SNS 1.0: Synchronized Network Solver

Marco Tilgner and Yukio Takahashi
Dept. of Mathematical and Computing Sciences
Tokyo Institute of Technology
2-12-1 Ookayama, Meguro-ku, Tokyo 152, JAPAN
marco,yukio@is.titech.ac.jp

Gianfranco Ciardo
Dept. of Computer Science
College of William and Mary
Williamsburg, VA 23187-8795, USA
ciardo@cs.wm.edu

Abstract

We present SNS, a Solver for Synchronized Networks modelled as generalized stochastic Petri nets. General enabling and firing functions and reward measures are specified in the C programming language, providing extensive flexibility with respect to conventional net descriptions.

Unstructured and structured steady-state and transient solutions are possible, to compute the specified rate-based and impulse-based reward measures. We allow the model to have both immediate and timed synchronizing transitions. Furthermore, the user can solve numerical linear algebra problems by a variety of algorithms based on solution vectors and matrices. We apply SNS on a case study of kanban systems and give numerical results. In the conclusion, we point out ideas to expand the package.

Keywords: generalized stochastic Petri nets, numerical solution of (structured) Markov chains, Markov reward models, Kronecker algebra, numerical linear algebra.

1 Introduction

Generalized stochastic Petri nets (GSPNs) [4] are very well suited to model manufacturing systems, since they can represent the flow of parts in the factory, the required synchronizations, and the potential concurrency inherent in such systems. Conventional packages [3, 9] for the solution of GSPNs, however, are limited by

the large amount of memory required to store the infinitesimal generator of the underlying continuous-time Markov chain (CTMC).

SNS (Synchronized Network Solver) is an advanced package which allows a *compositional* description and solution of GSPNs, thus the solution of larger models. Compared to the QPN tool [1], synchronization between submodels can occur through both timed or immediate transitions, and general reward structures, including impulse-based rewards, can be specified at the net level [8]. In addition, SNS allows the user to experiment with solution approaches, by giving access to the low level of the underlying CTMC or discrete-time Markov chain (DTMC).

The main features of SNS are:

- The GSPN is described using the C language for textual input.
- The usual reachability graph exploration of the GSPN is performed, so that reachability questions such as liveness or absence of deadlocks can be answered by exhaustive search.
- In addition to conventional "flat" approaches, models composed of sub-GSPNs connected by synchronizing transitions are automatically translated into structured Markov chains [8, 11, 13]. In this case, the storage of the infinitesimal generator is implicit and it is not a critical factor.
- Internally, the solution can be based on either an embedding of the process, resulting in a DTMC, or on the more common CTMC. These two approaches are known as "preservation" and "elimination" (of the vanishing markings), respectively [7].
- Both steady-state (with either approach) and transient analysis (with elimination) can be performed. These result in the computation of expected reward measures.
- The feasibility of an exact analysis is restricted by the size of the iteration vectors. On a modern workstation, models with state spaces having up to a few million reachable markings can be solved in a matter of hours. For even larger state spaces, it is possible to compute an approximate solution, where these vectors are decomposed into smaller ones, which are then updated iteratively. "Cross-aggregation" [15] and a formalized scheme proposed in [16] are two approaches for this type of models.
- SNS uses the linear algebra package *Meschach* [14]. Thus the user has access to the underlying Markov chain: matrix and vector operations, and iteration schemes on these can be specified and solved by direct methods, the power method, the Jacobi method with overrelaxation, or projection methods for dense or sparse matrices.

We present the first version of SNS, the concept of composition, and provide numerical results for a case study of kanban systems.

2 Theoretical background

We briefly summarize the ideas behind the hierarchical composition of GSPN and the structure of their underlying Markov chain. For more information on GSPNs, the reader is referred to [4, 5]. Sets are denoted by upper case calligraphic letters (e.g., $\mathcal{A}$), vectors and matrices are denoted by lower and upper case bold letters, respectively (e.g., $\mathbf{a}$, $\mathbf{A}$). In the following, we use these terms:

- $\mathcal{S}$: the reachability set.
- $\mathcal{S}_T$ and $\mathcal{S}_V$: a partition of $\mathcal{S}$ into tangible and vanishing markings ($\mathcal{S}_T \cup \mathcal{S}_V = \mathcal{S}$).

2.1 GSPN composition

In SNS, a GSPN model can be described as a set of interacting sub-GSPNs. In each sub-GSPN, transitions are characterized as either internal or synchronizing. Internal transitions are local to a particular sub-GSPN, while synchronizing transitions are (or can be, if we consider each sub-GSPN as a plug-in module) shared, or "merged", among two or more sub-GSPNs and allow to model interactions between them. Each sub-GSPN considered in isolation must fulfill three conditions:

1. Each place has at least one input and one output transition (P-restrictedness).
2. There is at least one synchronizing transition.
3. Each synchronizing transition has at least one arc (input or output) connecting it to a place.

However, these conditions are not sufficient to guarantee that the overall model has a finite state space; this is an additional global requirement that the model must satisfy. In particular, if an internal or an non-merged synchronizing transition with no input arcs exist, the model is "open". The time associated to such a transition is interpreted as the interarrival time in a Poisson arrival stream from the external environment. Some mechanism must then be present to prevent the transition from increasing without bound the number of tokens in its output place(s), which would cause the state space to be infinite; usually this is accomplished using marking-dependent transition rates.

Fig. 1 shows a sub-GSPN. Boxes and thin bars represent timed and immediate transitions, respectively. In addition, synchronizing transitions are shown in grey. Different GSPNs obtained by combining four copies of this sub-GSPN are shown in Fig. 2, where the grey arcs indicate which synchronized transitions are merged. In the first case, Fig. 2.(1), t_{out_1} is merged with either t_{in_2} or t_{in_3}, with probability 0.3 and 0.7, respectively. The semantic of this "probabilistic merging" is exactly as if sub-GSPN 1 had two copies of transition t_{out_1}, identified as t'_{out_1} and t''_{out_1}, with rates equal 3/10 and 7/10 of the original rate, and these were independently merged with t_{in_2} and t_{in_3}, respectively. Thus, the marking of sub-GSPNs 2 and

3 affects the actual routing probabilities (which are in general not equal to 3/10 and 7/10), since if, for example, sub-GSPN 3 is not ready to synchronize with sub-GSPN 1, the token in p_{out_1} will go to sub-GSPN 2 with probability one. Analogously, sub-GSPN 4 has two copies of t_{in_4}: t'_{in_4} merged with t_{out_2} and t''_{in_4} merged with t_{out_3}. In the second case, the tandem GSPN of Fig. 2.(2), output and input transitions of subsequent sub-GSPNs are merged: t_{out_1} with t_{in_2}, t_{out_2} with t_{in_3}, and t_{out_3} with t_{in_4}. In the third case, the fork/join configuration of Fig. 2.(3), t_{out_1} is merged with t_{in_2} and t_{in_3}. The same is true for t_{out_2}, t_{out_3}, and t_{in_4}. The fourth case of Fig. 2.(4) merges transitions t_{out_4} with t_{in_1} in addition to case 2, such that the GSPN becomes closed. The last two cases of Fig. 2.(5) and (6) correspond to the first and third case, respectively, except that the merged synchronizing transitions have been made immediate. Note that the fifth case is fundamentally different from the first case, since it allows the simultaneous firing of multiple synchronizing transitions.

In all but the fourth case, all merged synchronizing transitions have been drawn as local, so that the resulting GSPN can in turn act as a sub-GSPN with two synchronizing transitions, t_{in_1} and t_{out_4}, in an higher-level model, and so on, in a hierarchical fashion. Alternatively, one or both of the merged transitions could still be considered synchronizing, to allow merging them further at even higher levels.

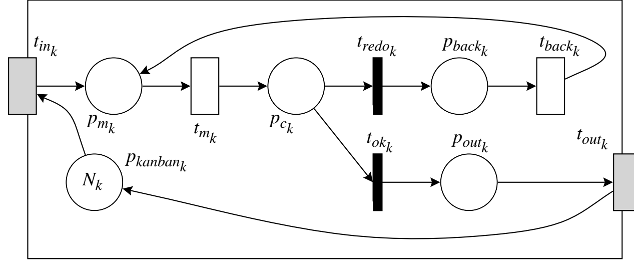


Figure 1: A sub-GSPN for our kanban system.

2.2 Structured state space

As long as the state space is finite, we can in principle build the overall transition matrix for the CTMC underlying such a model, but the memory requirements often become excessive. To tackle this problem we use a structural representation based on Kronecker expressions. Internal transitions describe the possible changes of marking within one and only one basic sub-GSPN in an asynchronous fashion, that is, independently of the state and activities in other sub-GSPNs. Their effect is captured by small matrices composed using Kronecker sums ($\bigoplus$). The effect of each synchronizing transition is instead captured by a Kronecker product ($\bigotimes$) of other small matrices, which model their synchronous effect on the state space (the

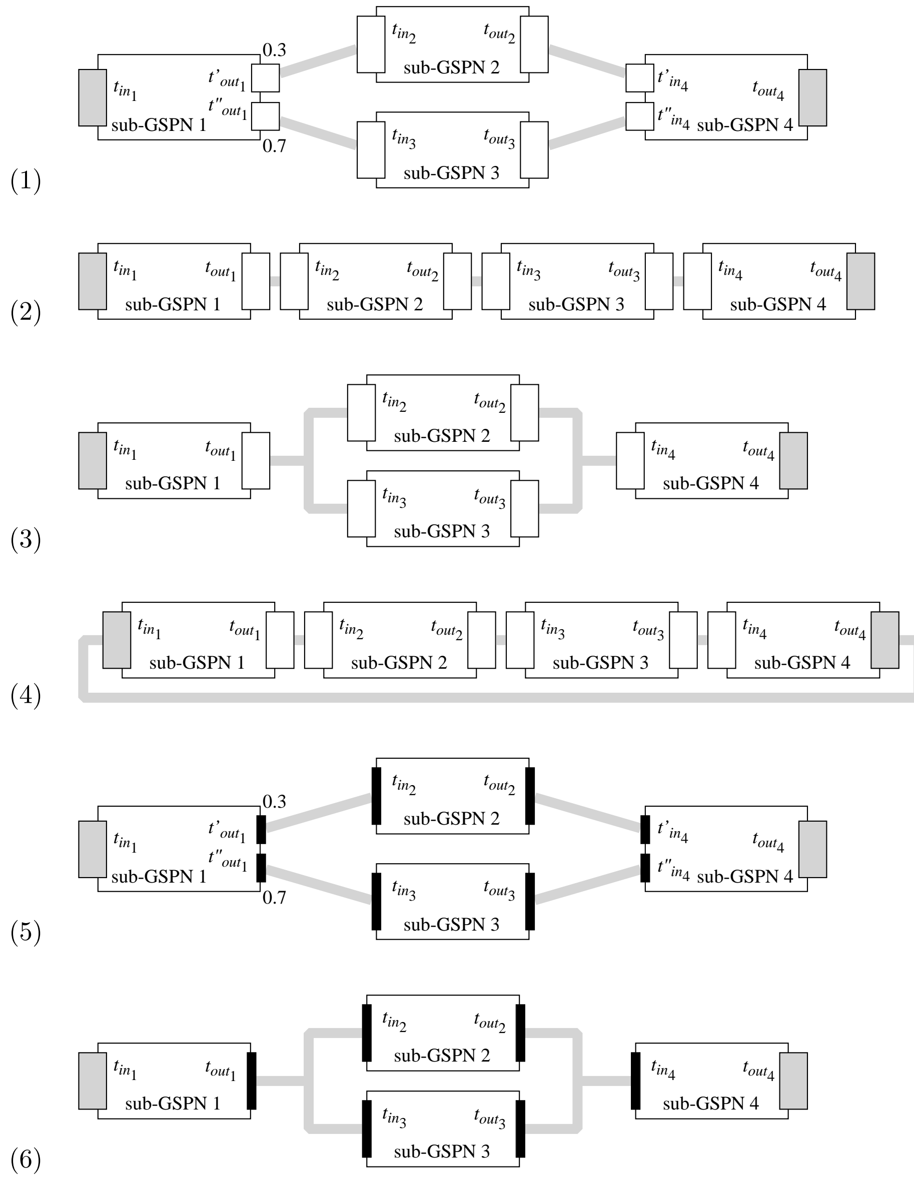


Figure 2: Various arrangements of four instances of our sub-GSPN.

simultaneous change of marking in two or more sub-GSPNs). At each hierarchical

level, we can define a matrix $\mathbf{Q}'$ as

$$\mathbf{Q}' = \bigoplus_{k=1}^{K} \mathbf{Q}^k + \sum_{t_j \in \mathcal{T}_{synch}} \boldsymbol{\lambda}(t_j) \bigotimes_{k=1}^{K} \mathbf{W}^{k,j} - \sum_{t_j \in \mathcal{T}_{synch}} \boldsymbol{\lambda}(t_j) \bigotimes_{k=1}^{K} \mathbf{D}^{k,j} \qquad (1)$$

where $\mathbf{Q}^k$ is an infinitesimal generator describing the local transitions for the k-th sub-GSPN (including the effect of immediate transitions), $\mathcal{T}_{synch}$ is the set of synchronizing transitions, which must be timed. For immediate synchronizing transitions, an embedded DTMC, not the underlying CTMC, is used, resulting in a different formula [8]. $\boldsymbol{\lambda}(t_j)$ is a "constant reference rate" for transition t_j, $\mathbf{W}^{k,j} \geq 0$ is a "corrective" matrix which captures both the effect of transition t_j on the k-th sub-GSPN and the effect of the marking of this sub-GSPN on the enabling and rate of t_j, while $\mathbf{D}^{k,j}$ is simply a diagonal matrix equal to the sum of the elements on each row of $\mathbf{W}^{k,j}$ (hence $\mathbf{Q}^{k,j} = \mathbf{W}^{k,j} - \mathbf{D}^{k,j}$ is also an infinitesimal generator). The Kronecker operations create a "potential state space" $\mathcal{S}'$ for the GSPN equal to the Cartesian product of the state spaces for each sub-GSPN. The actual infinitesimal generator $\mathbf{Q}$ for the overall model can be obtained from $\mathbf{Q}'$ by eliminating the rows and columns corresponding to the unreachable markings in $\mathcal{S}' \setminus \mathcal{S}$ [8, 13].

2.3 Numerical solution

The stochastic process underlying a GSPN can be characterized as an independent semi-Markov process where the sojourn time in each state is either exponentially distributed or zero. We can study this process by eliminating the zero-sojourn states (vanishing markings), thus obtain a CTMC where only the tangible markings are represented. This is known as the "elimination" approach and is commonly used for both steady-state and transient solutions.

However, in conjunction with our structured approach, elimination requires that all synchronizing transitions be timed. If this is not the case, we showed how "preservation" can be employed, to perform a steady-state analysis [8]. With preservation, the semi-Markov process is embedded at each change of marking and a DTMC is obtained. Only at the end the sojourn times in states are used to compute the actual marking probabilities (vanishing markings have zero probability in the semi-Markov process, but not in the DTMC).

2.3.1 Elimination method (CTMC)

Let $\boldsymbol{\pi} \in \mathbb{R}^{|\mathcal{S}_T|}$ be the probability vector for the tangible markings. Then, the transient probabilities $\boldsymbol{\pi}(t)$ are the solution of

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0) \cdot e^{\mathbf{Q}t} = \boldsymbol{\pi}(0) \cdot \sum_{k=0}^{\infty} \frac{(\mathbf{Q}t)^k}{k!}, \qquad (2)$$

where $\boldsymbol{\pi}(0)$ is a vector of appropriate size describing the probability of being in each state at time 0. If the initial marking $\mathbf{m}^0$ is tangible, $\boldsymbol{\pi}(0)$ is simply equal

to one in the entry corresponding to $\mathbf{m}^0$, and to zero elsewhere. The steady-state probability vector $\boldsymbol{\pi}$ is then defined as $\boldsymbol{\pi} = \lim_{t\to\infty} \boldsymbol{\pi}(t)$.

In practice, $\boldsymbol{\pi}(t)$ can be computed using the Uniformization algorithm [12], and $\boldsymbol{\pi}$ is computed as the solution of

$$\boldsymbol{\pi} \cdot \mathbf{Q} = \mathbf{0} \qquad \text{subject to the normalization} \qquad \boldsymbol{\pi} \cdot \mathbf{1} = 1.$$

2.3.2 Preservation method (DTMC)

Preservation uses the transition probability matrix $\mathbf{P}$ of the embedded DTMC, expressing the probability of going in one firing from any marking $\mathbf{m} \in \mathcal{S}$ to any other marking $\mathbf{n} \in \mathcal{S}$, regardless of whether they are tangible or vanishing. The steady-state probability vector $\boldsymbol{\gamma} \in \mathbb{R}^{|\mathcal{S}|}$ of the embedded DTMC is computed as:

$$\boldsymbol{\gamma} \cdot \mathbf{P} = \boldsymbol{\gamma} \qquad \text{subject to the normalization} \qquad \boldsymbol{\gamma} \cdot \mathbf{1} = 1.$$

Then, using the holding time vector $\mathbf{h}$, defined as

$$\mathbf{h}_{\mathbf{m}} = \begin{cases} -\mathbf{Q}^{-1}_{\mathbf{m},\mathbf{m}} & \text{if } \mathbf{m} \in \mathcal{S}_T \\ 0 & \text{otherwise} \end{cases},$$

we can compute both $\boldsymbol{\pi}$ and the rate at which each marking (tangible or not) is entered in steady-state, $\boldsymbol{\phi}$, (this is needed to compute impulse rewards, see below):

$$\forall \mathbf{m} \in \mathcal{S}_T,\ \boldsymbol{\pi}_{\mathbf{m}} = \frac{\gamma_{\mathbf{m}} \cdot \mathbf{h}_{\mathbf{m}}}{\sum_{\mathbf{n}\in\mathcal{S}_T} \gamma_{\mathbf{n}} \cdot \mathbf{h}_{\mathbf{n}}}, \quad \forall \mathbf{m} \in \mathcal{S},\ \boldsymbol{\phi}_{\mathbf{m}} = \frac{\gamma_{\mathbf{m}}}{\sum_{\mathbf{n}\in\mathcal{S}_T} \gamma_{\mathbf{n}} \cdot \mathbf{h}_{\mathbf{n}}}.$$

2.3.3 Reward measures

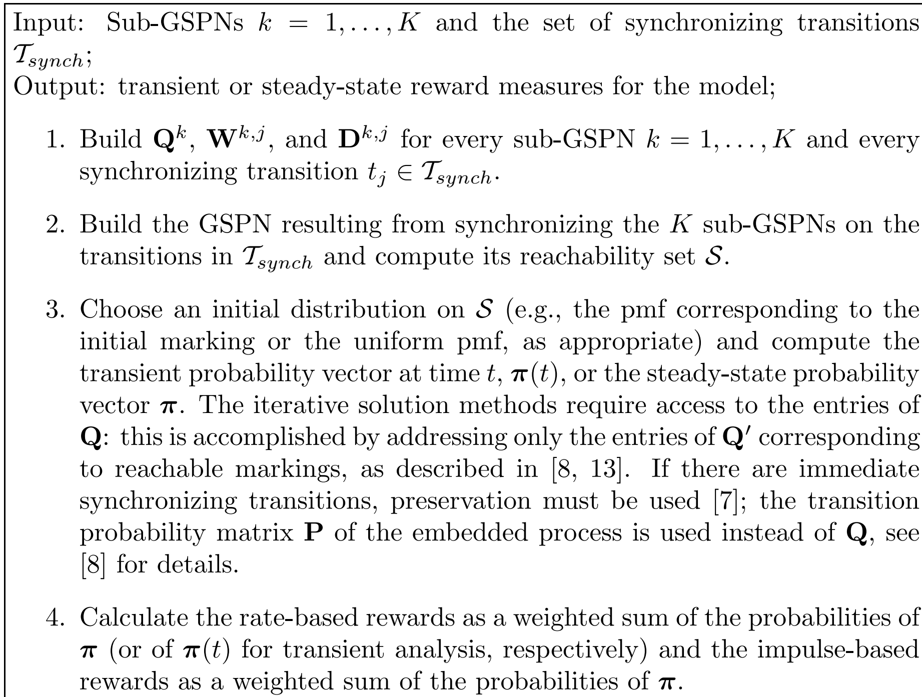
We can specify a quantity of interest for the GSPN using a *reward structure* $(\rho, \mathbf{r})$, where $\rho(\mathbf{m})$ is the *reward rate* gained while the GSPN is in marking $\mathbf{m}$, and $\mathbf{r}_j(\mathbf{m})$ is the *reward impulse* gained when transition $t_j \in \mathcal{T}$ fires in marking $\mathbf{m}$. The expected reward rate in steady state is then

$$\sum_{\mathbf{m}\in\mathcal{S}_T} \boldsymbol{\pi}_{\mathbf{m}} \rho(\mathbf{m}) + \sum_{\mathbf{m}\in\mathcal{S}} \sum_{t_j\in\mathcal{E}(\mathbf{m})} \Phi_{j,\mathbf{m}} \mathbf{r}_j(\mathbf{m}), \tag{3}$$

where $\Phi_{j,\mathbf{m}}$ is the rate at which transition t_j fires in steady state in marking $\mathbf{m}$ and $\mathcal{E}(\mathbf{m})$ is the set of enabled transitions in marking $\mathbf{m}$. If we let $\boldsymbol{\phi} \in \mathbb{R}^{|\mathcal{S}|}$ be the vector describing the rate at which each marking is entered in steady state, $\Phi_{j,\mathbf{m}}$ is obtained as

$$\Phi_{j,\mathbf{m}} = \boldsymbol{\phi}_{\mathbf{m}} \frac{\mathbf{w}_j(\mathbf{m})}{\sum_{t_l\in\mathcal{E}(\mathbf{m})} \mathbf{w}_l(\mathbf{m})},$$

where $\mathbf{w}_j(\mathbf{m})$ is the rate of transition t_j in tangible marking $\mathbf{m}$, or the weight (unnormalized probability) of transition t_j in vanishing marking $\mathbf{m}$. With preservation, the computation of $\boldsymbol{\phi}$ is straightforward. With elimination, we can observe


Input: Sub-GSPNs $k = 1, \ldots, K$ and the set of synchronizing transitions $\mathcal{T}_{synch}$;
Output: transient or steady-state reward measures for the model;

1. Build $\mathbf{Q}^k$, $\mathbf{W}^{k,j}$, and $\mathbf{D}^{k,j}$ for every sub-GSPN $k = 1, \ldots, K$ and every synchronizing transition $t_j \in \mathcal{T}_{synch}$.

2. Build the GSPN resulting from synchronizing the K sub-GSPNs on the transitions in $\mathcal{T}_{synch}$ and compute its reachability set $\mathcal{S}$.

3. Choose an initial distribution on $\mathcal{S}$ (e.g., the pmf corresponding to the initial marking or the uniform pmf, as appropriate) and compute the transient probability vector at time t, $\boldsymbol{\pi}(t)$, or the steady-state probability vector $\boldsymbol{\pi}$. The iterative solution methods require access to the entries of $\mathbf{Q}$: this is accomplished by addressing only the entries of $\mathbf{Q}'$ corresponding to reachable markings, as described in [8, 13]. If there are immediate synchronizing transitions, preservation must be used [7]; the transition probability matrix $\mathbf{P}$ of the embedded process is used instead of $\mathbf{Q}$, see [8] for details.

4. Calculate the rate-based rewards as a weighted sum of the probabilities of $\boldsymbol{\pi}$ (or of $\boldsymbol{\pi}(t)$ for transient analysis, respectively) and the impulse-based rewards as a weighted sum of the probabilities of $\boldsymbol{\pi}$.

Figure 3: Algorithm for the numerical analysis of a structured GSPN.

that, for $\mathbf{m} \in \mathcal{S}_T$, $\boldsymbol{\phi}_{\mathbf{m}} = \boldsymbol{\pi}_{\mathbf{m}} \cdot \sum_{t_l \in \mathcal{E}(\mathbf{m})} \mathbf{w}_l(\mathbf{m}) = \boldsymbol{\pi}_{\mathbf{m}} / \mathbf{h}_{\mathbf{m}}$. However, for $\mathbf{m} \in \mathcal{S}_V$, the computation is more involved [7, 8].

If no reward impulse is defined for immediate transitions, then Eq. (3) reduces to

$$\sum_{\mathbf{m} \in \mathcal{S}_T} \boldsymbol{\pi}_{\mathbf{m}} \left(\rho(\mathbf{m}) + \sum_{t_j \in \mathcal{E}(\mathbf{m})} \mathbf{w}_j(\mathbf{m}) \mathbf{r}_j(\mathbf{m}) \right).$$

If the reward structure $(\rho, \mathbf{r})$ can be defined locally for each sub-GSPN, the GSPN marking $\mathbf{m}$ is replaced by the local marking $\mathbf{m}_k$ of the corresponding sub-GSPN k, in the formula above. Then, $\rho(\mathbf{m}_k)$ and $\mathbf{r}(\mathbf{m}_k)$ denote local reward rates and impulses, respectively, and $\boldsymbol{\pi}_{\mathbf{m}_k}$, the marginal state probabilities for sub-GSPN k, is used instead of $\boldsymbol{\pi}_{\mathbf{m}}$. In this case, reward measures are also computable in a compositional fashion.

SNS computes measures on $\boldsymbol{\pi}(t)$, $\boldsymbol{\pi}$ and $\boldsymbol{\phi}$. The algorithm for the numerical analysis is summarized in Fig. 3. Step 2 is required only if the state space $\mathcal{S}$ of the overall model is a strict subset of $\mathcal{S}'$. If the two sets instead coincide, the solution approach is simpler, since it becomes completely compositional.

Remark: Service time distributions of transitions are assumed exponential. This constraint can be relaxed to phase-type distribution by an algorithm known

Option -m	Description
0.(—)	Unstructured GSPN: CTMC MGCR
1.(—)	Unstructured GSPN: CTMC LU
2.(—)	Unstructured GSPN: CTMC POWER
3.(8.)	Unstructured GSPN: CTMC (DTMC) JOR
4.(—)	Unstructured GSPN: CTMC UNIFORM
10.(12.)	Reachability sub-GSPNs: CTMC (DTMC)
11.(13.)	Reachability GSPN: CTMC (DTMC)
20.(—)	Structured GSPN: CTMC POWER, diagonal stored
21.(—)	Structured GSPN: CTMC POWER, diagonal recomputed
22.(32.)	Structured GSPN: CTMC (DTMC) JOR, diagonal stored
23.(33.)	Structured GSPN: CTMC (DTMC) JOR, diagonal recomputed
24.(—)	Structured GSPN: CTMC UNIFORM, diagonal stored
25.(—)	Structured GSPN: CTMC UNIFORM, diagonal recomputed
40.	Approximation: CTMC Cross aggregation
50.	Linear Algebra: Meschach

Table 1: SNS menu

from the package ESP [10]. We omit this discussion here, since numerical analysis of the kanban systems in our case study would become even more time- and memory-consuming due to the additional state space growth.

3 SNS implementation

3.1 Execution and textual output

The command to run SNS is *sns [-h] [-m] [-n] [-s] [-a] [-l] [-o]*. Option *-h* serves for help and prints the SNS menu as given in Table 1. Option *-m* executes the computations described in the menu. Option *-n* specifies the number of sub-GSPNs for the structured solution. Option *-s* determines the maximum number of states as a bound for the reachability search. Option *-a* is an accelerated algorithm for GSPNs with symmetric product state space, i.e. for $\mathcal{S} = \mathcal{S}'$. Option *-l* can be set to indicate that the reward measures are local to each sub-GSPN. Option *-o* specifies whether solution vectors or generator matrices are printed into corresponding files. If the user chooses to perform a numerical analysis using the package Meschach [14], these matrices can be read into memory again by appropriate routines.

The standard solution methods for the steady state analysis of unstructured GSPNs (conventional GSPN models) are direct (Gaussian elimination, LU decomposition), projection (Generalized conjugate residual algorithm, MGCR), power, or Jacobi with overrelaxation. The uniformization method is used for transient solution. To analyze a structured GSPN, the reachability sets and the genera-

Type	File $*.sns$	Contents
SPMAT*	$k$int_tim.sns	$\mathbf{Q}^k$ for timed local trans. of sub-GSPN k
SPMAT*	$k$int_imm.sns	$\mathbf{Q}^k$ for imm. local trans. of sub-GSPN k
SPMAT*	$k$sync_tim_j.sns	$\mathbf{W}^{k,j}$ for timed synch. trans. j of sub-GSPN k
SPMAT*	$k$sync_imm_j.sns	$\mathbf{W}^{k,j}$ for imm. synch. trans. j of sub-GSPN k
SPMAT*	$k$rew_rate.sns	rate-based local rewards for sub-GSPN k
SPMAT*	$k$rew_impuls.sns	impulse-based local rewards for sub-GSPN k
VEC*	rew_rate$i$.sns	ith global rate-based reward for the GSPN
VEC*	rew_impulse$i$.sns	ith global impulse-based reward for the GSPN
VEC*	0/$k$reach_tan.sns	tang. reach. set of GSPN (0) or sub-GSPN k
VEC*	0/$k$reach_van.sns	van. reach. set of GSPN (0) or sub-GSPN k
VEC*	0/$k$prob.sns	state prob. for the GSPN (0) or sub-GSPN k
SPMAT*	generator.sns	infinitesimal generator $\mathbf{Q}$ of the GSPN
SPMAT*	incidence.sns	incidence matrix of the GSPN
Text	output.sns	computation time, required memory, results

Table 2: SNS output files

tor matrices for the individual sub-GSPNs and the reachability set for the overall model are built first (the overall reachability set is not explicitly built if it is known to be the exact Cartesian product of the local state spaces). Then, either the elimination or the preservation methods are used. The structured GSPN can be solved as a CTMC using the power, Jacobi with overrelaxation, or uniformization methods. To accelerate the computation, the diagonal can be stored explicitly instead of being obtained through Kronecker operators, but this increases the memory requirements. The DTMC can be solved using the Jacobi method with overrelaxation. Finally, approximate solutions are provided by the Cross-aggregation method [15].

Transition matrices, reward, and probability vectors and reachability sets can be written to files $*.sns$ as given in Table 2. VEC*, IVEC*, and SPMAT* are Meschach data structures for double or integer vectors and sparse double matrices, respectively. UCVEC* is used by SNS for unsigned character vectors. Bookkeeping information about the numerical solution, such as timing, memory requirements, and results, is written to a file *output.sns*.

3.2 Textual input

3.2.1 SNS

The semantics of the Petri net model and the parameters specifying the numerical solution are given in an input file in C syntax. The number of rate-based and impulse-based rewards, the types of transitions, and the bounds on the places are set within the function *sns_parameter*(). The bounds do not have to be strict, they

Type	Function	Used to define:
void*	sns_parameter()	parameters for solution and ordered state space generation;
void*	sns0/k_enable()	enabling of transitions for GSPN/sub-GSPN;
void*	sns0/k_fire()	firing of transitions for GSPN/sub-GSPN;
void*	sns0/k_initial()	initial marking for GSPN/sub-GSPN;
double	sns0/k_reward()	reward measures for GSPN/sub-GSPN;

Table 3: SNS functions

are only needed for state lexicographic enumeration. The relevant data structures are explained in Table 4. Additionally, global solution parameters such as the maximum number of iterations, the convergence criterion, or the overrelaxation factor for JOR, might be re-set by the user.

First, a model can be described by a conventional GSPN. The enabling and firing rules, the initial marking, and the reward measures are defined in corresponding functions $sns0_*()$, see Table 3. As an example, the appendix illustrates the input file for the GSPN in Fig. 1. Once compiled, it can be solved exactly as an unstructured GSPN (using options $-m[0-8]$ of the SNS menu).

Instead, if we describe a model by a structured GSNP composed of at least two sub-GSPNs, we define the enabling and firing rules, the initial marking, and the reward measures for each sub-GSPN k analogously to those of a conventional GSPN, as mentioned above. For example, the structured GSNP of four sub-GSPNs in Fig. 2 is given by functions $snsk_*()$ for each sub-GSPN $k \in \{1, 2, 3, 4\}$ (see the functions $sns0_*()$ in the appendix). In addition to the specification of a conventional GSPN, the rates of synchronizing transitions are set within the function *sns_parameter*(). After compilation, we compute the reachability sets $\mathcal{S}$ of sub-GSPNs (using options $-m[10, 12]$) for either the elimination or preservation method. If the state space $\mathcal{S}$ of the overall model is a strict subset of $\mathcal{S}'$, we must compute the reachability set $\mathcal{S}_T$ of the GSPN for the elimination method and sets $\mathcal{S}_T$ and $\mathcal{S}_V$ for the preservation method (using options $-m[11, 13]$). Finally, options $-m[20-33]$ solve the structured GSPN exactly and option $-m[40]$ approximately.

3.2.2 Matrix computations using Meschach

SNS is linked to the linear algebra package *Meschach* [14]. The library of algorithms often needed in numerical linear algebra problems is briefly shown in Table 5. The routines are written in the C programming language and are all available in their original version. Any linear algebra problem might be specified within a function *meschach*(). The main achievement of the library Meschach is to offer a good compromise between efficient but inflexible Fortran programs and flexible but resource-consuming interpreter-based programs (like Mathematica or Mat-

Type	Name	Global solution variables
int	MAX_ITER	max. no. of iter. (default 1000);
double	CONVERGE	convergence criterion (default 10^{-6});
double	RELAXATION	overrelaxation for JOR (default 0.9);
Type	Name	Within sns_parameter
IVEC**	sns_place_bound	marking bounds for places;
IVEC**	sns_transition_type	type of trans. for GSPN/sub-GSPNs;
	[0-99]	imm. local trans.;
	[100-199]	imm. preserved local trans.;
	[200-299]	imm. preserved sync trans.;
	[500-599]	timed local trans.;
	[600-699]	timed sync trans.;
IVEC*	sns_sync_rate	rate of synch. trans. for the GSPN;
IVEC*	sns_reward_rate_no	no. of GSPN/sub-GSPNs rate rew.;
IVEC*	sns_reward_impuls_no	no. of GSPN/sub-GSPNs impulse rew.;
Type	Name	In sns0/k_enable,_fire,_initial,_reward
UCVEC*	place	vector of places;
return	timed(double)	firing rate of timed trans.;
return	immediate(double)	choice probability of imm. trans.;
return	measure(double)	reward measure;

Table 4: SNS data structures

lab). SNS uses predefined Meschach data structures for vectors, sparse matrices, or iteration schemes and thus provides a direct interface for the user, supporting the application of Meschach routines, and for the designer, easing implementation of new algorithms. For example, on the incidence matrix of the GSPN (file *incidence.sns*) transition and place invariants, the fundamental circuit matrix, or the firing count vector can be computed for structural analysis. Also, phase type distributions might be specified later within the textual input, and the user can define his own iterative procedures on the SNS vectors, matrices and, iterative data structures for alternative approximation schemes. The function *meschach*() is executed by option -m50 of the menu. In particular, we like to point out Meschach's sparse iterative solutions such as Sonneveld's Conjugate Gradients Squared (CGS) method, the LSQR method of Paige and Saunders, or the Generalized Minimal RESidual method (GMRES) of Saad and Schultz. The Krylov subspace methods cover the Lanczos and Arnoldi algorithms.

Basic dense	Dense factorization	Sparse and iterative
copy	BKP (Bunch-Kaufmann-Parlett)	copy
input/output	Cholesky LDL^T	input/output
allocate/deallocate	Band LDL^T	allocate/deallocate
extract row/column	LU (Gaussian Elimination)	extract row/column
initialize, resize	QR	resize
inner product	Band LU	matrix-vector product
transpose,adjoint	Givens' rotation	sparse Cholesky
multiplication	Householder transformations	sparse LU
norms	diagonal and triangular matrices	sparse BKP
permutation	eigenvalue/vector	iterative methods
linear combination	singular value decomposition	Krylov space methods
complex vectors	matrix polynomials	
addition	matrix exponentials	
	fast Fourier transform	

Table 5: Meschach operations on matrices and vectors.

4 A numerical case study

As a case study, we consider a kanban GSPN of four sub-GSPNs. Each sub-GSPN is modelled by the one shown in Fig. 1. The parameters specifying the internal stochastic behavior for sub-GSPN k are the rates $\boldsymbol{\lambda}_{m_k}$ and $\boldsymbol{\lambda}_{back_k}$ and the probabilities $\mathbf{p}_{ok_k}$ and $\mathbf{p}_{redo_k}$. We assume that these quantities are constant, but they could depend on the local marking (or even on the global marking, in the case of synchronizing transitions) without affecting the feasibility or the complexity of the solution algorithms. A pallet enter sub-GSPN k of the kanban GSPN through transition t_{in_k}, provided there is a kanban ticket in place p_{kanban_k}. Then, the pallet proceeds to the machine, in place p_{m_k}. After being worked by t_{m_k}, a part is checked for quality and it is either transported back to p_{m_k} by t_{back_k} for further rework, or moved out of the machine by t_{out_k}. The numerical values of the parameters for the model are: $\boldsymbol{\lambda}_{m_1} = 1.2$, $\boldsymbol{\lambda}_{m_2} = 1.4$, $\boldsymbol{\lambda}_{m_3} = 1.3$, and $\boldsymbol{\lambda}_{m_4} = 1.1$. Also, $\mathbf{p}_{ok_k} = 0.7$, $\mathbf{p}_{redo_k} = 0.3$, and $\boldsymbol{\lambda}_{back_k} = 0.3$ for $k = 1, 2, 3, 4$. The parameters affecting the synchronizing behavior are the rates $\boldsymbol{\lambda}_{in_k}$ and $\boldsymbol{\lambda}_{out_k}$, specified later. All transitions have single-server semantics. We consider six cases of a kanban GSPN, corresponding to the connections shown in Fig. 2(1)—(6).

(1.) We assign rate $0.4 \cdot 0.3 = 0.12$ to $\{t'_{out_1}, t_{in_2}\}$ and $0.4 \cdot 0.7 = 0.28$ to $\{t''_{out_1}, t_{in_3}\}$. We also assign rate 1.0 to both $\{t_{out_2}, t'_{in_4}\}$ and $\{t_{out_3}, t''_{in_4}\}$.

(2.) In a tandem GSPN, we assign rate 0.4 to $\{t_{out_1}, t_{in_2}\}$, 0.5 to $\{t_{out_2}, t_{in_3}\}$, and 0.6 to $\{t_{out_3}, t_{in_4}\}$.

(3.) In a fork/join GSPN, we assign rate 0.4 to the merged set of transitions

$\{t_{out_1}, t_{in_2}, t_{in_3}\}$ and 0.5 to $\{t_{out_2}, t_{out_3}, t_{in_4}\}$.

(4.) We additionally synchronize and assign rate 1.0 to $\{t_{out_4}, t_{in_1}\}$ of the tandem GSPN. Initially N tokens are set in places $p_{m_k}, k = 1, 2, 3$.

(5.) Unlike the first case, we regard synchronizing transitions as immediate and assign probability 0.3 to $\{t'_{out_1}, t_{in_2}\}$, and 0.7 to $\{t''_{out_1}, t_{in_3}\}$. We also assign probability 0.5 to both $\{t_{out_2}, t'_{in_4}\}$ and $\{t_{out_3}, t''_{in_4}\}$.

(6.) In a second fork/join GSPN, we assign probability 4/9 to $\{t_{out_1}, t_{in_2}, t_{in_3}\}$ and 5/9 to $\{t_{out_2}, t_{out_3}, t_{in_4}\}$.

The input and output rates for the entire kanban GSPN are set in all cases but case 4 by assigning $\boldsymbol{\lambda}_{in_1} = 1.0$ and $\boldsymbol{\lambda}_{out_4} = 0.9$.

In Table 6, we give e_k, the expected overall number of tokens in the steady state in p_{m_k}, p_{back_k}, and p_{out_k}, for each sub-GSPN k of the kanban GSPN and the throughput τ of the first sub-GSPN, as a function of the initial number $N_k = N$ of tokens initially in each place p_{kanban_k}. We observe that the loads of corresponding sub-GSPNs differ for the six configurations, while the flow through the system does not change significantly. In the first four cases, the first sub-GSPN is the most loaded and the fourth sub-GSPN is the least loaded. If we compare case (1) with case (5), making the synchronizing transitions immediate the behavior of the system changed so that the third sub-GSPN becomes the bottleneck. Going from case (3) to case (6) shifts the bottleneck to the second and third sub-GSPN. Naturally, for the fork/join GSPNs of cases (3) and (6), the second and third sub-GSPN are equally loaded.

The size of the state spaces $\mathcal{S}_T$, $\mathcal{S}_V$, and $\mathcal{S}'$, and the overall number of non-zero elements for the sparse storage of the "local" matrices $\mathbf{Q}^k$, $\mathbf{W}^{k,j}$, and $\mathbf{D}^{k,j}$ are given in Table 7. For comparison, we also list the number of nonzero elements in $\mathbf{Q}$, that would be generated with elimination, and in $\tilde{\mathbf{P}}$, that is, with preservation ($\tilde{\mathbf{P}}$ is actually smaller than $\mathbf{P}$, since not all vanishing markings need to be preserved, see [8]). This allows to estimate the memory requirements in addition to two double-precision iteration vectors $\boldsymbol{\pi}^{[m]}$ and $\boldsymbol{\pi}^{[m+1]}$, and the integer vector of the indices of the reachable markings, each of size $|\mathcal{S}|$. We can see how the memory requirements for the local matrices are negligible with respect to the storage of the three vectors mentioned above, and that, instead, the explicit storage of $\mathbf{Q}$ would require a much larger amount of memory. Table 8 shows the computation time "solve" for the entire solution process using the Jacobi method with relaxation parameter equal to 0.9, and the time "explore" to explore the reachability set, in seconds. For higher values of the parameter N, the exact solution becomes unfeasible. Thus, for cases (1) and (2), we use the Cross aggregation method, level 1 [15], resulting in relatively small approximation errors, as shown in Table 9. The computational effort is neglegible, see Table 10. The convergence criterion is set to $||\boldsymbol{\pi}^{[m]} - \boldsymbol{\pi}^{[m+1]}||_\infty < 10^{-6}$. The program was run on a Sony NWS-5000 workstation with 90Mbyte of main memory.

Based on this example, we might be interested in balancing the load of all sub-systems or in performing parametric sensitivity studies. The approximation

N	i	e_1	e_2	e_3	e_4	τ
1	1	0.877	0.337	0.520	0.469	0.174
2		1.753	0.545	0.997	0.994	0.310
3		2.646	0.657	1.406	1.487	0.400
4		3.561	0.717	1.747	1.939	0.458
1	2	0.903	0.638	0.518	0.368	0.137
2		1.803	1.266	1.041	0.797	0.255
3		2.713	1.847	1.538	1.223	0.339
4		3.642	2.401	2.035	1.667	0.394
1	3	0.907	0.671	0.671	0.355	0.132
2		1.810	1.329	1.329	0.765	0.248
3		2.722	1.946	1.946	1.160	0.332
4		3.647	2.520	2.520	1.523	0.393
5		4.588	3.053	3.053	1.876	0.439
1	4	0.828	0.687	0.722	0.761	0.128
2		1.696	1.344	1.428	1.531	0.243
3		2.593	1.963	2.127	2.316	0.327
4		3.515	2.542	2.822	3.119	0.388
5		4.455	3.081	3.523	3.939	0.433
1	5	0.801	0.632	0.674	0.762	0.198
2		1.594	1.280	1.405	1.684	0.333
3		2.413	1.958	2.174	2.632	0.415
4		3.240	2.644	2.962	3.613	0.476
1	6	0.860	0.810	0.810	0.534	0.139
2		1.691	1.671	1.671	1.268	0.263
3		2.543	2.548	2.548	1.974	0.343
4		3.379	3.437	3.437	2.748	0.413
5		3.789	4.454	4.454	3.914	0.529

Table 6: Expected number of tokens in each sub-GSPN and throughput as a function of N for cases i.

methods can give us rough estimates for the dynamic behavior of the system, while some selected exact computations provide us with solutions for a models with up to a few million states in a matter of hours.

5 Conclusion

We presented SNS, the synchronized network solver, and gave an example of its application to the modeling of a kanban system. Thanks to the compositional approach and to the approximation algorithms implemented in SNS, models with very large state spaces can be successfully analyzed. For a copy of the program

N	i	$\lvert\mathcal{S}_T\rvert$	$\lvert\mathcal{S}_V\rvert$	$\lvert\mathcal{S}'\rvert$	local	nonzero($\mathbf{Q}$)	nonzero($\tilde{\mathbf{P}}$)
1	1	256	0	256	22	960	960
2		10,000	0	10,000	88	62,400	62,400
3		160,000	0	160,000	220	1,280,000	1,280,000
4		1,500,625	0	1,500,625	440	13,965,000	13,965,000
1	2	256	0	256	20	944	944
2		10,000	0	10,000	80	60,800	60,800
3		160,000	0	160,000	200	1,240,000	1,240,000
4		1,500,625	0	1,500,625	400	13,475,000	13,475,000
1	3	160	0	256	20	616	616
2		4,600	0	10,000	80	28,120	28,120
3		58,400	0	160,000	200	446,400	446,400
4		454,475	0	1,500,625	400	3,979,850	3,979,850
5		2,546,432	0	9,834,496	700	24,460,016	24,460,016
1	4	108	0	256	20	360	360
2		2,808	0	10,000	80	15,768	15,768
3		34,240	0	160,000	200	247,296	247,296
4		261,100	0	1,500,625	400	2,194,800	2,194,800
5		1,445,904	0	9,834,496	700	13,463,340	13,463,340
1	5	202	52	256	22	800	852
2		5,472	3,336	10,000	88	35,064	38,400
3		61,600	55,104	160,000	220	492,640	547,744
4		420,625	467,000	1,500,625	440	3,833,000	4,300,000
5		2,075,346	2,075,346	9,834,496	770	20,631,240	23,273,940
1	6	152	8	256	20	600	608
2		3,816	697	10,000	80	23,832	24,529
3		41,000	13,656	160,000	200	316,360	330,016
4		268,475	128,000	1,500,625	400	2,343,050	2,471,050
5		1,270,962	769,480	9,834,496	700	12,025,566	12,795,046

Table 7: Memory requirements as a function of N for cases i.

please contact the first author.

Further work on the tool will undoubtedly focus on distributed implementations to exploit the availability of multiple workstations. This is particularly useful for the state space exploration, where efficient distributed state space generation algorithms have been devised [2, 6]. This will easily allow to manage models having one order of magnitude larger state spaces than the ones presented.

Another fundamental feature required in practice will be the use of more sophisticated approximation algorithms, since the exact Markovian solution of realistic models is likely to exceed the amount of memory and processing power we can access.

N	i	"explore"	"solve"	i	"explore"	"solve"
1	1	not req.	0.083	2	not req.	0.100
2		not req.	8.700		not req.	6.000
3		not req.	436.067		not req.	251.717
4		not req.	6,954.650		not req.	3,300.017
1	3	0.017	0.267	4	0.005	0.150
2		0.967	12.550		0.633	7.583
3		16.267	309.533		11.267	198.883
4		190.500	4,736.400		122.500	3,393.650
5		1,759.000	22,215.257		1,477.317	20,830.417
1	5	0.017	0.283	6	0.017	0.183
2		0.783	21.400		0.517	8.600
3		15.250	401.133		5.967	73.833
4		323.483	4,268.317		75.783	719.750
5		3,988.933			707.483	7,111.681

Table 8: Computational requirements as a function of N for cases i.

N	i	e_1	e_2	e_3	e_4	τ
1	1	0.875	0.292	0.494	0.477	0.177
2		1.746	0.471	0.942	1.008	0.316
3		2.637	0.565	1.319	1.487	0.406
4		3.551	0.611	1.622	1.899	0.464
10		9.350	0.660	2.325	3.062	0.565
15		14.334	0.662	2.379	3.200	0.570
1	2	0.903	0.626	0.502	0.372	0.138
2		1.795	1.226	1.014	0.807	0.259
3		2.702	1.785	1.486	1.212	0.344
4		3.623	2.299	1.910	1.575	0.404
10		9.392	4.443	3.480	2.827	0.542
15		14.349	5.297	3.924	3.112	0.563

Table 9: Expected number of tokens in each sub-GSPN and throughput as a function of N for the Cross aggregation, level 1 on cases i.

Acknowledgements

We greatly appreciate the use of the package Meschach [14].

N	i	$\|\mathcal{S}'\|$	local	"solve"
1	1	256	22	0.283
2		10,000	88	0.450
3		160,000	220	0.550
4		1,500,625	440	0.800
10		6,690,585,616	4,840	5.200
15		443,364,212,736	14,960	17.883
1	2	256	20	0.200
2		10,000	80	0.200
3		160,000	200	0.217
4		1,500,625	400	0.433
10		6,690,585,616	4,400	4.783
15		443,364,212,736	13,600	16.633

Table 10: Memory and computational requirements as a function of N for the Cross aggregation, level 1 on cases i.

References

[1] F. Bause and P. Kemper. QPN-Tool for qualitative and quantitative analysis of queueing Petri nets. In G. Haring and K. Kotsis, editors, *Lecture Notes in Computer Science 794 (Proc. 7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation, Vienna, Austria)*, pages 321–334. Springer-Verlag, 1994.

[2] S. Caselli, G. Conte, and P. Marenzoni. Parallel state space exploration for GSPN models. In G. De Michelis and M. Diaz, editors, *Application and Theory of Petri Nets 1995, Lecture Notes in Computer Science 935 (Proc. 16th Int. Conf. on Applications and Theory of Petri Nets, Turin, Italy)*, pages 181–200. Springer-Verlag, June 1995.

[3] G. Chiola. GreatSPN 1.5 Software Architecture. In *Computer Performance Evaluation*, pages 121–136. Elsevier Science Publishers, 1992.

[4] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte. Generalized Stochastic Petri Nets: a definition at the net level and its implications. *IEEE Trans. Softw. Eng.*, 19(2), pages 89–107, Feb. 1993.

[5] G. Ciardo, A. Blakemore, P. F. J. Chimento, J. K. Muppala, and K. S. Trivedi. Automated generation and analysis of Markov reward models using Stochastic Reward Nets. In C. Meyer and R. J. Plemmons, editors, *Linear Algebra, Markov Chains, and Queueing Models*, volume 48 of *IMA Volumes in Mathematics and its Applications*, pages 145–191. Springer-Verlag, 1993.

[6] G. Ciardo, J. Gluckman, and D. Nicol. Distributed state-space generation of discrete-state stochastic models. Submitted.

[7] G. Ciardo, J. K. Muppala, and K. S. Trivedi. On the solution of GSPN reward models. *Perf. Eval.*, 12(4), pages 237–253, 1991.

[8] G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. Submitted.

[9] G. Ciardo, K. S. Trivedi, and J. K. Muppala. SPNP: stochastic Petri net package. In *Proc. 3rd Int. Workshop on Petri Nets and Performance Models (PNPM'89)*, pages 142–151, Kyoto, Japan, Dec. 1989. IEEE Comp. Soc. Press.

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

[11] S. Donatelli. Superposed generalized stochastic Petri nets: definition and efficient solution. In R. Valette, editor, *Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815 (Proc. 15th Int. Conf. on Applications and Theory of Petri Nets, Zaragoza, Spain)*, pages 258–277. Springer-Verlag, June 1994.

[12] W. K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In W. J. Stewart, editor, *Numerical Solution of Markov Chains*, pages 357–371. Marcel Dekker, Inc., New York, NY, 1991.

[13] P. Kemper. Numerical analysis of superposed GSPNs. In *Proc. Int. Workshop on Petri Nets and Performance Models (PNPM'95)*, pages 52–61, Durham, NC, Oct. 1995. IEEE Comp. Soc. Press.

[14] D. E. Stewart and Z. Leyd. Meschach: Matrix Computation in C. In *Proceedings of the Center for Mathematics and Its Applications, Vol. 32.* The Australian National University, 1994.

[15] Y. Takahashi. Aggregate approximation for acyclic queuing networks with communication blocking. In H. G. Perros and T. Altiok, editors, *Queueing Networks with Blocking*, pages 33–47. Elsevier Science Publishers B.V., 1989.

[16] M. Tilgner. An approach to formalize structural decomposition and aggregation for stochastic reward net models. In *Proc. Int. Workshop on Petri Nets and Performance Models (PNPM'95)*, pages 252–261, Durham, NC, Oct. 1995. IEEE Comp. Soc. Press.

Appendix

As an example, we give the input file for the unstructured GSPN shown in Fig. 1.

```
void *sns_parameter(){
  sns_place_bound->me[0][0] = N;        /* place p_m */
  sns_place_bound->me[0][1] = N;        /* place p_c */
  sns_place_bound->me[0][2] = N;        /* place p_back */
  sns_place_bound->me[0][3] = N;        /* place p_out */
  sns_transition_type[0] = iv_get(6);
  sns_transition_type[0]->ive[0]=600;   /* transition t_out */
  sns_transition_type[0]->ive[1]=500;   /* transition t_in */
  sns_transition_type[0]->ive[2]=501;   /* transition t_m */
  sns_transition_type[0]->ive[3]=0;     /* transition t_redo */
  sns_transition_type[0]->ive[4]=1;     /* transition t_ok */
  sns_transition_type[0]->ive[5]=502;   /* transition t_back */
  sns_reward_rate_no->ive[0] = 2;       /* no. of tokens, throughput */
}

void *sns0_enable(unsigned char *type, UCVEC *place, int i, double *rate){
  switch (i) {
  case 1:   /* t_in */
    if((place[1] + place[2] + place[3] + place[4]) < N) timed(1.0); break;
  case 2:   /* t_m */
    if((place[1]) > 0) timed(1.2); break;
  case 3:   /* t_redo */
    if((place[2]) > 0) immediate(0.3); break;
  case 4:   /* t_ok */
    if((place[2]) > 0) immediate(0.7); break;
  case 5:   /* t_back */
    if((place[3]) > 0) timed(0.3); break;
  case 0:   /* t_out */
    if((place[4]) > 0) timed(1.0); break;
  }
}

void *sns0_fire(UCVEC *place, int i){
  switch (i) {
  case 1: place[1] += 1; break;
  case 2: place[1] -= 1; place[2] += 1; break;
  case 3: place[2] -= 1; place[3] += 1; break;
  case 4: place[2] -= 1; place[4] += 1; break;
  case 5: place[3] -= 1; place[1] += 1; break;
  case 0: place[4] -= 1; break;
  }
}

void *sns0_initial(UCVEC *place){}

double sns0_reward(UCVEC *place,int i){
  switch (i) {
  case 0:   /* no. of tokens in system */
    measure(place[1] + place[2] + place[3] + place[4]); break;
  case 1:   /* throughput of transition t_m */
    if(place[1]>0) measure(1.2); else measure(0); break;
  }
}
```