Composition of product-form Generalized Stochastic Petri Nets: a modular approach

Simonetta Balsamo and Andrea Marin
Dipartimento di Informatica
Università Ca' Foscari di Venezia
Via Torino, 155
Venice, Italy
{balsamo,marin}@dsi.unive.it

KEYWORDS
Stochastic modeling, product-form, exact analysis

ABSTRACT

In this paper we present a novel approach to specify and analyze complex system using product-form models. The main strengths of this approach are its high modularity and its ability of dealing with a very large class of product-form models. This has been possible because the product-form analysis is based on two properties that are formulated at a very low level, i.e., the Markov implies Markov property and the Reversed Compound Agent Theorem. We propose a unifying framework for combining product-form models defined in terms of different formalisms and we give the conditions that allow the composition to be in product-form. The semantic of their combination is formally defined because the various sub-models are transformed into GSPNs with an equivalent underlying process. In particular, we illustrate with several examples that we can perform analysis of models with non-linear traffic equations, including those with some components being G-queues, product-form stochastic Petri nets, or multi-class queueing stations.

INTRODUCTION

Stochastic models have proved to play a pivotal role in the performance analysis of software and hardware architectures. The model of the system can be defined according to a large set of formalisms which ranges from Petri nets extensions to queueing systems and others. Generalized Stochastic Petri Nets (GSPNs) are a well-known formalism capable of representing complex systems in a formal way. This formalism is a stochastic extension of Petri Nets (PNs) that had been introduced to describe systems with parallel computations. Informally, PNs consist of places, transitions and arcs that connect places to transitions or vice-versa. Tokens represent the state of the model and are associated with the places. The firing of the transitions change the state of the model. In the Stochastic Petri Nets (SPNs) the transition firing takes an exponentially distributed random time and, under a set of assumptions on the firing semantic, the marking process, i.e., the stochastic process that describes the state of the model along the time, is a Continuous Time Markov Chain (CTMC). GSPNs [Marsan et al. (1995)] can be seen as a extension of SPNs that admits two types of transitions: immediate and timed. The firing of the former ones occurs in a deterministically zero time, while the firing of the latter ones requires an exponentially distributed random delay. We summarize the main strengths of GSPNs.

- It has a strong semantic. Indeed, given a GSPN model with its initial marking, the underlying stochastic process is uniquely determined. This property is not shared with all the formalisms for the stochastic modeling, e.g., queueing networks are usually described by a high level language.
- It allows for qualitative analysis of the system, by the so-called structural analysis, e.g., using the net invariants.
- The state of the model and its structure are strongly separated. For instance, we can define a structurally finite model with an underlying process with infinite states.
- The formalism, with inhibitor arcs, is very expressive. Indeed, it is has been proved that PNs with inhibitor arcs are Turing-complete.

If the process underlying a GSPN has a steady state, then we can compute its stationary probability distribution. This plays a pivotal role in the performance evaluation field, because from the stationary distribution of a model we can derive a set of significant performance indices, such as the throughput, the response time distribution, the distribution of the number of tokens in a place, and so on. However, the analysis of a model defined in terms of GSPNs may easily become an unfeasible task. This is mainly due to two reasons: the first problem is shared with all PNs models, i.e., it is computationally expensive (when not impossible) to build the set of all the reachable states of the model given its initial state. Indeed, it is known that the reachability problem (given an initial marking, is a marking reachable after any number or sequence of transition firing?)

for PNs without inhibitor arcs is EXPSPACE, while it is equivalent to the halting problem of the Turing machines for PNs with inhibitor arcs. The second problem concerns the calculation of the performance indices when the model admits a steady-state (i.e., when the underlying CTMC is ergodic). Indeed, even small models may have huge state spaces and, in the general case, the stationary state probability distribution is calculated as the solution of a linear system whose rank is the number of states of the model. Usually, this is computationally expensive and may soon lead to numerical instability of the algorithms. These problems are partially overcome by product-form models. These admit a decomposition in a set of interacting sub-models whose stationary distribution can be computed in isolation after an appropriate parameterization. Then, the stationary solution of the entire model is obtained as normalized product of the distributions of the sub-models. Although the most important example of product-form models is defined in terms of queueing networks, i.e., the BCMP theorem [Baskett et al. (1975)], the investigation of this property has involved almost all the other formalisms with an underlying CTMC. In the case of GSPNs a set of results are presented in [Coleman et al. (1996); Balbo et al. (2002)]. However, more recently, the problem of defining a unique framework for the product-form analysis of Markovian models has been investigated. In our opinion a major result is stated in [Harrison (2003)], where the author introduces the Reversed Compound Agent Theorem (RCAT) whose low-level formulation allows for its application despite of the formalism used to specify the interacting models. Using this result, and a generalization of the Markov implies Markov property [Muntz (1972)], in [Marin (2009)] we propose a unifying framework for combining product-form models defined in terms of different formalisms. In particular, we show how it is possible to model G-networks and multiclass queueing networks using GSPNs and then, we give conditions that allow the composition of these building-blocks such that the stationary solution is in product-form. This is useful to model complex systems in a framework where different types of sub-models can be combined maintaining the product-form solution. The various type of sub-models can be defined using different performance modeling formalisms, such as queueing networks and their extensions, GSPNs and some stochastic process algebra. The semantic of their combination is formally defined because the various sub-models are transformed in GSPNs with an equivalent underlying process. Moreover, we show how it is possible to compute the stationary distribution in product-form.

In this paper we present a novel approach to define a GSPN sub-model in product-form. In simple words, we aim to allow the modeler to store a library of product-form GSPN sub-models so that he can use them to describe complex architectures by specifying the way they cooperate. Note that we do not aim to define an automatic way to decide whether a sub-model is in product-form or not (although it can sometimes be done, e.g., [Coleman et al. (1996); Balbo et al. (2002)]), but we introduce the idea that given a library of models that are known to satisfy a set of properties, and a system described as a composition of these models, we can automatically decide whether that system has a product-form solution, and calculate it. According to this approach the GSPNs have to be appropriately annotated with some information that we shall describe in details in the following sections. It is worthwhile pointing out that within this framework it is possible to specify models such as G-queues, SPNs, multi-class queueing stations and PEPA models that interact. Moreover, we present a practical contribution of the theoretical result just illustrated above. Since the modularization and standardization is really important in this approach (e.g., we would like that the modeler may define a GSPN with his favorite tool and then just add the data needed for the product-form analysis) we propose an implementation of this framework based on the Petri Net Markup Language (PNML) [Weber and Kindler (2003)] and its extension to the modules [Kindler and Weber (2001)]. Finally, as further practical contribution, we show how to define GSPN models equivalent to G-queues.

We shall now recall the GSPN definition and the modular composition of GSPN submodels. We start from an example of modular combination of G-queues. Then we introduce the two basic properties for product-form models, i.e., the Markov implies Markov property and the Reversed Compound Agent Theorem, formulated at the CTMC level. We present the proposed framework to combine different sub-models into a unique GSPN that maintains the product-form solution. The submodels can be defined in terms of different formalisms and can be combined because thy can be transformed in GSPNs whose underlying process is equivalent. We discuss how to implement the framework by using PNML. Finally, we present some examples of application of the proposed technique.

GSPN FORMALISM

In this section we briefly recall the Generalized Stochastic Petri Nets (GSPN) definition. A GSPN is a 8-tuple, defined as follows:

$$GSPN = (\mathcal{P}, \mathcal{T}, I(\cdot,\cdot), O(\cdot,\cdot), H(\cdot,\cdot), \Pi(\cdot), w(\cdot,\cdot), \mathbf{m_0})$$

where: $\mathcal{P} = \{P_1, \ldots, P_M\}$ is the set of M places, $\mathcal{T} = \{t_1, \ldots, t_N\}$ is the set of N transitions (both immediate and timed). $I(t_i, P_j) : \mathcal{T} \times \mathcal{P} \to \mathbb{N}$ is the input function, $1 \leq i \leq N$, $1 \leq j \leq M$, $O(t_i, P_j) : \mathcal{T} \times \mathcal{P} \to \mathbb{N}$ is the output function, $1 \leq i \leq N$, $1 \leq j \leq M$, $H(t_i, P_j) : \mathcal{T} \times \mathcal{P} \to \mathbb{N}$ is the inhibition function, $1 \leq i \leq N$, $1 \leq j \leq M$. $\Pi(t_i) : \mathcal{T} \to \mathbb{N}$ is a function that specifies the priority of transition t_i, $1 \leq i \leq N$,

$\mathbf{m} \in \mathbb{N}^M$ denotes a marking or state of the net, where m_i represents the number of tokens in place P_i, $1 \leq i \leq N$, $w(t_i, \mathbf{m}) : \mathcal{T} \times \mathbb{N}^M \rightarrow \mathbb{R}$ is the function which specifies for each timed transition t_i and each marking $\mathbf{m}$ a state dependent firing rate, and for immediate transitions a state dependent weight, and finally $\mathbf{m_0} \in \mathbb{N}^M$ represents the initial state of the GSPN, i.e., the number of tokens in each place at the initial state. For each transition t_i let us define the input vector $\mathbf{I}(t_i)$, the output vector $\mathbf{O}(t_i)$ and the inhibition vector $\mathbf{H}(t_i)$ as follows: $\mathbf{I}(t_i) = (i_1, \ldots, i_M)$, where $i_j = I(t_i, P_j)$, $\mathbf{O}(t_i) = (o_1, \ldots, o_M)$, where $o_j = O(t_i, P_j)$ and $\mathbf{H}(t_i) = (h_1, \ldots, h_M)$, where $h_j = H(t_i, P_j)$. Function $\Pi(t_i)$ associates a priority to transition t_i. If $\Pi(t_i) = 0$ then t_i is a timed transition, i.e., it fires after an exponentially distributed firing time with mean $1/w(t_i, \mathbf{m})$, where $\mathbf{m}$ is the marking of the net. If $\Pi(t_i) > 0$ then t_i is an immediate transition and its firing time is zero. We say that transition t_a is enabled by marking $\mathbf{m}$ if $m_i \geq I(t_a, P_i)$ and $m_i < H(t_a, P_i)$ for $i = 1, \ldots, M$ and no other transition of higher priority is enabled. The firing of transition t_i changes the state of the net from $\mathbf{m}$ to $\mathbf{m} - \mathbf{I}(t_i) + \mathbf{O}(t_i)$. The reachability set $RS(\mathbf{m_0})$ of the net is defined as the set of all markings that can be reached in zero or more firings from $\mathbf{m_0}$. We say that marking $\mathbf{m}$ is tangible if it enables only timed transitions and it is vanishing otherwise. For a vanishing marking $\mathbf{m}$ let $\mathcal{T}_\alpha$ be the set of enabled immediate transitions. Then the firing probability for any transition $t_i \in \mathcal{T}_\alpha$ and any state $\mathbf{m}$ is proportional to its weight. Given a tangible marking $\mathbf{m}$ the transition with the lowest associated stochastic time fires. A GSPN is represented by a graph with the following conventions: timed transitions are white filled boxes, immediate transitions are black filled boxes, places are circles, if $I(t_i, P_j) > 0$ we draw an arrow from P_j to t_i labeled with $I(t_i, P_j)$, if $O(t_i, P_j) > 0$ we draw an arrow from t_i to P_j labeled with $O(t_i, P_j)$, if $H(t_i, P_j) > 0$ we draw an circle ending line from P_j to t_i labeled with the value of $H(t_i, P_j)$, the marking $\mathbf{m}$ is represented by a set of m_j filled circles representing the tokens in place P_j for each $j = 1, \ldots, M$. For ordinary nets we do not use labels for the arrows. If we do not specify the weight of immediate transitions it is assumed to be 1 (usually we do this when we are sure there are no conflicts among immediate transitions).

GSPN analysis consists in finding the steady-state probability for each tangible marking of the reachability set, from which one can derive other average performance indices. Some analysis techniques are presented in [Marsan et al. (1995)].

GSPNs AND MODULES

The problem of giving a correct syntax and semantic of modular compositions of GSPNs has been addressed by several authors. In fact, the modularity allows for a definition of the models that is coherent with the principles of software and hardware engineering. In this paper, we use the module definition as proposed in [Kindler and Weber (2001)]. The main idea is that a module can be instantiated several times with possibly different parameterizations. It has an input and an output *interfaces* that allow the modeler to define how every instance interacts with the rest of the model, and an *internal implementation* that is invisible to the user. This approach could somehow be seen as the well-know procedure call schema implemented by most of the programming languages, where the input/output interfaces may be interpreted as the formal input/output parameters and the instances of a module as the procedure call. Within this interpretation, when the modeler connects the interfaces of a module with other elements of the model, he is defining the association between the actual and the formal parameters. In the following example we show a GSPN module whose underlying CTMC is equivalent to that of a G-queue, see [Gelenbe (1991)].

Example 1 (GSPN model of a G-queue) *G-queues are the smallest components of G-networks. They have been successfully used in a wide range of applications such as the analysis of database systems, communication networks or neural networks. In its simplest definition, a G-queue is a single-class queueing center with exponential service time distribution. Two arrival streams of customers are allowed: one for the so-called positive customers that exactly behave like ordinary customers in standard queueing stations, and the other for the negative customers. When one of these arrives to the station it can either delete a queued (positive) customer, if any is present, or simply vanish if the G-queue is empty. By now, we assume Poisson independent arrivals for positive and negative customers. Figure 1 illustrates a possible GSPN representation of a G-queue.*

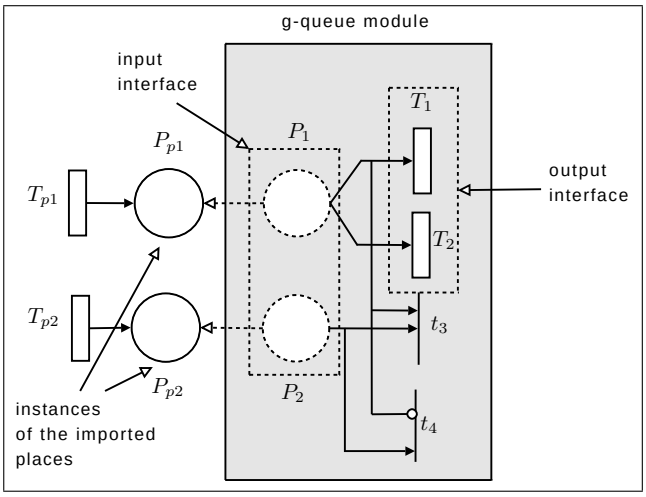

Figure 1: GSPN module equivalent to a G-queue.

The module consists of two input places P_1 and P_2. The former stores a token for each positive customer in the station, while the latter stores one token at a negative customer arrival epoch. Notice that if there is one token in P_2 then either immediate transition t_3 or t_4 is enabled. The firing of t_3 consumes also a token from P_1 (positive customer deletion), while the firing of t_4 simply consumes the token in P_2 (i.e., the queue is empty and it vanishes). Moreover, it is immediate to observe that in every tangible marking of the net there are no tokens in P_2. Finally, T_1 and T_2 model the service of a customer. We use two transitions in order to straightforwardly model two different routings for customers served in such a station. For example T_1 may model the departure of a positive customer and T_2 the departure of a negative customer. Therefore, the service rate of the station is $\mu = w(T_1, \cdot) + w(T_2, \cdot)$. The input places are associated with places P_{I1} and P_{I2}. T_{P1} and T_{P2} represent a hypothetical connection of a net with an instance of the module.

A brief introduction to concept of GSPN module. In order to keep this paper self-contained, in this part we review the main concepts concerning the idea of modularization that we refer to. For a formal definition of the syntax and of the semantic we refer to [Kindler and Weber (2001)]. Note that other approaches to PN modularization are available in literature, e.g., that used by Timenet [Zimmermann et al. (2000)], but the passage from one to the other is not complicated.

Informally, we can say that a module is a net with an interface. We can create several instances of a module, but only the objects specified in its interface are accessible from outside the instance. What is not in the interface is called internal implementation. Referring to the Object Oriented Programming, this corresponds to the encapsulation feature. The interface consists of two parts: the input part (formed by the imported objects) and the output part (formed by the exported objects). Imported objects play the same role of formal parameters in the programming languages. Indeed, they are representatives of objects that are provided when the module is instantiated. Conversely, the objects that are exported are defined inside the implementation of a module (e.g., they may be provided as referred objects for an input interface of other module instances).

One can import and export three type of objects, i.e., places, transitions and symbols. The import and export of symbols allows us to define the parameterization of the modules. For instance, input symbols may be the transition rates, the number of tokens in a place of the internal implementation of the module and so on. The technique described in [Kindler and Weber (2001)] is really flexible, so one can just import or export functions, or anything else which can be useful for the modeler purposes. In the following we use input (output) object and imported (exported) object as synonymous.

Let us now reconsider the module defined in Example 1 and let us build a simple G-network using the GSPN modularization.

Example 2 (G-networks) *A composition of G-queues is called G-network. These models have shown to be suitable for the analysis of several software and hardware architectures. Let us consider the G-network depicted in Figure 2-(A). The G-network consists of*

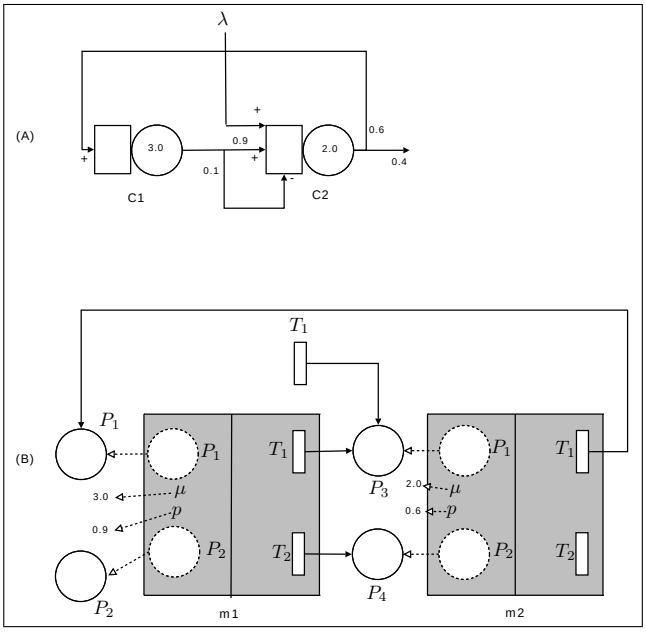

Figure 2: Use of GSPN modules to describe a G-network. (A) the original model. (B) the module composition.

two nodes, $C1$ and $C2$, with service rates 3.0 and 2.0, respectively. When customers leave $C1$ they can enter in $C2$ either as positive or negative customers, with probability 0.9 and 0.1. Customers may arrive from outside to $C2$ with rate λ. Once a customer is served by $C2$ it can leave the system with probability 0.4 or go back to $C1$ with probability 0.6.

In order to give a GSPN representation of such a network we use a composition of two instances of the module introduced in Example 1. Actually, we added two symbols in the input interface, μ and p, which represent the service rate of the node, and the probability of firing of T_1 with respect to T_2. Therefore, p and μ are used in the module definition to specify the rates of T_1 and T_2 in an obvious way: $w(T_1, \cdot) = p\mu$ and $w(T_2, \cdot) = (1-p)\mu$. Figure 2-(B) illustrates two instances of the module, $m1$ and $m2$, that are equivalent to the G-network of Figure 2-(A). In particular, the dotted arrows associate an object of an input interface with a concrete object (e.g., place P_1 in $m2$ with P_3, or μ in $m1$ with 3.0). Note that, since the scope *of the*

object names is the module instance itself, the net has no conflicts on names, e.g., P_1 in instance $m1$ cannot be confused with P_1 of the net.

THE PRODUCT-FORM FRAMEWORK

In this section we present a framework to represent complex models combining different product-form submodels into a unique GSPN that maintains the product-form solution. This work is based on two results, i.e., the Reversed Compound Agent Theorem (RCAT) [Harrison (2003)] and the Markov implies Markov property ($M \Rightarrow M$) [Muntz (1972)]. After formally defining the composition rules of the module instances, we show that, although deciding whether a GSPN model satisfies $M \Rightarrow M$ or RCAT conditions is generally very difficult to do algorithmically, it is possible to store some information in the module descriptions that will allow a tool to automatically decide if a composition of such models has product-form solution and then derive the stationary distribution. As already mentioned, this means that the modeler works with a library of product-form models that have been opportunely annotated and that can be equivalent to G-queues, BCMP stations or other models that have been proved to be in product-form. GSPN in product-form are studied in [Balbo et al. (2002)] and they are defined as GSPNs reducible to SPNs in Coleman, Henderson et al. product-form [Coleman et al. (1996)].

RCAT and the $M \Rightarrow M$ property. In this part we informally introduce RCAT and the $M \Rightarrow M$ property. Since the product-form analysis requires to study each components *as if* it were in isolation, we give the definition of what we mean by an isolated instance of a module (IIM).

Definition 1 (Isolated instance of a module) *Given an instance of a module in a net, its IIM is defined as follows:*

1. *For each input transition T_i of the module we associate a transition with a null input vector and rate χ_{ti}.*
2. *For each input place P_j we associate a place which is fed by a transition T_{pj} with a null input vector and an rate χ_{pj}.*

The rates χ_{ti} and χ_{pj} for each input transition T_i and each input place P_j are the input rates *of the IIM, and $\mathcal{I}$ is the set of input rates.*

As an instance we can consider the net of Figure 1 where we can observe an IIM of the G-queue. The input rates are the rates of T_{p1} and T_{p2}. We now introduce the set of reachable states of a module.

Definition 2 (Reachability set of a module) *The reachability set of a module is the set of all the markings reachable from its IIMs.*

Note that, in general, the reachability set of a module is not finite, and this is one of the reasons that makes the automatic decision of the following properties a very difficult task.
In order to simplify the formulation of RCAT and $M \Rightarrow M$ for GSPN modules, we limit the output objects to be transitions or symbols. This can be done without loss of generality possibly using immediate transitions.

Definition 3 (RCAT-compatible IIM) *We say that a IIM of a module is* RCAT-compatible *if and only if the following three conditions are satisfied:*

1. *For every tangible state, the instances of the input transitions must be always enabled. Informally, we can say that the module internal implementation cannot inhibit the input transition in any tangible marking.*
2. *Let $\mathbf{m}$ be a tangible marking of the reachability set. Then, if T_o is an output transition there must exist one tangible marking $\mathbf{m}'$ such that $\mathbf{m}$ is reachable by $\mathbf{m}'$ through the firing of T_o.*
3. *For every pair of $\mathbf{m}$ and $\mathbf{m}'$ such that $\mathbf{m}$ is reachable from $\mathbf{m}'$ through the firing of output transition T_o the following relation holds:*

$$\pi(\mathbf{m}')w(T_o, \mathbf{m}') = K_o \pi(\mathbf{m}), \tag{1}$$

where $\pi(\mathbf{m})$ is the stationary probability of marking $\mathbf{m}$ and $K_o \in \mathbf{R}^+$.

These three conditions are just a rewriting of RCAT conditions [Harrison (2003)]. Finally, we observe that K_o is a constant which is associated with each output transition T_o that in general depends on the structure of the module and the input rates.

Definition 4 ($M \Rightarrow M$-compatible module) *We say that a module is $M \Rightarrow M$-compatible if and only if the following three conditions are satisfied:*

1. *See Condition 1 of RCAT-compatible definition.*
2. *See Condition 2 of RCAT-compatible definition.*
3. *Let $\mathbf{m}$ be a tangible state reachable from and $\mathcal{M} = \{\mathbf{m}'\}$ through the firing of an output transition T_o. Then, the following relation holds:*

$$\sum_{\mathbf{m}' \in \mathcal{M}} \pi(\mathbf{m}')w(T_o, \mathbf{m}') = K_o \pi(\mathbf{m}), \tag{2}$$

where $\pi(\mathbf{m})$ is the stationary probability of marking $\mathbf{m}$ and K_o is a linear combination of the input rates.

In this case, it is not immediate to see that the conditions on the GSPN module are equivalent to the $M \Rightarrow M$ property. Indeed, this property is formulated in the context of queueing theory, therefore it involves concepts such as customers, class of customers, work-conserving an so on. The proof of the equivalence can be found in [Marin (2009)] and is based on a generalization of the $M \Rightarrow M$.

Product-form composition and derivation of the stationary probabilities. Let us introduce the problem of the product-form composition of the module instances with ax example. Suppose that $m1$ and $m2$ are instances of RCAT-compatible module(s). Our aim is to define the appropriate input rates of the IIMs of $m1$ and $m2$ such that if $\mathbf{m} = (\mathbf{m}_1, \mathbf{m}_2)$ is a state of the original net, and $\mathbf{m}_1$ ($\mathbf{m}_2$) the state of $m1$ ($m2$), then:

$$\pi(\mathbf{m}) \propto \pi_1(\mathbf{m}_1)\pi_2(\mathbf{m}_2),$$

where $\pi(\mathbf{m})$, $\pi_1(\mathbf{m}_1)$ and $\pi_2(\mathbf{m}_2)$ are the stationary distributions of the whole net and of the IIMs of $m1$ and $m2$, respectively.
Obviously, these operations have not to be manually performed by the modeler, but we expect a tool to do them automatically. Indeed, the difficult task is the definition of the input rates.
Both $M \Rightarrow M$ and RCAT give a way to set these rates and they depend on the solution of a system called *traffic equation system*. Note that, in this framework it is not the case that the system of traffic equations is always linear as for example in BCMP queueing networks [Baskett et al. (1975)]. Therefore, we are able to study more general cases of product-form models than those based on the analysis of queueing networks.
The traffic equations depend on they way the module instances are connected. We allow two types of connections as specified by the following definition, where we consider that the arc weights are all 1.

Definition 5 (Valid net) *A net consisting of instances of $M \Rightarrow M$-compatible or RCAT-compatible modules is* valid *if the connection among the instances of the modules are such that for each instance:*

1. *if T_i is an input transition then it is associated with just one output transition of another instance or a transition with null input and output vector and vice-versa*
2. *each place of the net is associated with one input place and an output transition which is not associated with an input transition can have an outgoing arc to just one place.*

In a valid net the interactions among the module instances are pairwise. In other words, at most two instances can change their markings as a consequence of the firing of a transition. Pairwise interactions are the only interactions that are considered both by RCAT and the $M \Rightarrow M$ property.
It is worthwhile pointing out that the validity of a net can be decided by a trivial algorithm.

THE FRAMEWORK IN PRACTICE

In this section we illustrate how we use the theoretical results recalled in the previous section to specify complex systems with product-form solutions. Informally, we can say that once the modeler has chosen the modules to instantiate, he/she connects them in one of the two ways that have been described. This operation can be seen as a graphical way to specify the traffic equations. We think that this is the key-point of our approach, i.e., the modeler uses a library of module whose behavior is known and when he connects them he is simply specifying the system of traffic equations. Note that, although this idea may seems trivial, it should be pointed out that it can be implemented thanks to the combination of the recent theoretical results about product-form such as RCAT and the idea of mapping each formalism into equivalent GSPNs.
What do we need to know about a module to be able to generate the system of traffic equations? As we already mentioned, the analysis of a single module with the aim of deciding whether it is RCAT-compatible or $M \Rightarrow M$-compatible may be a really hard task. Specifically, it can be often the case that the reachability set of the module may be not finite. In order to overcome this problem, we introduce the concept of product-form GSPN module (PF-GSPN module). Let $\mathcal{I}$ be the set of the input rates and $\mathcal{V}$ be the set of the parameters of the module.

Definition 6 (PF-GSPN module) *A PF-GSPN module is a module with the following features:*

- $f_{RCAT} : \mathcal{I} \times \mathcal{V} \rightarrow \{\textbf{\textit{true}}, \textbf{\textit{false}}\}$ *is a boolean function which assumes the value* ***true*** *if, for a given parameterization, the module is RCAT-compatible*
- $f_{M \Rightarrow M} : \mathcal{I} \times \mathcal{V} \rightarrow \{\textbf{\textit{true}}, \textbf{\textit{false}}\}$ *which assumes the value* ***true*** *if, for a given parameterization, the module is $M \Rightarrow M$-compatible.*
- *For each output transition T_o, $K_o : \mathcal{I} \times \mathcal{V} \rightarrow \mathbf{R}^+$ is the function which specifies the reversed rate of T_o in case of RCAT-compatibility or the sum of the reversed rates in case of $M \Rightarrow M$-compatibility. Obviously K_o is defined only if $f_{RCAT}(\mathcal{I}, \mathcal{V}) \vee f_{M \Rightarrow M}(\mathcal{I}, \mathcal{V})$.*

We now illustrate a set of example of PF-GSPN modules.

Example 3 (G-queue) *Let us consider again the G-queue of Example 1. In this case we have $\mathcal{I} = \{\chi_{p1}, \chi_{p2}\}$*

and $\mathcal{V} = \{\mu, p\}$. The station is known to be always in RCAT product-form, [Harrison (2003)], while if fulfills the $M \Rightarrow M$ property only if there are not negative customer arrival (i.e., it is a standard exponential queue), therefore, we have:

$$f_{RCAT}(\chi_{p1}, \chi_{p2}, \mu, p) = \textbf{\textit{true}} \quad always$$
$$f_{M \Rightarrow M}(\chi_{p1}, \chi_{p2}, \mu, p) = \textbf{\textit{true}} \quad if\ \chi_{p2} = 0$$

From the G-network analysis [Gelenbe (1991)] the stationary probability of observing m customers in P_1 is $(1-\rho)\rho^m$, where $\rho = \chi_{p1}/(\mu + \chi_{p2})$. Then, we straightforwardly obtain:

$$K_1(\chi_{p1}, \chi_{p2}, \mu, p) = \frac{\chi_{p1}\mu}{\chi_{p2} + \mu} p$$
$$K_2(\chi_{p1}, \chi_{p2}, \mu, p) = \frac{\chi_{p1}\mu}{\chi_{p2} + \mu} (1 - p)$$

Example 4 (A model of a shared bus contention) *In this example we address the problem introduced in [Afshari et al. (1982)], where the author propose a queueing model to study the access to a shared bus of a set of customers that are clustered into R classes. The authors assume that the bus is able to handle K simultaneous transmissions. As soon as a channel of the bus becomes available, a waiting customer is chosen with uniform probability among the queued ones to get served. The service time distribution is exponential and identically distributed for all the customer classes. In the paper the authors prove the stationary distribution and that the station satisfies the $M \Rightarrow M$ property.*
In Figure 3 we propose a module of this system considering $R = 2$ classes of customers. Customers of

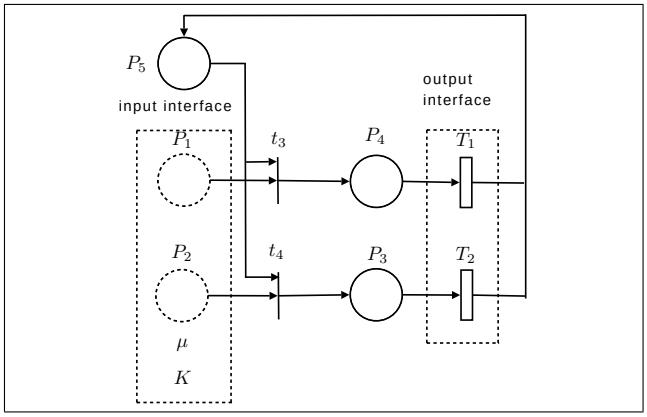

Figure 3: PF-GSPN module of the queueing model of a shared bus contention for two classes of customers.

*class 1 and 2 arrive to input places P_1 and P_2, respectively. Place P_5 contains as many tokens as the free channels of the bus are. Immediate transitions t_3 and t_4 model the contention policy of the bus, i.e. their weight function is $w(t_3, \mathbf{m}) = m_1$ and $w(t_4, \mathbf{m}) = m_2$, where $\mathbf{m} = (m_1, \ldots, m_5)$ is a tangible marking and m_i denotes the number of token in P_i. The rates of timed transition T_1 and T_2 are μ. Finally, K is the initial number of tokens in place P_5, i.e., the number of channels of the shared bus. In this case $\mathcal{I} = \{\chi_{p1}, \chi_{p2}\}$ and $\mathcal{V} = K, \mu$. As mentioned, in [Afshari et al. (1982)] the authors prove that the model satisfies the $M \Rightarrow M$ property, the $f_{M \Rightarrow M}(\mathcal{I}, \mathcal{V}) =$ **true** always. In [Marin (2009)] we prove that it satisfies RCAT conditions if $K = 1$, i.e., $f_{RCAT}(\mathcal{I}, \mathcal{V}) =$ **true** if $K = 1$. Finally, $K_1(\mathcal{I}, \mathcal{V}) = \chi_{p1}$ and $K_2(\mathcal{I}, \mathcal{V}) = \chi_{p2}$.*

For the sake of brevity we omit to provide other examples. We just mention that in [Balsamo and Marin (2008)] a set of GSPN models equivalent to the BCMP stations are defined and can be straightforwardly used in this context.

Automatic derivation of the traffic equations. In order to be able to decide whether a valid net is in product-form, and in this case provide the stationary solution, we need to generate and solve the set of traffic equations. The unknowns of these equations are the input rates of the module instances of the net. If we are able to solve the traffic equations we can check if they satisfy the conditions for the $M \Rightarrow M$ or RCAT application for each instance of the module using functions f_{RCAT} and $f_{M \Rightarrow M}$. If this is the case, then we can derive the stationary distribution of the IIMs associated with every module instance using the input rates obtained by the solution of the traffic equations. Then, the stationary solution of the original net is proportional to the product of these stationary solutions.
In the following, in order to avoid conflicts of names in the equations, we use the notation instance_name.object (e.g., $m1.\chi_{p1}$). A valid net admits only two types of connections. Each of these generate the following equations:

- Suppose that output transition T_o of instance mk is the input transition T_i of instance mh with $mk \neq mh$. In this case we have $mh.\chi_{ti} = mk.K_o(mk.\mathcal{I}, mk.\mathcal{V})$.
- Suppose that the set output transitions $\mathcal{T}^*\{mk.T_o\}$ is such that all the elements $mk.T_o$ have an outgoing arc to P_{Ii} that is an instance of input place P_i of mh, with $mk \neq mh$. In this case we have that:

$$mh.\chi_{ti} = \sum_{mk.T_o \in \mathcal{T}^*} mk.K_o(mk.\mathcal{I}, mk.\mathcal{V})$$

It is out of the scope of this paper to address the problem of an efficient solution of such a system. However, using Muntz's result [Muntz (1972)] we can state that the system is linear if all the instances of the modules are $M \Rightarrow M$-compatible. An approach used in [Argent-Katwala (2006)] is to export the equations in ASCII format and solve them using general purpose software

on Mathematics. If all the used modules have a finite reachability set, then the algorithm presented in [Marin and Rota Bulò (2009)] may be used.

EXAMPLE

The purpose of the following example is to show how the technique previously described may be applied to study a system consisting of sub-components that cannot be modeled by ordinary queueing stations.

System description. Two classes of requests arrive through a communication line from two networks. The communication channel is bidirectional and has a waiting room where the packets are stored. When a transmission is completed a packet is chosen from the waiting room according to a random policy. Once transmitted, the requests are pre-processed by an ad-hoc system and finally sent to the database. Some of the requests of the first class may be converted into requests of the second class. In some cases, the pre-processing phase may decide to cancel a transaction that has already been sent to the database. Some transactions fail, and have to be sent back to the communication line to get processed again. The database answers are sent back to the clients through the channel. Figure 4 shows a sketch of this system.

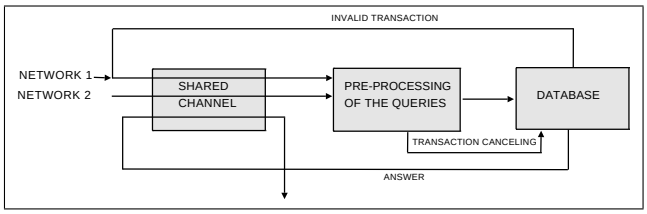

Figure 4: Software architecture analyzed in the Example section.

Model description The modeling assumptions are the following. The channel behaves like a shared bus as described in Example 4 and the database is modeled by a G-queue as described in Example 1. This means that the service time distributions are exponential and class independent both for the database and the communication lines. The pre-processing of the requests is modeled by the Module of Figure 5. For brevity, we do not specify each phase of the processing, but it is important to note that fork-join constructs are present, and this makes the model impossible to be studied by most of the existing product-form analyzer. For this module, we have $\mathcal{I} = \{\chi_{p1}, \chi_{p2}\}$ and $\mathcal{V} = \emptyset$. In [Marin (2009)] is proved that $f_{M \Rightarrow M}(\mathcal{I}) = f_{\text{RCAT}}(\mathcal{I}) = \textbf{true}$, and $K_4(\mathcal{I}) = \chi_{p1}$ and $K_5(\mathcal{I}) = \chi_{p2}$ (note that also the stationary distribution is provided). Another module that we use and that has not been previously described is

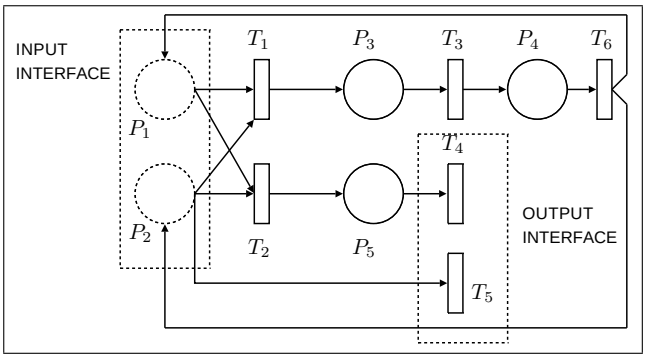

Figure 5: SPN module of the query pre-processing phase.

a switching module. This simply routes the tokens that arrive to its incoming place according to a static probability as depicted by Figure 6. In our framework we can

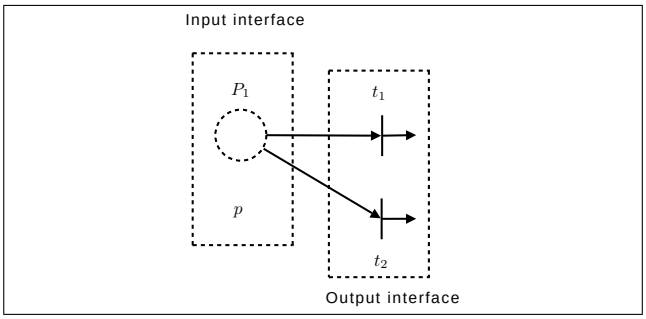

Figure 6: Simple router module. We have $w(t_1, \cdot) = p$ and $w(t_1, \cdot) = 1 - p$. $\mathcal{I} = \{\chi_{p1}\}$ and $\mathcal{V} = \{p\}$. $f_{M \Rightarrow M} = f_{\text{RCAT}} = \textbf{true}$ and $K_1(\mathcal{I}, \mathcal{V}) = p\chi_{p1}$, $K_2(\mathcal{I}, \mathcal{V}) = p\chi_{p2}$.

model the system as depicted by Figure 7, where $m1$ is an instance of the module of a shared bus described in Example 4, $m2$ is an instance of a router, $m3$ an instance of the SPN module of Figure 5 and, finally, $m4$ is an instance of a G-queue module described in Example 1. The model parameters are: the firing rates of T_A (λ_A) and T_B (λ_B), i.e., the arrival rates of the requests, μ_{TR}, i.e., the transmission rate of one line of the channel, p_{SWITCH}, i.e., the probability that a request of the first type becomes a request of the second type, p_{ERR}, i.e., the probability of reprocessing of a transaction and finally μ_{SER}, i.e., the service rate of the database.

Traffic equations. In our framework we can algorithmically derive the traffic equations using the rules pre-

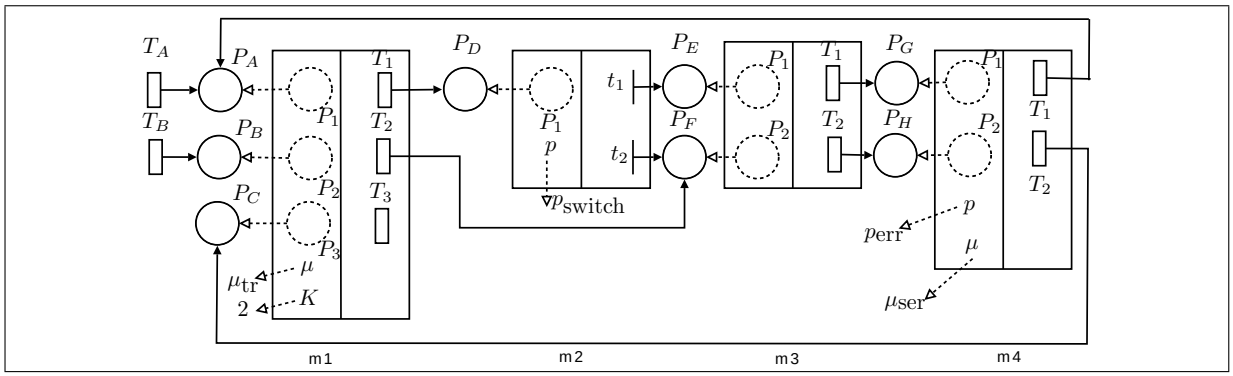

Figure 7: Modular composition of the system of Figure 4.

sented in the previous sections, obtaining:

$$\begin{cases} m1.\chi_{p1} = \lambda_A \\ m1.\chi_{p2} = \lambda_B \\ m1.\chi_{p3} = m4.K_2 = \frac{m4.\chi_{p1}}{m4.\chi_{p2}+\mu_{\text{SER}}}(1-p_{\text{ERR}})\mu_{\text{ERR}} \\ m2.\chi_{p1} = m1.K_1 = m1.\chi_{p1} \\ m3.\chi_{p1} = m2.K_1 = m2.\chi_{p1}p_{\text{switch}} \\ m3.\chi_{p2} = m2.K_1 + m1.K_2 \\ \qquad = m2.\chi_{p1}(1-p_{\text{switch}}) + m1.\chi_{p2} \\ m4.\chi_{p1} = m3.K_1 = m3.\chi_{p1} \\ m4.\chi_{p2} = m3.K_2 = m3.\chi_{p2} \end{cases}$$

Once derived the solution for the traffic equations, this is used to set the input rates of the IIMs of the module instances. Then, we observe that all the IIMs are RCAT-compatible and therefore the model is in product-form. The stationary solution π_i of each IIM for $i = 1, \ldots, 4$ is then derived and the stationary probabilities π of the whole model are such that $\pi \propto \prod_{i=1}^{4} \pi_i$. Knowing the stationary distribution π of the model allows us to compute some interesting performance indices, e.g., the mean response time of the database, or distribution of the number of customers in the communication line.

CONCLUSIONS

In this paper we have presented a novel approach to analyze product-form GSPNs. Its main strengths are the high modularity and the fact it is capable to deal with several product-form model classes, such as BCMP queueing networks, G-queues, product-form SPNs, and so on. The idea underlying this work is to use the module concept as defined in [Kindler and Weber (2001)] to define product-form models. These have to be annotated in order to allow a software tool to take advantage from the product-form property in the analysis phase. It can be shown that all this work may be implemented using PNML without violating the standard. Finally, it is worthwhile pointing out that in this framework, a modeler is not supposed to have particular knowledge about product-form models or GSPN modeling. Indeed, modelers just need to pick some modules from a library and then create and connect their instances according to the simple rules that we have described. Then, the steady state analysis and the derivation of the desired performance indices can be automatically computed. Further research efforts should have two directions. One is the implementation or the extension of an existing tool capable to perform such an analysis. This should not be hard, since it suffices to specify an appropriate PNML grammar and use a symbolic tool to solve the traffic equations. Another research open problem could deal with the possibility of connecting the module instances with arcs with arbitrary weights. This introduces some complex problems in the analysis but would enhance the flexibility of the framework.

REFERENCES

Afshari P.V.; Bruell S.C.; and Kain R.Y., 1982. *Modeling a new technique for accessing shared buses.* In *Proc. of the Computer Network Performance Symp.* ACM Press, New York, NY, USA, 4–13.

Argent-Katwala A., 2006. *Automated product-forms with Meercat.* In *SMCtools '06: Proc. from the 2006 workshop on Tools for solving structured Markov chains.* ACM, New York, NY, USA, 10.

Balbo G.; Bruell S.C.; and Sereno M., 2002. *Product Form Solution for Generalized Stochastic Petri Nets. IEEE Trans on Software Eng*, 28, no. 10, 915–932.

Balsamo S. and Marin A., 2008. *From BCMP Queueing Networks to Generalized Stochastic Petri Nets: An Algorithm and an Equivalence Definition.* In *Proc. of ESM 2008.* Le Havre, FR, 447–455.

Baskett F.; Chandy K.M.; Muntz R.R.; and Palacios F.G., 1975. *Open, Closed, and Mixed Networks of Queues with Different Classes of Customers. J ACM*, 22, no. 2, 248–260.

Coleman J.L.; Henderson W.; and Taylor P.G., 1996. *Product form equilibrium distributions and a convolution algorithm for Stochastic Petri nets. Perform Eval, Elsevier*, 26, no. 3, 159–180.

Gelenbe E., 1991. *Product form networks with negative and positive customers. Journal of Applied Prob*, 28, no. 3, 656–663.

Harrison P.G., 2003. *Turning back time in Markovian process algebra. Theoretical Computer Science*, 290, no. 3, 1947–1986.

Kindler E. and Weber M., 2001. *A universal module Concept for Petri nets.* In G.J. und Robert Lorenz (Ed.), *Proc. of 8th Workshops AWPN.* Katholische Universität Eichstätt, Germany, 7–12.

Marin A., 2009. *On the relations among product-form stochastic models.* Ph.D. thesis, Università Ca' Foscari di Venezia, Venice.

Marin A. and Rota Bulò S., 2009. *A general algorithm to compute the steady-state solution of cooperating Markov chains.* In *Proc. of 17th Annual Meeting of the IEEE Inter. Symp. MASCOTS.* In press.

Marsan M.A.; Balbo G.; Conte G.; Donatelli S.; and Franceschinis G., 1995. *Modelling with generalized stochastic Petri nets*. Wiley, New York, NY, USA.

Muntz R.R., 1972. *Poisson Departure Processes and Queueing Networks*. Tech. Rep. IBM Research Report RC4145, Yorktown Heights, New York.

Weber M. and Kindler E., 2003. *Petri Net Technology for Communication-Based Systems*, H. Ehrig, W. Reisig, G. Rozenberg, H. Weber ed., chap. The Petri Net Markup Language. 124–144.

Zimmermann A.; Freiheit J.; German R.; and Hommel G., 2000. *Petri Net Modelling and Performability Evaluation with TimeNET 3.0*. In *TOOLS '00: Proc. of the 11th Int. Conf. on Computer Perf. Eval.: Modelling Techniques and Tools*. Springer-Verlag, London, UK, 188–202.