# Verifying Stochastic Well-formed Nets with CSL Model-Checking Tools*

D. Cerotti, D. D'Aprile, S. Donatelli, and J. Sproston
Dipartimento di Informatica, Università di Torino, Torino, Italy
{cerotti,daprile,susi,sproston}@di.unito.it

## Abstract

*Model-checking algorithms for Continuous Stochastic Logic (*CSL*) properties have been introduced to facilitate the verification of stochastic systems against a variety of formally-defined performance indices.*

*In this paper, we consider the application of* CSL *model-checking methods and tools to Stochastic Well-formed Nets (*SWN*), a colored extension of Stochastic Petri Nets (*SPN*). Our approach is to connect an existing tool for the description and performance analysis of* SWN*, called* GREATSPN*, to two model-checking tools for* CSL *properties, namely* PRISM *and* MRMC*. We illustrate our approach using a simple example of resource usage. As a by-product of the implementation of the model translation from* GREATSPN *to* PRISM*, a method for unfolding* SWN *models into* SPN *models has been implemented, as a stand-alone, re-usable component.*

## 1. Introduction

Model checking [8] is a verification method which establishes automatically whether a formal description of a system satisfies a property. Examples of properties which can be verified by model checking are absence of deadlock or reachability of error states. Typically, the system is described in terms of high-level languages such as Petri nets or process algebras, whereas the property to be verified is specified in terms of a temporal logic.

Systems which exhibit non-trivial probabilistic or stochastic behavior are modelled more appropriately using formalisms such as stochastic Petri nets [21] or stochastic process algebra [17, 15], the underlying semantics of which take the form of Markov chains. Model-checking algorithms for continuous-time Markov chains (CTMCs) have been developed, where the property to be verified is described in terms of Continuous Stochastic Logic (CSL) [2, 3]. The logic CSL provides a formal way to describe potentially complex properties reasoning about both the functional behavior and the performance of a stochastic system. It includes a steady-state operator, which can refer to the probabilities of the system being in certain states in equilibrium, and a probabilistic operator, which can refer to the probability with which a certain (possibly timed) property is satisfied, such as "does the system reach an error state within 5 minutes with probability greater than 0.01?" The model-checking tools ETMCC [16], its successor MRMC [19], and PRISM [20, 24], have been used to analyze CSL properties of stochastic systems in application areas such as fault-tolerant systems, manufacturing systems and biological processes.

In this paper, we focus on the use of CSL model-checking tools for the verification of systems described using *Stochastic Well-formed Nets* [7] (SWN). SWN are a colored extension of Generalized Stochastic Petri Nets [1] (GSPN), and have two main advantages over GSPN. Firstly, SWN allow us to define models that are parametric in the "structure" of the system: for example, a GSPN model of a system in which a pool of servers visit a set of stations in a given order is parametric in the number of servers, in the number of clients at each station, but not in the number of stations, whereas in an SWN model a color can be used to distinguish the stations, without the need of replicating station subnets. The second advantage of SWN is related to the solution process: if the color is used to represent a symmetric behavior of the system, this symmetric behavior can be exploited to build a "compact" state space, thus enlarging the size of systems that can be solved. Despite the appealing characteristics of SWN, and the availability of an associated tool (GREATSPN [4, 23] of the University of Torino and the University of Paris 6), SWN modelling is not widespread. One reason for this is the limited support available to validate an SWN model (for example, structural analysis, such as P- and T- invariant computation, is not available for SWN models).

In order to (partially) address this problem, model-checking techniques have been used previously to provide support for the validation of SWN models: in [13],

*Supported in part by the FIRB-Perf project of the Italian Ministry of Research.

an extension of GreatSPN that allows the use of the PROD tool [25] (a reachability graph analyzer defined for predicate-transition nets) to model check an SWN net against a number of temporal logic properties is presented. However, this approach considers only functional properties of the system, such as “can the system reach an error state”, rather than performance indices, and hence is only partially adequate when considering analysis of stochastic formalisms such as SWN. In this paper, we extend the work of [13], and our work on CSL model checking of stochastic Petri nets [12], by considering CSL model checking of SWN. Following our work in [12], we have not built a new CSL model checker, but we have provided a CSL model-checking facility for SWN models in GreatSPN by linking GreatSPN to MRMC and Prism. These links take the form of programs which translate SWN from the format used in GreatSPN to the (different) input languages of Prism and MRMC.

We have preferred to consider two “stand-alone” model checkers because they have been built and are maintained by researchers in the specific field of stochastic modelling and verification, and are constantly updated to reflect recent research results. Moreover, although neither tool is based on Petri nets, we considered that they would not be difficult to connect with the GreatSPN modules for SWN, given that there is already some reported experience in interfacing these tools with other tools (ETMCC, the precursor of MRMC, has been interfaced with the Petri net tool DaNAMiCS [11] and the process algebra tool TIPP [14], while Prism has been interfaced with the PEPA process algebra [17]). We observe that this choice is based on re-use, and it has the significant advantage of being able to profit of all future developments of MRMC and Prism. However, on the other hand, there is the drawback of not being able to exploit all the peculiarities and properties of nets in the model checking algorithm, as it would have been the case if an ad-hoc solution had been devised.

Note that, as a by-product of the translation from GreatSPN models to Prism models, we have implemented a method for unfolding SWN nets to GSPN nets within GreatSPN, which can be used in other contexts aside from CSL model checking.

Another contribution of this paper is the consideration of the way in which meaningful CSL properties of SWN may be specified. In particular, we concentrate on the way in which “atomic propositions”, which are used in model checking to identify portions of the state space of interest (for example, error states or goal states), can be described. Note that, for SWN, this task is not always straightforward, as the model-checking tool user may need to refer to complex expressions including net places, transitions, and colored tokens, to identify the required part of the state space.

In this paper we concentrate on SWN and on the GreatSPN tool, but there are other colored extensions of stochastic Petri nets with associated tools. The tool APNN has a built-in CSL model checker [5] which works on the colored GSPN class defined in APNN; however, we were unfortunately not able to use APNN extensively due to lack of documentation. A notion of colors based on replication is present in SAN nets in UltraSAN [10] and Möbius [9]. There are plans to add a CSL model checker to the latter.

The paper is organized as follows: Section 2 recalls the definitions of CSL, GSPN, and SWN, and presents our running example. Section 3 discusses CSL model-checking of SWN, in particular with respect to the choice of the set of atomic propositions. Section 4 describes linking GreatSPN to Prism and to MRMC with the help of the running example. Section 5 concludes the paper.

## 2. Background

In this section, we summarize the temporal logic CSL and the formalisms GSPN and SWN, together with an associated running example of a central server system.

CSL **model checking.** Let $AP$ be a set of atomic propositions, and let $\mathbb{R}_{\geq 0}$ be the set of non-negative real numbers. The syntax of CSL [2, 3] is defined as follows:

$$\Phi ::= \mathsf{a} \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie\lambda}(X^I\Phi) \mid \mathcal{P}_{\bowtie\lambda}(\Phi U^I \Phi) \mid \mathcal{S}_{\bowtie\lambda}(\Phi)$$

where $\mathsf{a} \in AP$ is an atomic proposition, $I \subseteq \mathbb{R}_{\geq 0}$ is a nonempty interval, interpreted as a time interval, $\bowtie \in \{<, \leq, \geq, >\}$ is a comparison operator, and $\lambda \in [0,1]$ is interpreted as a probability. Formulae of CSL are evaluated over continuous-time Markov chains (CTMCs), each state of which is labelled with the subset of atomic propositions $AP$ that hold true in that state.

The logic CSL can be divided into *state formulae*, which are true or false in a state, and *path formulae*, which are interpreted as being true or false for a path of the system (where a path is a sequence of transitions which represents a computation of the CTMC). The interpretation of the state formulae is as follows: a state $s$ satisfies the atomic proposition $\mathsf{a}$ if $s$ is labelled with $\mathsf{a}$, the operators $\wedge$ and $\neg$ have the usual interpretation, while $\mathcal{S}_{\bowtie\lambda}(\Phi)$ is true in $s$ if, assuming $s$ as initial state, the sum of the steady state probabilities of the states that satisfy $\Phi$ is $\bowtie \lambda$. For a path formula $\varphi \in \{X^I\Phi, \Phi_1 U^I \Phi_2\}$, the state formula $\mathcal{P}_{\bowtie\lambda}(\varphi)$ is true in $s$ if the probability of the paths leaving $s$ which satisfy $\varphi$ is $\bowtie \lambda$. We say that $\mathcal{P}_{\bowtie\lambda}(\varphi)$ is a *probabilistic formula*, whereas $\mathcal{S}_{\bowtie\lambda}(\Phi)$ is a *steady-state formula*.

The interpretation of the path formulae $X^I\Phi$ and $\Phi_1 U^I \Phi_2$ is as follows. The Next formula $X^I\Phi$ is true for a path if the state reached after the first transition along the path satisfies $\Phi$, and the duration of this transition lies in the interval $I$. The Until formula $\Phi_1 U^I \Phi_2$ is true along a path if $\Phi_2$ is true at some state along the path, possibly also the initial state of the path under consideration, the time elapsed before reaching this state lies in $I$, and $\Phi_1$ is true along the path until that state. The formal semantics of CSL can be found in [3]. The usual abbreviations for propositional and temporal logic will be used throughout the paper; for example, $F^I\Phi \equiv \texttt{true}U^I\Phi$, $\mathcal{P}_{\geq\lambda}(G^I\Phi) \equiv \mathcal{P}_{<1-\lambda}(F^I\neg\Phi)$.

*Model checking* a CTMC against a CSL formula $\Phi$ consists of computing the set of states $Sat(\Phi)$ such that $s \in Sat(\Phi)$ if and only if $\Phi$ is true in $s$. The set $Sat(\Phi)$ is constructed by computing recursively the sets of states satisfying the sub-formulas of $\Phi$. When probability bounds are present in the CSL formula ($\mathcal{P}_{\bowtie\lambda}$ or $\mathcal{S}_{\bowtie\lambda}$), the model-checking algorithm requires the computation of transient or steady-state probabilities of the original CTMC as well as, depending on the formula, a number of additional CTMCs built through manipulation of the original one: see [3] for more details.

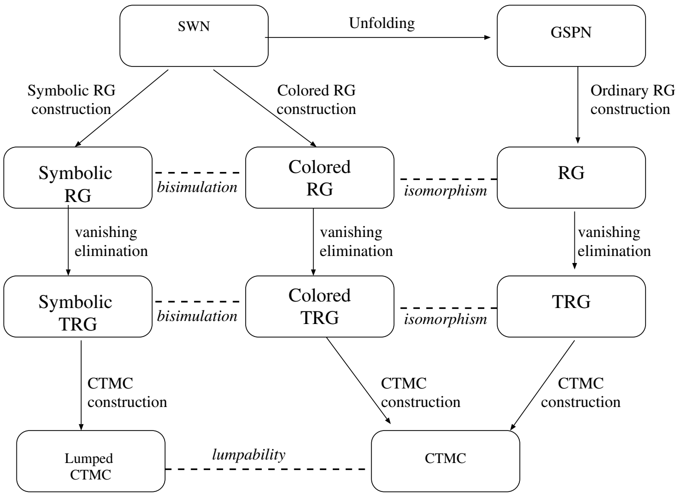


**Figure 1. Solution approaches for SWN**

**GSPN and SWN.** A Stochastic Petri Net [21] (SPN) is a tuple $S = (P,T,I,O,W,m_0)$, where $P$ is the set of places, $T$ is the set of transitions, $I,O : P \rightarrow T \rightarrow 2^{\mathbb{N}}$ define the input and output arcs with associated multiplicity, $W : T \rightarrow \mathbb{R}_{\geq 0}$ defines the rate of the exponential distributions associated to transitions, and $m_0 : P \rightarrow \mathbb{N}$ describes the initial marking. It is well-known that the stochastic process underlying an SPN is a CTMC which is isomorphic to the reachability graph (RG) of the SPN built disregarding the timing aspects. Generalized SPN (GSPN) [1] are an extension of SPN in which the set $T$ is split into immediate and stochastic transitions. Immediate transitions fire in zero time, with priority over timed ones, and conflicts are resolved probabilistically. As a consequence the RG includes also some *vanishing* states in which zero time elapses. We distinguish the RG from the Tangible RG (TRG) in which only tangible (non-vanishing) states are preserved. The stochastic process associated to a GSPN is a semi-Markov process from which a CTMC can be produced considering only the tangible states. Figure 1, in its right-most path, describes the solution process of a GSPN, in particular that followed by GREATSPN: the Reachability Graph is built first, and each arc is labelled with the name of the transition that causes that change of state; then the vanishing states are eliminated, thus creating the Tangible Reachability Graph, in which each arc is labelled with either the name of a timed transition or of a timed followed by a sequence of immediates. The CTMC is then built using the TRG and the rates and probabilities associated to the transitions of the net.

Stochastic Well-formed Nets [7] are a colored extension of GSPN. The peculiarity of SWN is that the *color domain* of places is built as the Cartesian product of a limited number of *basic color classes*, and that functions on arcs are

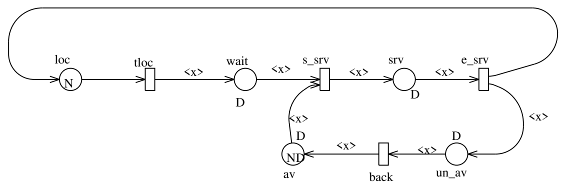


**Figure 2. The SWN model of the running example**

expressed as linear combination of a few basic functions (projection to select an element, sum function to select the whole color class). We do not provide a formal definition of SWN in this context, due to space reasons, but we recall the main points by using an example.

The net of Figure 2 shows an SWN model of a simple computing system: there is a central server, modelled by place loc and transition tloc, at which there are initially $N$ jobs.

Once the computation local to the central server terminates, a job chooses one of the peripherical devices, and a request for that device is put in place wait. Different devices are modelled as different colors of the color class $D$. Hence $D = \{d_1, d_2, \ldots, d_K\}$ indicates that there are $K$ different devices. A device $d$ can be in one of the following states: available (one token of color $d$ in place av), being used by a job (one token of color $d$ in place srv), and unavailable (one token of color $d$ in place un_av). The choice of a device by a job is modelled by the single server transition tloc that, due to the free variable $x$ associated to the arc out of tloc, puts in place wait a token of color $d$, randomly chosen with equal probability amongst all $K$ colors of the color class $D$.

A function $\langle x \rangle$, such as the one on the arc from wait to s_srv, is called a projection function and evaluates to one of the elements of $D$: if in a marking there is a token in place loc then there are $K$ *instances* of transition tloc enabled, one per each possible color assignment to $x$.

Transition s_srv can fire for an assignment to variable $x$ of color $d$ only if there is at least one token of color $d$ in place wait and at least one token of color $d$ in place av. The firing of s_srv for $x = d$ puts a token of the same color in srv. Observe that, once the device has provided the required service, transition e_srv fires, and puts an uncolored token (the job) back to the server place loc, and a colored $d$ token (the device) into place un_av. Place un_av models some reset time of the device (a period in which the device is unavailable). We assume that all devices have the same speed and that each device can work on a single job at a time (the service policy of transition s_srv and e_srv is "single server per color").

The other basic function of SWN is the constant function $S$: if we change the function on the arc out of tloc to $\langle S \rangle$, then each firing of tloc adds $K$ tokens to place wait, one token per color in $D$, thus representing a fork of a job into $K$ threads, one per device. If the function $\langle S \rangle$ is associated also to the arc from srv to e_srv then the transition can fire only if there is in srv at least one token per color: therefore it expresses a synchronization in the elaboration at the $n$ peripherical devices.

From an SWN an equivalent GSPN can be generated, which has the same structure, replicated as many times as there are colors (in the example above there are $K$ replicas of all colored places). The process of generating a GSPN equivalent to a given SWN is called *unfolding* [18]: each place is duplicated as many times as there are distinguished colored tokens assigned to that place, transitions are duplicated as many times as there are transition instances, and arcs are computed according to the functions associated to the arcs of the SWN. Figure 3 shows the GSPN model obtained through unfolding of the SWN model of Figure 2.

Figure 1, in its central path, describes the *colored* solution process of an SWN: the Colored Reachability Graph (CRG) is built first, followed by a colored TRG, from which the CTMC is computed. While a state in a GSPN is an assignment of tokens to places, in SWNs, as in all colored nets, a state is an assignment of colored tokens to places. Listed in the last three lines of Table 1 are three colored markings: state $C_{1a}$ corresponds to a situation in which there is a job waiting for device $d1$, while a job is using device $d2$, and another one is using $d3$.

Figure 1 depicts also the relationship between an SWN and its unfolded GSPN: the CRG and the RG are isomorphic, and the same Markov chain is produced.

The real advantage of colored nets in general, and of

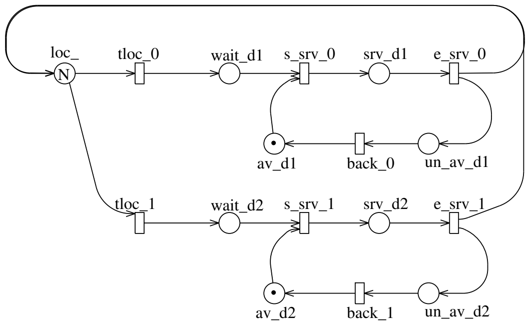


**Figure 3. Unfolding of the SWN example model**

| State | State Description | $Z$ Cardinality |
|---|---|---|
| $S_1$ | $\mathrm{M}(\text{wait}) = 1 \cdot Z_{D,1}$, $\mathrm{M}(\text{srv}) = 1 \cdot Z_{D,2}$ | $\lvert Z_{D,1}\rvert = 1$, $\lvert Z_{D,2}\rvert = 2$ |
| $S_2$ | $\mathrm{M}(\text{wait}) = 1 \cdot Z_{D,1}$, $\mathrm{M}(\text{srv}) = 1 \cdot Z_{D,1} + 1 \cdot Z_{D,2}$ | $\lvert Z_{D,1}\rvert = 1$, $\lvert Z_{D,2}\rvert = 1$ |
| $C_{1a}$ | $\mathrm{M}(\text{wait}) = 1 \cdot d_1$, $\mathrm{M}(\text{srv}) = 1 \cdot d_2 + 1 \cdot d_3$ | n.a. |
| $C_{1b}$ | $\mathrm{M}(\text{wait}) = 1 \cdot d_2$, $\mathrm{M}(\text{srv}) = 1 \cdot d_1 + 1 \cdot d_3$ | n.a. |
| $C_{1c}$ | $\mathrm{M}(\text{wait}) = 1 \cdot d_3$, $\mathrm{M}(\text{srv}) = 1 \cdot d_1 + 1 \cdot d_2$ | n.a. |

**Table 1. Symbolic and colored states for SWN**

SWN in particular, is the ability to exploit the symmetries described by the color. This exploitation is fully automatic for SWN. Figure 1, in its left-most path, describes the *symbolic* solution process of an SWN: the Symbolic Reachability Graph (SRG) is built first, followed by a symbolic TRG, from which the CTMC is computed. We do not enter in the details of the SRG construction [7] here: however, we recall that a state of the SRG (*symbolic marking*) is an equivalence class of colored markings. The equivalence classes are represented in terms of a partition of a color class into dynamic subclasses $Z_{ColorClass,index}$, characterized simply by the subclass cardinality.

Examples of symbolic markings for the central server model are given in the first two lines of Table 1. State $S_1$ is the equivalence class of all markings that have a job waiting for a device and two other jobs in service using the two other devices: indeed state $S_1$ corresponds to the three colored markings $C_{1a}$, $C_{1b}$ and $C_{1c}$. State $S_2$ corresponds to all colored markings that have a job waiting for a device that is already in use by another job (the same dynamic subclass $Z_{D,1}$ is associated to places wait and srv), and a different device (one of the remaining two) in use by another job.

An interesting feature of SWN is that the equivalence classes are directly computed from the initial marking of the

SWN, thanks to a definition of *symbolic transition firing*: a firing in which dynamic subclasses, instead of specific colors, are assigned to variables. Therefore in the SRG, out of state $S_1$ there is an arc in which $x$ is associated to $Z_{D,1}$: the single arc represents the three arcs corresponding to the possible assignment to $x$ of a color in $D$.

The SRG is a bisimulation of the CRG (although the bisimulation relation is never computed), while the CTMC built from the SRG is a lumped CTMC with respect to the CTMC generated from the CRG, as summarized by Figure 1. Observe that, for each symbolic marking, it is possible to build the list of corresponding colored markings.

Note that both of the CSL model-checking tools which we consider in this paper, PRISM and MRMC, do not allow actions to happen in zero-time and with priority over exponential activities: therefore we limit our scope to nets without immediate transitions only.

## 3. CSL model checking of GSPN and SWN

CSL model checking of GSPN/SWN requires the following set of ingredients: a GSPN/SWN model, a set of atomic propositions expressed in terms of net elements, a CSL model checker, and a way to interpret the results of model checking.

Let us consider the set of atomic propositions $AP$. With regard to place-related atomic propositions, which we henceforth call *place propositions*, the expressions of interest are of the form:

**(Type M):** $\sum_{\mathsf{p} \in P} w_p \cdot \mathrm{M}(\mathsf{p}) \sim K$

**(Type Mcol):** $\sum_{\mathsf{p} \in P, c \in CD(\mathsf{p})} w_{p,c} \cdot \mathrm{M}(\mathsf{p})[c] \sim K$

where $w_p$ and $w_{p,c}$ are integers, $CD(\mathsf{p})$ is the color domain of place p (set of possible colors of tokens in p), and $\sim \in \{\leq, =, \geq\}$: observe that, since $w_p$ and $w_{p,c}$ can be zero, the atomic propositions can refer also to an arbitrary set of places or to an arbitrary set of colors and places.

Examples of place propositions, using the net of Figure 2, are: $\mathrm{M}(\mathsf{loc}) = 1$ (there is one job at the central server), $\mathrm{M}(\mathsf{wait})[d2] \geq 2$ (there are at least two jobs waiting for the services of device $d2$).

With regard to transition-related atomic propositions, which we henceforth call *transition propositions*, the expressions of interest are of the form **(Type T)** the transition t is enabled, or **(Type Tcol)** the transition t is enabled for a given assignment to the variables of t. Examples of transition propositions, using the net of Figure 2, are: transition s_srv is enabled (a device has been assigned to a job), transition s_srv is enabled for variable $x$ instantiated to color $d1$ (device $d1$ has been assigned to a job).

For what concerns the evaluation of atomic propositions we can again distinguish marking propositions from transition propositions. Propositions of type M can be defined for GSPN and SWN, and can be computed trivially using the RG for GSPN, and with some effort using the CRG or SRG for SWN (indeed a sum over all colors is necessary for the CRG case, while the cardinality of the dynamic subclasses is used for the SRG case). Propositions of type Mcol are defined instead only for SWN, and can be computed only for the CRG, since this is a type of proposition that may hold in some of the states corresponding to a symbolic marking, but not in all of them. In the running example, the proposition $\mathrm{M}(\mathsf{srv})[d_1] \geq 1$, which is of type Mcol, is valid only for two of the three states represented by the symbolic marking $S_1$ (more precisely, states $C_{1b}$ and $C_{1c}$). In this case, the proposition distinguishes more than the net, and requires a modified SRG construction, as in [6]. We do not consider this possibility in this paper.

Propositions of type T can be defined for GSPN and SWN, and can be computed trivially on the three types of reachability graphs, by simply checking the name of the transitions associated to the arcs out of a state. Propositions of type Tcol are defined instead only for SWN, and can be computed easily from the CRG, where this information is explicitly present, while for the SRG the considerations are similar to the Mcol case; again using the example, if we consider the Tcol proposition "s_srv enabled for an assignment to $x$ of $d_1$," then not all colored states represented by a symbolic marking enable transition s_srv for the required variable assignment.

There is another type of proposition of interest when dealing with SWN, which we call *symbolic*: these are proposition that consider the color, but not the specific value of the color. Examples of such proposition on the SWN of Fig. 2 are: there are at least two tokens of the same color in place wait; transition s_srv is enabled for a value of variable $x$ equal to the value of variable $y$ (assuming a modified example in which the function on the arc from av to s_srv is $y$). Observe that the first (second) atomic proposition can be computed as the logical disjunction of atomic propositions of type Mcol (Tcol respectively).

Instead of defining two new types of properties that may be cumbersome to compute, we prefer to consider the use of "observation transitions": the SWN is modified so as to include a new transition for each such property, and the property is associated to a colored or symbolic marking only if the transition is enabled in that marking. In the first case we can add to the net a transition having an arc to and from wait, with the function $2\langle x \rangle$, and in the second case we have to split transition s_srv in two transitions with the same input and output function, but with a guard to distinguish whether $x = y$ or not. Observe that Tcol can be rephrased in terms of enabling of an observation transition.

Observation transitions have been introduced by the authors of SPOT, a model checker for colored (non-stochastic) nets [26], and are described in detail in [27]: "observation" transitions are there defined as transitions with a modified semantics (they are enabled but they never fire), a change in the semantics that is not necessary in our case, if we exclude the use of the Next operator of CSL, because an exponential transition which does not change the marking does not alter the (infinitesimal generator of the) CTMC.

**Running example: properties of interest.** The verification of our running example can range from classical Petri net properties like liveness of transitions and marking invariants, to probabilistic properties that ensure that the service is provided according to certain quality criteria. The liveness of a transition t can be restated as the CSL formula:
$(\Psi_1): \mathcal{S}_{\geq 1.0}(\mathcal{P}_{\geq 1.0}(F^{[0,\infty)}\mathsf{t}(\text{ is enabled}) \geq 1))$
Less trivial is the check of marking invariants. The fact that, in any state, there is only one device per type, can be checked as the logical conjunction over all $d \in D$ of following property:
$(\Psi_2): \mathcal{S}_{\geq 1.0}(\mathrm{M}(\mathsf{av})[d] + \mathrm{M}(\mathsf{srv})[d] + \mathrm{M}(\mathsf{un\_av})[d] = 1)$
Note that $\mathrm{M}(\mathsf{av})[d] + \mathrm{M}(\mathsf{srv})[d] + \mathrm{M}(\mathsf{un\_av})[d] = 1$ is an atomic proposition of Mcol type and it is not simple to define the property in a symbolic manner using an observation transition.

If we are interested instead in proving that each device $d$ will be available upon request, we can verify that the following formula is satisfied in all states:
$(\Psi_3): \mathrm{M}(\mathsf{wait})[d] \geq 1 \Rightarrow \mathcal{P}_{\geq 1.0}(F^{[0,\infty)}\mathrm{M}(\mathsf{srv})[d] \geq 1)$

For what concerns probabilistic aspects, assume we are interested in identifying states that are "hot spots", in which the number of jobs waiting for a device exceeds a certain amount. Given a constant $hs$, the atomic proposition HS is valid in states in which the number of tokens in place wait exceeds $hs$, HS[d] is valid in those states in which the number of tokens of color $d$ in place wait exceeds $hs$, while HSx is valid in those states in which there are at least $hs$ jobs waiting for the same device (independently of the device identity). The first two propositions belong to the M and Mcol types, while the third requires the introduction of an observation transition t_HS having an arc to and from wait, with the function $hs * \langle x \rangle$. In the following properties, hot_spot can be either HS, HS[d], or HSx.
$(\Phi_1): \mathcal{S}_{>0.7}(\mathsf{hot\_spot})$: in steady state the sum of the probabilities of states that are hot spots is greater than 0.7. The running example has a CTMC with a single strongly connected component, hence $\Phi_1$ is either true in all states or false in all states.
$(\Phi_2): \mathcal{S}_{\leq 0.2}(\mathcal{P}_{\geq 0.9}(F^{[0,5]}\mathsf{hot\_spot}))$: this property is true for those states in which the probability of being, in equilibrium, in "bad" states which can reach a hot spot within 5 time units with probability 0.9 or greater, is at most 0.2.
$(\Phi_3): \mathcal{P}_{\geq 0.9}(F^{[0,5]}(\mathsf{hot\_spot} \wedge \mathcal{P}_{\geq 0.7}(F^{[0,3]}\neg\mathsf{hot\_spot}))$: this property is true for those states in which the probability of reaching "good hot spot" states within 5 time units is at least 0.9, where "good hot spot" states are hot spot states in which, with probability 0.7 or greater, the system exits from hot spot states within 3 time units.

In the following sections, we illustrate two different approaches to SWN model checking using existing tools: the first is the interface with PRISM, and is realized at the net level. The second is the interface with MRMC, and is realized at the CTMC level. The two approaches will be illustrated on our central server example.

## 4. Linking GREATSPN to PRISM and MRMC

### 4.1 From GREATSPN to PRISM

PRISM [24] is a probabilistic model-checking tool of the University of Birmingham. The PRISM input language is a state-based language in which a state of a system is described in terms of a valuation of a number of bounded variables declared by the user. The variables may be organized into a series of interacting modules. The dynamics of the system is represented by a set of guarded commands which define sets of state-to-state transitions of the CTMC model. Each command is of the form GUARD $\longrightarrow$ RATE : UPDATE, where GUARD specifies a logical condition on the system variables describing the states in which the command is enabled, RATE is the rate of the command, and UPDATE is a set of assignments that specify the new values of the variables in terms of old ones (where a prime is used to distinguish the new value from the old one), and thus describes the target states of the CTMC transitions defined by the command.

The generation of a PRISM model corresponding to an SPN has been presented in [12]: a single PRISM module is created with as many variables as there are places, and as many commands as there are transitions, where GUARD encodes the transition enabling conditions, and UPDATE encodes the state modification caused by the firing. The set $AP$ of atomic propositions is implicitly defined in PRISM models, as the user is allowed to include in a CSL formula any logical condition on the values of the variables. In the implemented translation place names are mapped one-to-one to variable names, and therefore any logical expression on place markings is allowed and is realized trivially (there is no need to translate type M and Mcol atomic propositions, whereas T and Tcol propositions have to be restated in terms of markings).

Based on our experience with the translation of SPN models we examined two possible ways of connecting GREATSPN to PRISM for SWN: producing directly a PRISM module that is equivalent to the SWN, meaning that the CRG of the SWN and the state space of the PRISM module are isomorphic, and that the same CTMC (up to state numbering) is produced; or *unfolding* the SWN into an

SPN, followed by the translation of the SPN into a PRISM module using the already-existing translation for SPN.

Let us consider the feasibility of the first option. Because variables in PRISM are simple, unstructured variables, it is not possible to associate a single variable to an SWN place; moreover, since transitions can fire for different assignment of colors to transition variables, it is not possible to associate a single PRISM command to an SWN transition. Furthermore, the enabling conditions are much more complicated to write. Basically, to translate an SWN into a PRISM module means that we have to address the same problems as defining an unfolding of the SWN into an SPN.

We have therefore taken the second option: the SWN is unfolded into an equivalent SPN, and then the translator to PRISM is applied. Unfortunately, although unfolding algorithms have been defined for Colored Nets (for example in [18]), there is no such detailed definition for SWN, and there is no implementation available for the complete SWN class. The algorithm that we implemented works as follows:

- for each colored place $p$, and for each distinct tuple of colors in the color domain of $p$, a neutral place *p_colortuple* is generated;
- for each colored transition $t$, and for each possible assignment $\gamma$ of colors to the variables in the input and output colored functions of $t$, a neutral transition $t\_\gamma$ is created, if it does not violate the predicate associated to $t$;
- for each neutral transition $t\_\gamma$, and for each colored place $p$ (input or output place of $t$), the colored function associated with the arc from $p$ to $t$ (or viceversa) is evaluated on the variable assignment $\gamma$; the result of the evaluation is a multiset on the set of neutral places generated from $p$, and which can be used to assign a multiplicity to the arcs between $t\_\gamma$ and the neutral places.

As an example of the application of the algorithm, we consider as input the net of Figure 2 with only two devices: the SPN of Figure 3 is the algorithm's output. The uncolored place loc is translated in the uncolored place loc_, whereas the colored place wait of color domain $D = \{d_1, d_2\}$ is translated in two uncolored places wait_d1 and wait_d2.

The color domain $cd$ of transitions in SWN is defined by a pair $\langle transition\_parameters\_type, guard \rangle$. For example, for transition tloc we have $cd(\mathsf{tloc}) = \langle \langle x \rangle \in D, true \rangle$. The set of possible assignments $\Gamma$ for tloc is $\Gamma(\mathsf{tloc}) = \{x \leftarrow d_1, x \leftarrow d_2\}$. Transition tloc is therefore replaced by two new transitions: tloc_0, for binding of variable $x$ with $d\_1$, and tloc_1 for binding of $x$ with $d\_2$. The resulting output arcs connect transition tloc_0 with uncolored place wait_d1, for assignment $\gamma_1$, and transition tloc_1 with uncolored place wait_d2, for assignment $\gamma_2$; the multiplicity of each arc is 1.

Some care is necessary when translating arc expressions such as $[x \neq z]\langle x + !y, z\rangle + \langle w, Sc\rangle$ where the multiset returned by a tuple of basic functions is obtained by Cartesian product composition of the multisets returned by the tuple elements and where there are predicates associated to single components of the arc expression.

Unfolding in SWN is non-trivial to implement due to the structured form of the color domains, the complexity of the functions (which may have a variable number of terms), and the use of predicates associated to transitions and arcs. The current implementation included in the GREAT2PRISM translator treats the full class of SWN, except for server semantics different from "single server per color", and for the use of dynamic subclasses to define the initial marking.

Although efficiency was not the main objective of the unfolding, the tool was able to translate an SWN net with composite color domain of five classes, producing a SPN of about 100 places and 3500 transitions in a matter of minutes. For the examples in this paper the time required to produce the unfolding was negligible. The resulting SPN model is then translated into a PRISM module using the GREAT2PRISM translator [12]. As for the SPN case, the CSL formulas are expressed using variable names, which, as explained before, encode the place names and the color.

**Running example.** We will now show the results of the CSL model-checking of the properties defined for the running example. We considered a variable number of jobs ($N$), and a variable number of devices ($|D|$ is either 2 or 3).

First the net has been unfolded and translated into a PRISM module using the program GREAT2PRISM. Following is a fragment of PRISM code obtained for the case in which $N$ is equal to 4. Note that the colored place un_av is translated into two PRISM variables, `un_av_d1` and `un_av_d2`, one for each device. Instead, the neutral place loc corresponds to the single PRISM variable `loc_`.

```
const int N = 4;
module M
un_av_d2 : [0..1];
loc_ : [0..4] init 4;
un_av_d1 : [0..1];
wait_d2 : [0..4];
av_d2 : [0..1] init 1;
srv_d2 : [0..1];
wait_d1 : [0..4];
av_d1 : [0..1] init 1;
srv_d1 : [0..1];

[tloc_0] (loc_ > 0) & (wait_d1 < N)
    -> 1.000000 : (wait_d1' = wait_d1 +1) & (loc_' = loc_ -1);

[tloc_1] (loc_ > 0) & (wait_d2 < N)
    -> 1.000000 : (wait_d2' = wait_d2 +1) & (loc_' = loc_ -1);

[back_0] (un_av_d1 > 0) & (av_d1 < 1)
    -> 10.000000 : (av_d1' = av_d1 +1) & (un_av_d1' = un_av_d1 -1);

[back_1] (un_av_d2 > 0) & (av_d2 < 1)
    -> 10.000000 : (av_d2' = av_d2 +1) & (un_av_d2' = un_av_d2 -1);
```

Observe that an Mcol atomic proposition, such as $\mathrm{M}(p)[d] > k$, is translated as $\mathtt{p\_d} > k$, where p_d is the PRISM variable representing the place p with color $d$. Given an appropriate choice for the constant $hs$ (we chose $hs = \lfloor \frac{N}{2} \rfloor$), the atomic propositions HS, HS[d] and HSx can then be defined in terms of PRISM variables and used as input to the tool.

Table 2 shows the results for the various instances of the model (with different values of $N$, different numbers of devices in the set $D$, and different choices for the atomic proposition hot_spot) for the CSL formulae $\Phi_1$, $\Phi_2$ and $\Phi_3$. The probabilities illustrated in the table correspond to those computed for the outermost probabilistic or steady-state operators in the formulae, and were computed for the initial state of the system, as in Figure 2. The cases of $\Phi_1$ and $\Phi_2$ for HS[d1] are combined, as they result in the same probability for all values of $|D|$ and $N$. The results were obtained using the backwards Gauss-Seidel method, with a maximum number of iterations equal to 100000, and with an error of $10^{-9}$ (these are the default options of MRMC, and were adopted in order to compare our results with the two tools). We note that all of the other properties listed in Section 3 ($\Psi_1$, $\Psi_2$ and $\Psi_3$) are satisfied in all states of the system.

### 4.2 From GREATSPN to MRMC

MRMC [19] is a probabilistic model-checking tool for Markov chains of the Universities of Twente and Aachen. When interfacing GREATSPN to MRMC, we consider verification of the CTMCs corresponding to the CRG or the SRG. To link GREATSPN to MRMC it is necessary to provide the two MRMC input files: a .tra file that contains an ASCII description of the CTMC rate matrix, and a .lab file that lists all possible atomic propositions and associates to each state the atomic propositions valid in that state. The link for SWN model checking has been realized as an upgraded version of our previous tool GREATSPN2ETMCC [12], which linked GREATSPN to ETMCC, and which had the limitation of working only for SPN and of being able to express only atomic propositions of type M. The resulting tool, named GREATSPN2MRMC, consists of two main modules, GMC2MRMC and APGENERATOR.

The operational flow to model check an SWN with GREATSPN2MRMC is the following. *Firstly,* GREATSPN is used to define a SWN. *Secondly,* the user creates a .ap text file that contains the atomic propositions of interest of the M, Mcol, T, Tcol types. *Thirdly,* the description of the net (.net and .def-file) is passed to the GMC2MRMC module, which uses offline solvers[1] for the generation of the net's CRG or SRG and associated CTMC. Two output files are generated: the .tra-file, in the MRMC syntax, and a .xlab-file, which is the base for the production of the .lab file required by MRMC. The .xlab-file contains triples of the form ($CTMC-state-id$, $corresponding-net-marking$, $enabled-transitions$) if the CRG is considered, or pairs ($CTMC-state-id$, $enabled-transitions$) if the SRG is used. *Finally,* the .xlab-file and the .ap-file are processed by the APGENERATOR module; if the SRG is used, only M and T propositions present in the .ap-file are evaluated and the labelled CTMC is produced; if CRG is used, also Mcol and Tcol propositions are considered.

[1]These are stand-alone SWN solvers that are not yet part of the current GREATSPN distribution, and for which we would like to thank Marco Beccuti from the University of Piemonte Orientale.

**Running example.** We now discuss the approaches to the specification of atomic propositions in term of SWN elements for the cases of CRG and SRG. To do this, we describe some of the steps involved in the verification of the running example (with $N = 8$ and $|D| = 2$) against property $\Phi_1$, using either the CRG or the SRG option.

*CRG approach.* If we pass the .net and .def-file to the GMC2MRMC module, then we obtain the following MRMC-ready-to-use .tra-file, consisting of a list of transitions, each described in terms of a source state, target state and rate, respectively:

```
STATES 352
TRANSITIONS 1206
1 2 1.000000
1 3 1.000000
2 4 10.000000
...
```

GMC2MRMC also produces an .xlab file, which encodes the marking of each state and the transition enabled in the state:

```
1 av(1<d2>1<d1>) loc(8) tloc
2 av(1<d2>1<d1>)loc(7)wait(1<d1>) s_srv_1 ...
...
```

With the latter file, we are able to derive the hot_spot definitions. To achieve this, we create an .ap-file (the content of which is shown in Table 3) and call the APGENERATOR module, which scans the .xlab-file and the .ap file, and produces the final .lab file for MRMC. The .lab file, which associates atomic propositions to each state, takes the following form:

```
#DECLARATION
t_HS
#END
...
25 wait>=4 wait_d1>=4
...
34 wait>=4 wait_d2>=4
...
```

*SRG approach.* For the SRG case, only HS and HSx will be considered as hot_spot (since HS[d] cannot be checked against the SRG, as explained in Section 3). If we pass the

| Dimension | | | HS | | | HS[d1] | | HSx | | |
|---|---|---|---|---|---|---|---|---|---|---|
| $\|D\|$ | $N$ | $\|States\|$ | $\Phi_1$ | $\Phi_2$ | $\Phi_3$ | $\Phi_1, \Phi_2$ | $\Phi_3$ | $\Phi_1$ | $\Phi_2$ | $\Phi_3$ |
| 2 | 4 | 106 | 0.6207 | 1.0 | 0.9461 | 0.2559 | 0.5121 | 0.5115 | 0.8336 | 0.8580 |
| | 16 | 1276 | 0.9064 | 0.9289 | 0.0695 | 0.3583 | 0.0053 | 0.7166 | 0.7494 | 0.0107 |
| | 32 | 4852 | 0.9826 | 0.9858 | 2.0745e-4 | 0.4175 | 4.4289e-7 | 0.8351 | 0.8495 | 8.8578e-7 |
| | 48 | 10732 | 0.9968 | 0.9973 | 6.5822e-8 | 0.4447 | 1.8819e-12 | 0.8894 | 0.8987 | 3.7638e-12 |
| | 64 | 18916 | 0.9994 | 0.9995 | 3.2673e-12 | 0.4590 | 8.4724e-19 | 0.9180 | 0.9248 | 1.6944e-18 |
| 3 | 4 | 584 | 0.5756 | 1.0 | 0.9929 | 0.1411 | 0.3800 | 0.2821 | 0.2821 | 0.6712 |
| | 16 | 22184 | 0.9298 | 0.9663 | 0.2231 | 0.1728 | 0.0057 | 0.3456 | 0.3456 | 0.0116 |
| | 32 | 16184 | 0.9885 | 0.9920 | 0.0035 | 0.1964 | 4.4399e-7 | 0.3928 | 0.3928 | 8.8798e-7 |
| | 48 | 529640 | 0.9980 | 0.9984 | 0.0053 | 0.2098 | 1.8819e-12 | 0.6294 | 0.6294 | 5.6458e-12 |
| | 64 | 123614 | 0.9996 | 0.9997 | 8.9402e-5 | 0.2177 | 8.4724e-19 | 0.6531 | 0.6531 | 2.5417e-18 |

**Table 2. Model checking results**

| **Atomic proposition** hot_spot | **Content of .ap-file** CRG **approach** | SRG **approach** |
|---|---|---|
| HS | wait $>= 4$ | t_HS |
| HS[d] | wait_$d1 >= 4$ | N/A |
| HSx | (wait_$d1 >= 4$)\|\|(wait_$d2 >= 4$) | t_HSx |

**Table 3. Atomic proposition specifications**

.net and .def-file to the GMC2MRMC module (after having modified the net by the addition of observation transitions called t_HS and t_HSx, the meaning of which will be clear in the following), then we obtain the MRMC-ready-to-use .tra-file and the .xlab-file. The latter file contains *only* the enabled transitions for each state, and not the encoding of the markings (which is lost because we are working with the SRG). We consider the two hot_spot definitions HS and HSx as enabling conditions of two observation transitions called t_HS (self-loop transition on place wait, labelled as $< x1 > + < x2 > + < x3 > + < x4 >$) and t_HSx (self-loop transition on place wait, labelled as $4* < x >$). Therefore, the .ap file contains two lines, for t_HS and t_HSx, and will be passed, with the .xlab-file, to the APGENERATOR module, in order to produce the .lab file for MRMC.

We performed a set of model-checking tests for the CSL formulae described in Section 3, obtaining identical results as those obtained for PRISM for properties $\Psi_1$, $\Psi_2$, $\Psi_3$ and $\Phi_3$, and results which differ with an error of approximately $10^{-6}$ for the steady-state computations in properties $\Phi_1$ and $\Phi_2$. We note that the size of the CRG for each instance of the model was the same as the unfolded PRISM model, as shown in Table 2, and as expected. Instead, the number of states of the SRGs were approximately a half of the CRG/unfolded state spaces in the case of two devices (for example, when $N = 64$ the SRG contained 9507 states), and approximately one fifth of the CRG/unfolded state spaces in the case of three devices (for example, when $N = 48$ the SRG contained 90992 states).

# 5. Conclusions and future work

This paper discusses how we have exploited two CSL model checkers, PRISM and MRMC, to add CSL model checking facilities for SWN in the GREATSPN tool. We allow checking of the unfolding of an SWN via PRISM permitting us to take advantage of the efficient MTBDD data structures used in PRISM. We can alternatively check the CRG or SRG of an SWN, in the case of SRG taking advantage of the symmetries of the SWN, using MRMC. We are currently testing the implementation against a set of benchmarking SPN and SWN nets from the literature. The current paper extends the work of [12] along two directions. The first direction permits us to treat SWN models, whereas the second allows us to define atomic propositions also on transition enabling.

We note that model-checking methods for variants of CSL which refer to rewards have also been implemented in MRMC and PRISM, and can be used to verify an even wider variety of performance properties than those considered by CSL. The presence of rewards is orthogonal to our translation methods, and therefore our programs can be extended easily to accommodate rewards.

We intend to improve the links between GREATSPN and the two model-checking tools by reducing the level of expertise that a GREATSPN user requires to have of the tools. In particular, we aim to automate as much as possible the description of CSL properties, so that the user can express such properties at the net level. For example, in the context of the translation to PRISM models, the mapping of places to variables is partially lost due to unfolding, and therefore it may be advantageous to have a translator from CSL formulae of Mcol and Tcol type to formulae given in terms of PRISM variables. Another problem that we will address is the presentation of the model checking results to the GREATSPN user.

With regard to the translation to PRISM, the MTBDD representation of the transition matrix depends heavily on the ordering of the PRISM variables in the module description. A good ordering for the variables of the example is obtained by declaring together the variables which are closely related (as suggested by the developers of PRISM [22]). For example, in general, places which are connected to each other by transitions are listed together in the variable ordering. The translator does not have any associated heuristic: for the purposes of this paper we chose to manually change the variable ordering in the PRISM models obtained by our translator. To avoid such manual intervention, we intend to exploit the structure of the Petri net model to define an efficient ordering of the variables used in the PRISM model's description.

Other future work includes extension of CSL and associated tools to allow the use of immediate transitions. Recall that the reachability graph of a GSPN/SWN corresponds to a semi-Markov process, from which a CTMC can be obtained by eliminating vanishing states (in which no time elapses). As noted in [5], the elimination of vanishing states removes information about the dynamic behavior of the system which may be relevant for the evaluation of a CSL property.

## Acknowledgements

We would like to thank Dave Parker and Ivan Zapreev for help with PRISM and MRMC.

## References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. J. Wiley, 1995.

[2] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model-checking continuous time Markov chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[4] S. Bernardi, C. Bertoncello, S. Donatelli, G. Franceschinis, R. Gaeta, M. Gribaudo, and A. Horváth. GreatSPN in the new millenium. In *In Tools of Aachen 2001, International MultiConference on Measurement, Modelling and Evaluation of Computer-Communication System*, 2001. Research Report no. 760/2001, Universität Dortmund (Germany), pages 17–23.

[5] P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper. Model-checking large structured Markov chains. *Journal of Logic and Algebraic Programming*, 56:69–96, 2003.

[6] L. Capra, C. Dutheillet, G. Franceschinis, and J.-M. Ilié. Exploiting partial symmetries for Markov chain aggregation. In *MTCS'00*, volume 39 of *ENTCS*. Elsevier, 2000.

[7] G. Chiola, C. Dutheillet, G. Franceschinis, and S. Haddad. Stochastic well-formed coloured nets for symmetric modelling applications. *IEEE Transaction on Computers*, 42(11):1343–1360, 1993.

[8] E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. MIT Press, 1999.

[9] T. Courtney, D. Daly, S. Derisavi, S. Gaonkar, M. Griffith, V. Lam, and W. H. Sanders. The Möbius modeling environment: Recent developments. In *QEST'04*, pages 328–329. IEEE Computer Society, 2002.

[10] J. A. Couvillion, R. Freire, R. Johnson, W. Douglas Obal II, M. A. Qureshi, M. Rai, W. H. Sanders, and J. E. Tvedt. Performability modeling with UltraSAN. *IEEE Software*, 8(5):69–80, 1991.

[11] DANAMiCS web page. http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS/.

[12] D. D'Aprile, S. Donatelli, and J. Sproston. CSL model checking for the GreatSPN tool. In *ISCIS'04*, volume 3280 of *LNCS*, pages 543–552. Springer, 2004.

[13] S. Donatelli and L. Ferro. Validation of GSPN and SWN models through the PROD tool. In *TOOLS'02*, volume 2324 of *LNCS*, pages 131–140. Springer, 2002.

[14] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, 39(1–4):5–35, 2000.

[15] H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic process algebras: Between LOTOS and Markov chains. *Computer Networks and ISDN Systems*, 30(9–10):901–924, 1997.

[16] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model-checking Markov chains. *Software Tools for Technology Transfer*, 4(2):153–172, 2003.

[17] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[18] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Monographs in Theoretical Computer Science. Springer-Verlag, 1997.

[19] J.-P. Katoen, M. Khattri, and I. S. Zapreev. A Markov reward model checker. In *QEST'05*, pages 243–244. IEEE Computer Society Press, 2005.

[20] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 2.0: A tool for probabilistic model checking. In *QEST'04*, pages 322–323. IEEE Computer Society Press, 2004.

[21] M. Molloy. Performance analysis using Stochastic Petri Nets. *IEEE Transactions on Computers*, 31(9):913–917, 1982.

[22] D. Parker. Personal communication, 2005.

[23] Performance Evaluation group of University of Torino. The GreatSPN tool. http://www.di.unito.it/ greatspn.

[24] PRISM web site. http://www.cs.bham.ac.uk/ dxp/prism.

[25] PROD web site. http://www.tcs.hut.fi/Software/prod/.

[26] SPOT web site. http://spot.lip6.fr/.

[27] Y. Thierry-Mieg. *Techniques pour le Model-Checking de spécifications de Haut Niveau*. PhD thesis, Laboratoire d'Informatique de Paris 6, 2004.