateval_1(taux,ALL,n/time); /* int. trans. rate */
  iarc_1_0(taux,0,first)       /* first input arc */
  oarc_1_0(taux,n-1,last);     /* last output arc */
  for (i = 0; i < n-1 ; i++) {
    iarc_1_1(taux,i+1,paux,i);  /* n-1 input arcs */
    oarc_1_1(taux,i,paux,i);   /* n-1 output arcs */
    harc_1_1(taux,0,paux,i);/* n-1 inhibitor arcs */
  }
}
```

If the calls

```
erlang("p","p1",4,3.0,"a","b");
erlang("p","p2",5,2.0,"c","d");
```

are used in the definition of a GSPN, two instances of the Erlang subnet are defined, one from `p` to `p1` with four stages and average 3.0, the other from `p` to `p2` with five stages and average 2.0.

The expressive power of CSPL is greater than that available in most modeling languages. A CSPL file exploiting structural parametrization is indeed the representation of a large class of GSPNs (at times, its graphical representation may become difficult).

We note that there is some relation between array of places and transitions and Colored PNs [19]. Both can be used to represent variations on a given population of tokens, by spatially separating them into different places or by "color-coding" them, respectively. The most natural choice among the two approaches depends on the particular system. The ability to instantiate several copies of a subnet can also be related to Colored PNs, especially when the instances are organized in an array fashion. In general, though, instances may be completely unrelated and quite different, according to the value of the parameters used during initialization.

2.3 Output measures

When performing steady-state analysis, most (G)SPN packages [6, 10] output, by default, a "standard" set of measures, such as the expected number of tokens in each place and, in some cases, the expected throughput of each timed transition. If transient analysis is possible, the same measures may be obtained, for one or more values of t, the time parameter.

While these quantities are often adequate, there are many instances where more complex measures are needed. A general expression for a steady-state measure computed by SPNP is

$$\sum_{k \in \mathcal{T}} r_k \pi_k$$

where $\mathcal{T}$ is the set of tangible markings ($\mathcal{V}$ is the set of vanishing markings), π_k is the steady-state probability of (tangible) marking k, and r_k is the reward rate in marking k.

The same expression for a transient or cumulative measure becomes, respectively:

$$\sum_{k \in \mathcal{T}} r_k \pi_k(t) \qquad \text{and} \qquad \sum_{k \in \mathcal{T}} r_k \int_0^t \pi_k(x)\, d\, x$$

where $\pi_k(t)$ is the probability of being in marking k at time t.

Sensitivity analysis computes the derivatives of these measures with respect to the independent parameter μ. The corresponding expressions for steady-state, transient, and cumulative measures are, respectively,

$$\frac{\partial}{\partial \mu} \sum_{k \in \mathcal{T}} r_k \pi_k = \sum_{k \in \mathcal{T}} \frac{\partial r_k}{\partial \mu} \pi_k + \sum_{k \in \mathcal{T}} r_k \frac{\partial \pi_k}{\partial \mu}$$

$$\frac{\partial}{\partial \mu} \sum_{k \in \mathcal{T}} r_k \pi_k(t) = \sum_{k \in \mathcal{T}} \frac{\partial r_k}{\partial \mu} \pi_k(t) + \sum_{k \in \mathcal{T}} r_k \frac{\partial \pi_k(t)}{\partial \mu}$$

$$\frac{\partial}{\partial \mu} \sum_{k \in \mathcal{T}} r_k \int_0^t \pi_k(x)\, d\, x \;=$$

$$\sum_{k \in \mathcal{T}} \frac{\partial r_k}{\partial \mu} \int_0^t \pi_k(x)\, d\, x \;+\; \sum_{k \in \mathcal{T}} r_k \frac{\partial}{\partial \mu} \int_0^t \pi_k(x)\, d\, x$$

The computation of the quantities π_k, $\pi_k(t)$, $\int_0^t \pi_k(x)\, d\, x$, $\frac{\partial}{\partial \mu}\pi_k$, $\frac{\partial}{\partial \mu}\pi_k(t)$, and $\frac{\partial}{\partial \mu} \int_0^t \pi_k(x)\, d\, x$ is described in Section 3; in this section we show how the reward rate vector can be defined in a compact way using the syntax for marking dependency.

The steady-state expected value of the marking dependent function

```
reward_type    measure1() {
  if ( mark("p1") < 10 )
    return( mark("p1") * 3 * mark("p2") );
  else
    return( 30 * mark("p2") );
}
```

is printed in SPNP by specifying

```
pr_expected("value for 1st measure",measure1);
```

The above definition of `measure1` uses the predefined function `mark` to set a reward rate in each marking. The predefined function `rate` can be used as well. For example,

```
reward_type    measure2() {
  if ( enabled("t2") )
    return( rate("t1") );
  else
    return( 0 );
}
```

can be used to compute the expected throughput of transition `t1` given that `t2` is enabled. This quantity cannot be obtained from the expected number of tokens in each place, or the expected throughput of each transition, because the events "`t1` enabled" and "`t2` enabled" may be stochastically dependent.

The predefined function `pr_expected` prints the expected value of the corresponding expression. At times, this value may need to be manipulated further before being printed:

```
measure3 = 1/expected(measure1);
```

sets `measure3` to the reciprocal of the expected value for `measure1`. This is different from the expected value of the reciprocal of `measure1`, since $1/E[X] \neq E[1/X]$. The flexibility obtained from using the C language allows us to define arbitrarily complex reward functions, and then to combine their expected values in arbitrarily complex ways. The time-dependent expected value of the expression can be printed using the same predefined function `pr_expected` when the solution method for the GSPN is set to transient analysis. The expected value of the cumulative measure and the time averaged value can be computed by using the predefined functions `pr_cum_expected` and `pr_time_avg_expected`, respectively. The derivatives of these expressions with respect to a parameter μ can be computed using the predefined functions `pr_sens_expected` and `pr_sens_cum_expected` (both the expression and its derivative must be supplied). In conclusion, the results obtained from SPNP do not need further processing.

3 Solution techniques

3.1 Steady-state solution

The traditional technique for the steady-state analysis of GSPNs is based on reduction of the underlying reachability graph by eliminating the vanishing markings. A different solution method, called PRESERVATION (in contrast to ELIMINATION) which treats the reachability graph as a semi-Markov process, has been analyzed in [8].

3.1.1 ELIMINATION

The stochastic process corresponding to a GSPN is completely described by the "extended reachability graph" (ERG), which is the reachability graph with the appropriate stochastic information associated to each arc. The ERG is reduced to an equivalent CTMC by eliminating the vanishing markings and by changing the rates between tangible markings accordingly[1, 2].

Define $P^{\mathcal{V}} = [P^{\mathcal{VV}}|P^{\mathcal{VT}}]$, the matrix describing the probability of transition from each vanishing marking to each vanishing ($P^{\mathcal{VV}}$) or tangible ($P^{\mathcal{VT}}$) marking. Define $U^{\mathcal{T}} = [U^{\mathcal{TV}}|U^{\mathcal{TT}}]$, the matrix describing the rate of transition from each tangible marking to each vanishing ($U^{\mathcal{TV}}$) or tangible ($U^{\mathcal{TT}}$) marking. $P^{\mathcal{V}}$ and $U^{\mathcal{T}}$ completely describe the ERG.

Compute $U = U^{\mathcal{TT}} + U^{\mathcal{TV}}(I - P^{\mathcal{VV}})^{-1}P^{\mathcal{VT}}$, the matrix describing the rate of transition from each tangible marking to each tangible marking in the corresponding CTMC. The diagonal of U may be nonzero because *self-transitions*[1] may already be present in $U^{\mathcal{TT}}$ or

[1] A self-transition in $U^{\mathcal{TT}}$ corresponds to the firing of a timed transition whose input and output bags coincide (its firing does not change the marking).

because *self-paths*[2] may be found during the operation $U^{\mathcal{TV}}(I - P^{\mathcal{VV}})^{-1} P^{\mathcal{VT}}$. These entries may be ignored: the infinitesimal transition rate matrix Q for the underlying CTMC is given by

$$Q_{i,j} = \begin{cases} U_{i,j} & \text{if } i \neq j \\ -\sum_{k \in \mathcal{T}, k \neq i} U_{i,k} & \text{if } i = j \end{cases}$$

The problem is then reduced to the computation of $\underline{\pi}$ from

$$\underline{\pi} Q = 0 \quad \text{subject to} \quad \sum_{k \in \mathcal{T}} \pi_k = 1$$

Steady-state analysis of the CTMC using matrix iterative methods is described in detail in [43]. In SPNP, the user can select the method to be used; currently, Successive Over Relaxation (SOR) and Gauss-Seidel are implemented. There are cases where SOR does not converge while Gauss-Seidel converges and vice versa: SPNP automatically switches from SOR to Gauss-Seidel if convergence is not obtained.

3.1.2 PRESERVATION

The ERG is a particular SMP [5] where the holding times are restricted to be either exponentially distributed or constant zero. Underlying every SMP there is a DTMC, describing transitions between states independently of time and it is well known that, given a SMP, only the vector $\underline{h}$ of the expected holding times in each state and the one-step transition probability matrix P are needed to compute the steady-state probability for each state of the SMP.

Given the matrices $P^{\mathcal{V}}$ and $U^{\mathcal{T}}$ defined earlier, $\underline{h}$ and P are obtained as:

$$h_i = \begin{cases} 0 & \text{if } i \in \mathcal{V} \\ (\sum_{k \in \mathcal{V} \cup \mathcal{T}} U^{\mathcal{T}}_{i,k})^{-1} & \text{if } i \in \mathcal{T} \end{cases}$$

$$P = \left[\frac{P^{\mathcal{V}}}{P^{\mathcal{T}}} \right] = \left[\begin{array}{c|c} P^{\mathcal{VV}} & P^{\mathcal{VT}} \\ \hline P^{\mathcal{TV}} & P^{\mathcal{TT}} \end{array} \right]$$

where $P^{\mathcal{T}}$ is obtained from $U^{\mathcal{T}}$ by normalizing the sum of each row to one ($P^{\mathcal{T}}_{i,j} = U^{\mathcal{T}}_{i,j}\, h_i$).

Self-transitions do not need to be eliminated, they can be explicitly represented and included into the computation of the holding time; analogously, self-paths do not require a particular treatment.

[2] A self-path corresponds to the firing of a timed transition in the (tangible) marking x, followed by the firing of one or more immediate transitions, such that the the final marking reached is x itself.

The steady-state probability vector $\underline{p}$ for the DTMC is computed by solving

$$\underline{p} = \underline{p}P \quad \text{subject to} \quad \sum_{k \in \mathcal{V} \cup \mathcal{T}} p_k = 1$$

and the steady-state probability vector $\underline{\pi}$ for the ERG is obtained by weighing the probabilities of the states of the DTMC according to the holding times:

$$\pi_i = \frac{p_i h_i}{\sum_{k \in \mathcal{V} \cup \mathcal{T}} p_k h_k} = \frac{p_i h_i}{\sum_{k \in \mathcal{T}} p_k h_k}$$

Since h_i is zero for $i \in \mathcal{V}$, the value obtained for the steady-state probability of the vanishing markings is zero, as it should be.

3.2 Transient Analysis

Transient analysis implemented in SPNP is based on the ELIMINATION method [33]. The underlying CTMC is obtained from the reachability graph by eliminating the vanishing markings. This elimination process is similar to that used in steady-state analysis.

Several solution methods for transient analysis of CTMCs exist. GSPNs often correspond to CTMCs with large state spaces and sparse generator matrices. UNIFORMIZATION, the analysis method implemented in SPNP, is based on the randomization of the CTMC and exploits the sparse structure of the matrix. In this method, the CTMC is reduced to a DTMC subordinated to a Poisson process. The time-dependent probabilities are given as an infinite series. In practice, this sum is carried out until an absolute error tolerance is satisfied. The number of terms required to satisfy the error tolerance can be pre-computed. This is one of the advantages of using UNIFORMIZATION. This method can easily handle large state spaces and is numerically stable but not efficient for stiff problems [35]. Computation of the cumulative probabilities for the CTMC is also based on UNIFORMIZATION [36].

The initial probability distribution for the CTMC affects the results of transient analysis. If the initial marking of the GSPN is tangible, the underlying CTMC contains a state corresponding to this marking. In this case, the initial probability for this state is 1.0 and all the other states have an initial probability of zero. However, if the initial marking is a vanishing marking, the set of tangible markings reachable from the initial marking through immediate transitions firings must be determined; each marking in this set has a non-zero initial probability [33]. SPNP automatically recognizes a vanishing initial marking and computes the initial distribution during the elimination step of the analysis.

3.3 Expected accumulated reward until absorption

If the GSPN has absorbing markings, the expected accumulated reward until absorption can be computed as

$$\sum_{k \in \mathcal{T}} r_k x_k$$

where x_k is the expected time spent in transient tangible marking k before absorption. Vector $\underline{x}$ can be computed by solving [43]

$$\underline{x} Q^{\mathcal{N}} = -\underline{\pi}(0)^{\mathcal{N}}$$

where $Q^{\mathcal{N}}$ and $\underline{\pi}(0)^{\mathcal{N}}$ are the restrictions of Q and the initial probability vector for the CTMC, $\underline{\pi}(0)$, to the set $\mathcal{N}$ of transient (non-absorbing) markings, respectively.

It is possible to compute this output measure for different reward rate definitions and for different initial probability vectors. The predefined function "set_prob0" can be used to set the initial probability individually for each marking. For example,

```
double   proportional_p1() {
  return( mark("p1") );
}
```

can be used in

```
set_prob0(proportional_p1);
```

to set the initial probability to a value proportional to the number of tokens in place p1 in each marking (a normalization assures that the probabilities sum to one); in particular, the probability is zero for the markings where p1 is empty.

If PRESERVATION is used, the expected number of visits v_k to each marking k is computed first, solving

$$\underline{v}(P^{\mathcal{N}} - I) = -\underline{p}(0)^{\mathcal{N}}$$

where $P^{\mathcal{N}}$ and $\underline{p}^{\mathcal{N}}$ are the restrictions of P and the initial probability vector for the DTMC, $\underline{p}(0)$, to $\mathcal{N}$. The expected accumulated reward is then given by

$$\sum_{k \in \mathcal{T}} v_k h_k r_k$$

3.4 Sensitivity Analysis

Sensitivity analysis implemented in SPNP is based on ELIMINATION [32]. The rates and probabilities of the transitions and their derivatives are specified as functions of an independent parameter μ in the model. SPNP automatically constructs the generator matrix as well as the derivative of this matrix for the underlying CTMC.

Steady-state sensitivity analysis for the CTMC implemented in SPNP is also based on matrix iterative methods: the steady-state probabilities and the derivatives of these probabilities are computed using either SOR or Gauss-Seidel. Transient sensitivity analysis is based on UNIFORMIZATION [16] and it has been extended for cumulative measures [32].

4 Comparison

Before concluding this presentation of SPNP, it is appropriate to compare it with similar existing tools: GreatSPN [6, 7], from the Università di Torino, Italy, and METASAN [40], from the University of Michigan at Ann Arbor and Industrial Technology Institute, also in Ann Arbor. GreatSPN is mainly based on GSPNs, but it allows constant firing times, which can results in a DSPN (solved analytically using the technique of embedded Markov chains) or in a more general SPN solved with simulation. The underlying model for METASAN is Stochastic Activity Networks (SANs), which are very similar to SPNs. METASAN can use either a Markovian analysis or a simulative approach.

The comparison is presented in a tabular form and it is based on what we believe to be key features for the type of tools considered. The characteristics of both GreatSPN and METASAN are stated to the best of our knowledge, as obtained from published literature and conversations with the authors.

	SPNP	GreatSPN	METASAN
Structural analysis	RG construction. Ability to stop execution if a user-defined "assertion" on the marking is not satisfied.	RG construction. Invariant analysis. Animation.	RG construction.
SPN advanced constructs	Arrays (of places, transitions). Subnets. Marking-dependent arc multiplicities. Marking-dependent enabling functions. General priorities.	Priorities (timed transitions cannot have priority over immediate transitions).	Cases. Input and output gates.

Flexibility	General marking-dependency. Full power of C. Possibility to - define/use variables, - read/write files, - runtime input/output.	Marking-dependent rate and probabilities	Marking-dependent rate, probabilities, and inhibitions
SPN model	GSPN only	GSPN, DSPN, DTPN	GSPN, general SPN
Solution methods:			
analytical steady-state	Gauss-Seidel, Optimal SOR. Ability to use PRESERVATION or ELIMINATION	Gaussian elimination, Gauss-Seidel. ELIMINATION only.	Gaussian elimination, Gauss-Seidel. ELIMINATION only.
analytical transient	Uniformization, cumulative Uniformization.	Matrix exponentiation.	Uniformization
simulation	N/A	Regenerative	Regenerative, independent replications
Automated sensitivity analysis	Steady-state: Gauss-Seidel, Optimal SOR Transient: Uniformization	N/A	N/A
Output measures	General rewards specification (any legal C expression containing predefined functions such as "mark" and "rate").	Limited reward specifications (see the BNF in [7]).	General rewards specification.
Interface	Textual. Full power of C.	Graphic. Generates Pascal code when marking-dependent expressions are used.	Textual. Generates C code when marking-dependent expressions are used.

5 Conclusions

We have described SPNP, a new Stochastic Petri Net Package. It is based on the GSPN model where reward rates can be specified at the net level. An important feature of this tool is the marking dependency. This is useful in developing concise models for large and complex systems. In addition, transient analysis and sensitivity analysis can be performed using SPNP.

Future enhancements to the tool include providing a graphical interface. This would greatly increase the user-friendliness of the tool. However, a major problem to be solved is the graphical representation of structural parametrizations in the GSPN. SPN simulation is also under consideration [11, 15] (non-exponentially distributed firing times would be allowed). Symbolic input of the rates and probabilities of the SPN would be useful in performing sensitivity analysis. If the rates/probabilities are specified as functions of an independent parameter μ, their derivatives can be automatically computed.

References

[1] M. Ajmone Marsan, G. Balbo, G. Ciardo, and G. Conte. A software tool for the automatic analysis of Generalized Stochastic Petri Net models. In D. Potier, editor, *Modelling Techniques and Tools for Performance Analysis*, pages 155–170, INRIA, Elsevier Science Publishers B.V. (North Holland), 1985.

[2] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance models of multiprocessor systems.* The MIT Press, Cambridge, MA 02142, USA, 1986.

[3] M. D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, C-27(6):540–547, June 1978.

[4] M. Boyd, M. Veeraraghavan, J. Dugan, and K. Trivedi. An Approach to Solving Large Reliability Models. In *Proceedings of IEEE/AIAA DASC Symposium*, San Diego, 1988.

[5] E. Çinlar. *Introduction to Stochastic Processes.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1975.

[6] G. Chiola. A software package for the analysis of Generalized Stochastic Petri Net models. In *Proceedings of the International Workshop on Timed Petri Nets*, Torino, Italy, July 1985.

[7] G. Chiola. *GreatSPN Users' Manual.* Technical Report, Dipartimento di Informatica, Università degli Studi di Torino, Torino, Italy, 1987.

[8] G. Ciardo. *Analysis of large stochastic Petri net models*. PhD thesis, Duke University, Durham, NC, USA, 1989.

[9] G. Ciardo, R. A. Marie, B. Sericola, and K. S. Trivedi. Performability analysis using semi-Markov reward processes. *IEEE Transactions on Computers*. To appear.

[10] A. Cumani. ESP - A package for the evaluation of stochastic Petri nets with phase-type distributed transitions times. In *Proceedings of the International Workshop on Timed Petri Nets*, Torino, Italy, July 1985.

[11] J. Bechta Dugan, K. S. Trivedi, R. M. Geist and V. F. Nicola. Extended Stochastic Petri Nets: Applications and Analysis. *Performance '84*. E. Gelenbe, editor, Elsevier Science Publishers, B. V. (North-Holland), Amsterdam, 1984.

[12] J. Bechta Dugan, A. Bobbio, G. Ciardo and K. S. Trivedi The Design of a Unified Package for the Solution of Stochastic Petri Net Models. In *Proceedings of the International Workshop on Timed Petri Nets*, Torino, Italy, July 1985.

[13] J. Bechta Dugan, K. S. Trivedi, M. K. Smotherman and R. M. Geist. The Hybrid Automated Reliability Predictor. *AIAA Journal of Guidance, Control and Dynamics*, May-June 1986.

[14] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi. The System Availability Estimator. In *Proceedings of the Sixteenth International Symposium on Fault-Tolerant Computing*, pages 84–89, Vienna, Austria, July 1986.

[15] P. J. Haas and G. S. Shedler. Regenerative stochastic Petri nets. *Performance Evaluation*, 6:189–204, 1986.

[16] P. Heidelberger and A. Goyal. Sensitivity analysis of continuous time Markov chains using Uniformization. In P. J. Courtois, G. Iazeolla, and O. Boxma, editors, *Proceedings of the 2nd International Workshop on Applied Mathematics and Performance/Reliability Models of Computer/Communications Systems*, Rome, Italy, May 1987.

[17] R. A. Howard. *Dynamic Probabilistic Systems, Volume II: Semi-Markov and Decision Processes*. John Wiley and Sons, New York, NY, 1971.

[18] M. C. Hsueh, R. K. Iyer, and K. S. Trivedi. Performability modeling based on real data: a case study. *IEEE Transactions on Computers*, Apr. 1988.

[19] K. Jensen. Coloured Petri nets. *Applications and Relationships to Other Models of Concurrency, Advances in Petri Nets 1986 - Part I, Lecture Notes of Computer Science*, 254:248–299, 1987 W. Brauer, W. Reisig and G. Rozenberg, editors, Springer-Verlag.

[20] B. W. Kernighan and D. M. Ritchie. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1978.

[21] W. Kleinoder. Evaluation of Task Structures for Hierarchical Multiprocessor Systems. *Modeling Techniques and Tools for Performance Analysis*. D. Potier, editor, Elsevier Science Publishers. B. V. (North-Holland) 1985.

[22] K. C. Y. Kung. Concurrency in Parallel Processing Systems. *Ph.D Dissertation, Dept. of Computer Science, UCLA*. 1984.

[23] S. S. Lavenberg, editor. *Computer Performance Modeling Handbook*. Academic Press, 1983.

[24] V.O. Li and J.A. Silvester. Performance Analysis of Networks with Unreliable Components. *IEEE Transactions on Communications*, COM-32(10):1105-1110, October 1984.

[25] E. D. Lazowska, J. Zahorjan, G. S. Graham and K. C. Sevcik. *Quantitative System Performance*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1984.

[26] Victor W. K. Mak, Performance Prediction of Concurrent Systems. *Ph.D Dissertation, Computer Systems labaratory, Stanford University*, Dec. 1987.

[27] J. F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, C-29(8):720–731, Aug. 1980.

[28] J. F. Meyer. Closed-Form Solutions of Performability. *IEEE Transactions on Computers*. vol. C-31, No. 7, July 1982.

[29] J. F. Meyer. Performability Modeling of Distributed Real-Time Systems. *Mathematical Computer Performance and Reliability*, G. Iazeolla, P. J. Courtois and A. Hordijk, editors, Elsevier Science Publishers. B. V. (North-Holland), 1984.

[30] J. F. Meyer, A. Movaghar, and W. H. Sanders. Stochastic activity networks: structure, behavior, and application. In *Proceedings of the International Workshop on Timed Petri Nets*, Torino, Italy, July 1985.

[31] R. R. Muntz, E. de Souza e Silva, and A. Goyal. Bounding Availability of Repairable Computer Systems. In *Proceedings of 1989 ACM SIGMETRICS and PERFORMANCE'89 International Conference on Measurement and Modeling of Computer Systems*, Berkeley, CA, May 1989.

[32] J. K. Muppala and K. S. Trivedi. GSPN Models: Sensitivity Analysis and Applications. In preparation.

[33] J. K. Muppala and K. S. Trivedi. GSPN Models: Transient Analysis and Applications. In preparation.

[34] J. L. Peterson. *Petri Net Theory and the Modeling of Systems.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.

[35] A. L. Reibman and K. S. Trivedi. Numerical transient analysis of Markov models. *Computers and Operations Research*, 15(1):19–36, 1988.

[36] A. L. Reibman and K. S. Trivedi. Transient analysis of cumulative measures of Markov model behavior. *Stochastic Models.* To appear.

[37] J. T. Robinson. Some Analysis Techniques for Asynchronous multiprocessor Algorithms. *IEEE Trans. Software Eng.* vol. SE-5, January 1979.

[38] R. A. Sahner and K. S. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Transactions on Software Engineering*, Oct. 1987.

[39] R. A. Sahner and K. S. Trivedi. Reliability modeling using SHARPE. *IEEE Transactions on Reliability*, R-36(2):186–193, June 1987.

[40] W. H. Sanders and J. F. Meyer. METASAN: a performability evaluation tool based on Stochastic Activity Networks. In *Proceedings of the ACM-IEEE Comp. Soc. Fall Joint Comp. Conf.*, Nov. 1986.

[41] M. L. Shooman. *Probabilistic Reliability: An Engineering Approach.* McGraw-Hill, New York, 1968.

[42] R. M. Smith, K. S. Trivedi, and A. V. Ramesh. Performability analysis: measures, an algorithm, and a case study. *IEEE Transactions on Computers*, Apr. 1988.

[43] W. Stewart and A. Goyal. *Matrix methods in large dependability models.* Technical Report RC-11485, IBM T.J. Watson Res. Center, Yorktown Heights, NY, 10598, Nov. 1985.

[44] K. S. Trivedi. *Probability & Statistics with Reliability, Queueing, and Computer Science Applications.* Prentice-Hall, Englewood Cliffs, NJ, USA, 1982.

[45] M. Veeraraghavan and K. S. Trivedi, Hierarchical Modeling for Reliability and Performance Measures. *Concurrent Computations: Algorithms, Architecture and Technology* S. K. Tewksbury, B. W. Dickson and S. C. Schwartz, editors, Plenum Press, New York, 1987.