Available online at www.sciencedirect.com
ScienceDirect
The Journal of Systems and Software 80 (2007) 1843–1865
The Journal of Systems and Software
www.elsevier.com/locate/jss

# Performance evaluation of UML design with Stochastic Well-formed Nets

Simona Bernardi [a,*,1], José Merseguer [b,2]

[a] Dipartimento di Informatica, Università di Torino, 10149 Torino, Italy
[b] Departamento de Informática e Ingeniería de Sistemas, Universidad de Zaragoza, 50018 Zaragoza, Spain

Received 20 April 2006; received in revised form 31 January 2007; accepted 10 February 2007
Available online 25 February 2007

**Abstract**

The paper presents a method to compute performance metrics (*response time*, *sojourn time*, *throughput*) on Unified Modeling Language design. The method starts with UML design annotated according to the UML Profile for Schedulability, Performance and Time. The UML design is transformed into a performance model where to compute the referred metrics. Being the performance model a Stochastic Well-formed Net, the method is enabled to analyze systems where the object identities are relevant as well as those where they are not. A complete case study reveals how to apply the method and its usefulness.


*Keywords:* Unified modeling language; UML profile for schedulability performance and time; Stochastic Well-formed Net; Software performance engineering; Model driven architecture

## 1. Introduction

The quantitative analysis of software systems has been a topic of interest during the last decades. Today, the challenges still remain open, since the software demands high-quality non-functional properties such as performance, scalability, dependability or quality of service.

According to the Model Driven Architecture (MDA) approach, conceived by the Object Management Group (OMG), the development of the entire software life-cycle can be seen as a process based on models. MDA proposes an efficient use of system models, where the primary modeling notation is the Unified Modeling Language (UML) (OMG-UML, 2005). While other OMG standard languages, such us the UML Profile for Schedulability, Performance and Time (UML-SPT) (OMG-UML-SPT, 2005), are used to add information to the original UML system design, in this case quantitative information.

One of the main goals of MDA is to support transformations between models that emphasize different views and levels of the system. In the last years, several works have been proposed to transform automatically UML-SPT models (i.e., UML models enriched with performance annotations) into performance models, using as target modeling formalism either queueing networks (Cortellessa and Mirandola, 2000; Gu and Petriu, 2005), or stochastic process algebras (Canevet et al., 2003; Jansen et al., 2003) or stochastic Petri nets (Bernardi et al., 2002; Bondavalli et al., 2001; López Grao et al., 2004). Other works transform the UML-SPT models into any kind of performance model (Woodside et al., 2005; Grassi et al., 2005) or into simulation models (Balsamo and Marzolla, 2003; De Miguel et al., 2000). Performance models, unlike UML-SPT models, benefit from the analysis techniques and tools developed during decades.

* Corresponding author.
*E-mail address:* bernardi@di.unito.it (S. Bernardi).
[1] Simona Bernardi has been supported by the European IST project CRUTIAL-027513 (CRitical UTility InfrastructurAL resilience).
[2] José Merseguer has been supported by the project DPI2006-15390 of the Spanish Ministry of Science and Technology.

0164-1212/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2007.02.029

In Merseguer et al. (2002) and Bernardi et al. (2002) we proposed a method that converts automatically UML state machines (SM) and sequence diagrams (SD) into Generalized Stochastic Petri Net (GSPN). This method can be used to analyze software systems in which the objects of the classes are considered indistinguishable (i.e., the object identities are not modeled). The method is scalable with respect to the class population, i.e., the number of objects per class.

From the UML point of view, models use to be interpreted in terms of objects that can be distinguishable among each other. But, when large class populations have to be modeled and the object identities need to be considered as well, GSPN use to scale bad and their modeling and analysis becomes intractable. Stochastic Well-formed Nets (SWN) – the colored version of GSPN – overcome these difficulties by providing a support to the construction of compact models (Balbo, 1995; Bobbio et al., 2001). Concretely, the common behavior of several entities constituting a large system is described by the SWN topology and different entities are identified by different colored tokens. Moreover, the analysis of SWN models can be carried out with efficient techniques that exploit model symmetries to reduce the size of the state space representation (Chiola et al., 1993).

In this work, we propose a new transformation method of UML-SPT design into SWN models, that builds on the one proposed in Merseguer et al. (2002) and Bernardi et al. (2002). The method exploits the properties of the SWN to gain scalable, with respect to class population, performance models that represent the object identities.

From the performance analysis point of view, it is worth noting that it is not always relevant to consider the object identities. Indeed, for some systems, the analysis of the colored model and the corresponding uncolored model gives the same performance results. Then, software analysts face the modeling issue of whether considering the object identities in the performance evaluation or not. To the best of our knowledge, there is no work that has identified a set of system conditions that guarantee same results when analyzing the colored and uncolored models. The most we can say is that there are systems (Ballarini et al., 2002, 2003; Franceschinis et al., 2001) where the results from colored and uncolored models differ. While there are other case studies, such as Merseguer et al. (2003), where the two models are performance equivalent.

Therefore, the main contribution of this work is to provide software engineers with a method, suited to object-oriented methodologies, for the computation of performance metrics, such as *response time*, *sojourn time* and *throughput*, on UML design.

The application of the method does not require expertise in Petri net modeling and analysis since all the method steps but one have tool support. Moreover, guidelines are provided to the analysts in the step of the method that is not currently automated, that is the conversion of uncolored models into colored ones. Nevertheless, the method does not still support *performance assessment*, so the software engineers need knowledge to read the SWN and then pinpoint the performance problems in the UML model. This an easy task since traceability between models is provided.

### 1.1. Related works

The problem of deriving formal models from UML design has been studied by several researchers during these last seven years, so there exists a lot of literature on this topic. We restrict our discussion to the works that aim at deriving, from UML design, formal models that capture the object identities.

One of the first proposals of deriving high-level Petri net models, that preserve the object identities, from UML design was made by Baresi and Pezzè (2001). The authors exploit the net composition property to obtain the final high-level Petri net model of the system. The behavior of the classes, specified by UML statecharts, is represented by high-level Petri net component models with interface places. The objects belonging to the same class share the same net structure and their identity is captured by the token values. The collaboration diagram guides the connection among the component models, which is carried out through the merging of interface places, then providing the final model for the system under study.

Saldhana and Shatz (2000) propose a method to derive Colored Petri Net (CPN) models from UML statecharts and collaboration diagrams. As in Baresi and Pezzè (2001), the Petri net composition approach is adopted to get the final analyzable CPN model. However, unlike Baresi and Pezzè (2001), the objects belonging to the same class do not share the same Petri net structure, then producing a final CPN model that is not scalable with respect to the class population. More recently, in Hu and Shatz (2004), the authors investigate model-driven simulation and propose the use of the CPN models derived according to Saldhana and Shatz (2000), as the engine that drives the simulation. The scenarios generated by the simulation runs are represented as Message Sequence Charts, then providing a support to the user in the verification of system properties, such as checking the occurrence of events and the causality between event occurrences.

The work of Bouabana-Tebibel and Belmesk (2004) builds on the Shatz et al.'s approach (Saldhana and Shatz, 2000) and proposes new rules of interaction of the CPN component models. Objects diagrams are also considered by Bouabana-Tebibel and Belmesk (2004) providing information on the class population and on the object attribute values. Token colors are not only used to specify object identities but also to represent their attribute values.

The works cited above have a different focus with respect to our proposal; indeed, the gain of the automatic mapping of UML to high-level Petri nets in Baresi and Pezzè (2001), and to CPN in Saldhana and Shatz (2000), Hu and Shatz (2004) and Bouabana-Tebibel and Belmesk

(2004), is to have a formal model to prove system *qualitative* properties (e.g., absence of deadlocks, fairness). In our proposal, instead, we aim at deriving Petri net *quantitative* models to be used for the performance evaluation of the system. Moreover, it should be observed that the SWN models are also suitable for qualitative analysis purposes.

The approach of Pettit and Gomaa (2004) derives, in a semi-automatic manner from UML design, CPN models suitable for both qualitative and quantitative system assessment. This method consists in translating, systematically, UML collaboration diagrams into CPN models by means of a set of predefined CPN model components, called "templates". Such templates are defined according to a set of object behavioral roles.

In Canevet et al. (2003) a method is proposed to derive automatically performance models from UML statecharts and collaboration diagrams. In this case, the Performance Evaluation Process Algebra (PEPA) is the target modeling formalism, then exploiting its composition capabilities. So, each UML statechart is mapped onto a PEPA component. According to the information drawn from the collaboration diagram, these PEPA components are synchronized over common activities, through the cooperation operator, to obtain the final PEPA system model. As in Saldhana and Shatz (2000), the approach in Canevet et al. (2003) is not scalable with respect to the class population; since a UML statechart is assumed to represent the behavior of a single object, then a PEPA component is generated for each object in the system.

Finally, the works (Merseguer et al., 2003; Bernardi and Merseguer, 2006) combine the use of UML state machines and a sequence diagram to produce models for performance evaluation and quality of service analysis. In this paper, we have built on the experience gained on them.

### 1.2. Structure of the article

The paper is organized as follows. Section 2 presents an overview of the method that consists of four main steps. Section 3 describes how to add performance information to the UML design of the target system (first step). Section 4 goes over how to translate the UML-SPT design into SWN components (second step). Section 5 accomplishes the third step of the method, that gains an SWN performance model from the SWN components. Section 6 describes how to analyze the performance model, i.e., the last step. Section 7 applies the method to the case study of a software retrieval system. Finally, in Section 8 conclusions and future work are presented.

## 2. Overview of the proposed method

The method aims at supporting the software analyst in the performance evaluation of UML design. The UML design, enriched with performance annotations, is then converted into an SWN, where the metrics can be effectively computed via stochastic analysis. The method has been devised to satisfy the following properties:

- *Suitability for object-oriented methodologies*. Since the method uses SMs, SDs, interaction overview diagrams (IOD) and deployment diagrams (DD), it can be applied within any software methodology that encompasses these types of UML diagrams as design notation.
- *Preservation of object identities*. Since we interpret the UML design in terms of the object-oriented paradigm, then objects carry identities. SWN models can deal with this interpretation, in particular the SWN concept of color can be exploited to represent object identities.
- *UML-SPT compatibility*. The OMG standard profile for Schedulability, Performance and Time (OMG-UML-SPT, 2005) is used to annotate, on the UML design, the performance characteristics of the system. The UML-SPT profile is easy to apply from the software analyst point of view and it has been integrated in several CASE tools.
- *Repeatability and partial automation*. The software analyst is provided with repeatable steps to gain the final performance SWN model. The method is partially automated and three software tools help the software analyst in its application: ArgoSPE (http://argospe.tigris.org), algebra (Bernardi et al., 2001) and GreatSPN (http://www.di.unito.it/~greatspn).
- *Traceability*. The use of labels in the SWN elements (i.e., places, transitions) supports traceability between the UML design and the SWN performance model. Indeed, the UML elements (such as states, events or activities) can be identified by their names as labels in the SWN elements.
- *Scalability*. Compact SWN performance models are derived from large UML design and they can be efficiently analyzed by using the SWN solvers that exploit model symme- tries.

The method consists of four steps that are summarized in the following.

### 2.1. Construction of a UML model for SWN analysis

The input to the method is a UML design that consists of a set of SMs, modeling the behavior of system components (classes), and one *performance scenario* (OMG-UML-SPT, 2005), modeling the interactions among system components (i.e., the messages exchanged among the classes).

The performance scenario can be represented either by an SD or, alternatively, by an IOD with a set of referenced SDs. The IOD is a new notation introduced in UML (OMG-UML, 2005) that uses the UML activity diagram notation, where a node can be either an SD or a referenced SD, and it is a way to describe interactions where messages and lifelines are abstracted away.

The UML design has to be annotated with the *performance input parameters* and with the *performance metrics* to be computed. Both are specified using the annotation approach of the UML-SPT profile. Appendix A details the set of UML-SPT extensions used in the method.

Performance input parameters include system workload, activity demand, routing rates and message transmission delays. Moreover, when the network speed is a parameter of interest, then a DD, where to annotate it, augments the UML design. The performance metrics encompass instead *scenario response time*, *sojourn time* and *throughput*.

### 2.2. Translation of the UML model into SWN

This step includes two sub-steps. First, each SM is translated into an SWN component. Second, the performance scenario is translated into one SWN component.

This paper provides a translation to convert an SD with combined fragments into an SWN (Section 4.2). It is important to note that when the performance scenario is represented by an IOD with a set of referenced SDs, they have to be converted into an SD where to apply such translation. Then we use the technique proposed by Haugen et al. (2005) to get a unique and equivalent SD.

The translation of the SM and the SD ensures the *traceability* of the method. In particular, the performance input parameters in the UML-SPT model are mapped onto the input parameters of the SWN components, basically transition firing rates/weights and place initial markings.

### 2.3. Obtention of the performance SWN

The goal is to get an *analyzable* SWN $\mathcal{N}$ for the system, that represents the internal behavior of the system components as well as their interactions. $\mathcal{N}$ is obtained by exploiting the SWN composition features: the SWN components representing the SMs are composed over places to get an intermediate model $\mathcal{N}_{\mathrm{SMs}}$. Then, the latter is composed over transitions with the SWN component representing the scenario, to get the final SWN model $\mathcal{N}$. Appendix B gives an introduction to the basic concepts of SWN and to their composition operators.

The method supports two different interpretations of the possible concrete interactions modeled by the performance scenario. Each interpretation corresponds to define a different initial marking of $\mathcal{N}$.

### 2.4. Performance analysis

The performance analysis is carried out on $\mathcal{N}$, where the metrics annotated in the UML design can be computed. Each metric is mapped onto a function of the throughput of transitions and/or mean marking of places, properly identified through the labeling. The metrics are calculated, on the steady state assumption, by applying well established SWN solution methods (Chiola et al., 1993; Gaeta and Chiola, 1995; Chiola et al., 1997).

The following sections describe in detail each step of the method.

## 3. Construction of a UML model for SWN analysis

This first step of the method is illustrated by using, as running example, a modified version of the gas station system originally presented in Hu and Shatz (2004). The system consists of $N$ customers and four pumps which process the customers' requests for filling the gas.

### 3.1. UML design: the input to the method

The software analyst has to provide the method with a UML design describing the behavior of the system. The UML design consists of:

- a set of SMs, in the example the SM of the customers in Fig. 1a and the SM of the pumps in Fig. 1b;
- a performance scenario, in the example it is represented either by the SD in Fig. 2c or by the IOD in Fig. 2a together with the set of referenced SDs. One of the referenced SDs is shown in Fig. 2b;
- and, optionally, a DD where to annotate the network speed. The DD is not necessary in the example, since the objects do not exchange messages throughout a network.

Therefore, the system functionality is modeled through a set of UML SMs, each one representing the behavior of its class and cooperating with the other SMs by exchanging messages.

A SM is made of states, used to place the *do-activities* defined for the class, and transitions, used to represent message exchange and labeled as *eventRec/class.eventProd*. The SM represents a *reactive* model, i.e., an object is in a state either waiting for an event occurrence or executing a *do-activity*. When the object receives an *eventRec* then it "reacts" by changing state and sending an *eventProd* to the target *class*. If an *eventRec* is received while executing the *do-activity*, then the latter is aborted and the event accepted. A transition is considered immediate if it has not modeled an *eventRec* then, it fires just when the *do-activity* finishes its execution.

In the example, a customer in state *Arriving* performs the activity *arrive*, not computing but spending some time before to send the *ServiceRequest* event to a pump. Then, an *Unused* pump receives the event and answers to the customer with an *OK* event, and then it *Waits for Payment*. The customer *Waiting for availability* performs the activity *count-down* to zero before making a *ServiceRequest* to another pump. If, in the meantime, an *OK* event arrives then the *count-down* activity is aborted, the payment is sent to the pump (*OK/Pump.Pay*) and the customer moves in state *Paid* where it waits until the event *PumpReady* arrives from the pump. After payment, the customer selects the gas grade and presses the nozzle. The pump fills gas and it

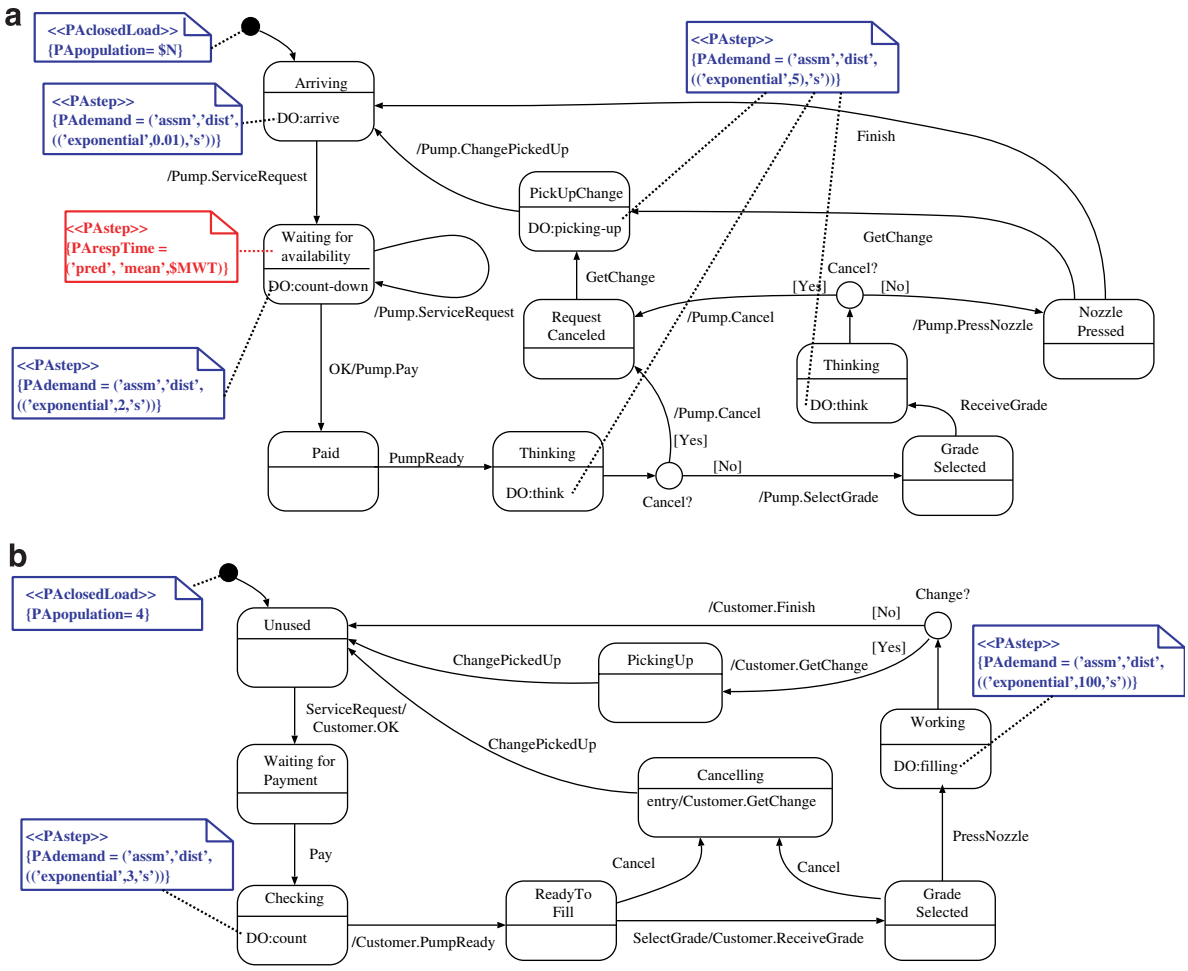


Fig. 1. UML state machines of customers (a) and of pumps (b).

stops either when the pre-paid money is spent or when the tank is full. The pump can prompt the customer to pick up the change. The customer can cancel the operation after the pre-payment or after the selection of the gas grade.

The UML SMs model is augmented with the modeling of one *performance scenario* due to the following reasons:

- Performance engineers feel comfortable analyzing their system under an scenario where performance values can be parameterized.
- An SD captures the *causal relation* among the events exchanged in the system (object control flow). This relation is not explicitly described by the SMs model.

Concerning the SD modeling assumptions, we consider that each lifeline names a generic object *Obj*[*i*] of class *Class*. The behavior of such class has to be modeled by an SM. According to the UML interpretation, any object belonging to the class can execute the lifeline, but obviously when a concrete object (a customer, in Fig. 2c) sends the *ServiceRequest* event to a concrete pump and the latter accepts the event, the rest of the SD is executed by this pair of objects. In terms of the SWN model, it will mean that the identities of the objects participating in the interaction have to be preserved along the whole interaction execution.

By representing only one object per lifeline, a message is characterized by a sender and a receiver; we do not consider broadcast messages since there is not a clear semantics in UML for this concept. Although some proposals have been made, for example by Whittle (2006), they are still not standard.

### 3.2. Performance annotation of the UML design

Being our objective to analyze the system performance, the UML design has to be annotated with *performance characteristics*, i.e., performance input parameters and performance metrics. The UML-SPT profile (OMG-UML-SPT, 2005) gives the syntax to define them through the use of predefined stereotypes and attributes. Appendix A summarizes the UML-SPT approach and the annotations used in the method.

First, we describe the meaning of the *performance input parameters* and how to annotate them in the design.

The system (closed) workload has to be defined by specifying in each SM their population (i.e., the number of

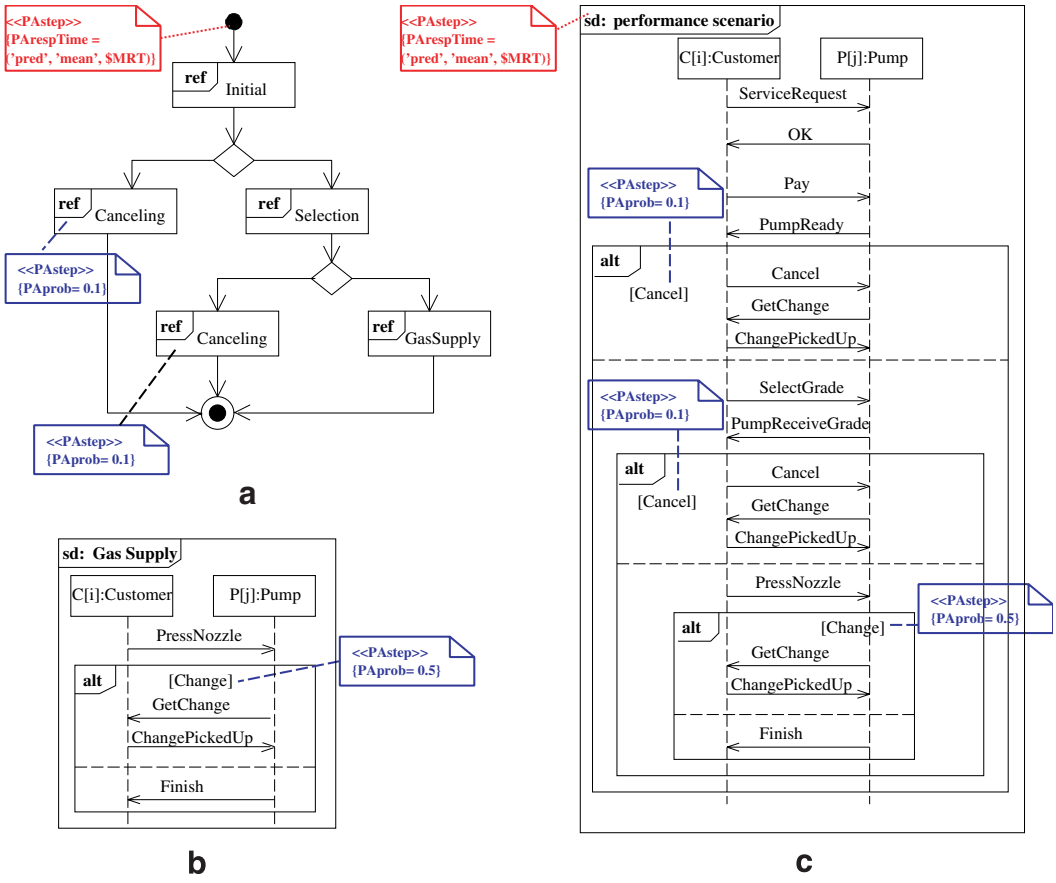


Fig. 2. Alternative representations of the gas system performance scenario: an IOD (a) with the referenced SDs, such as the gas supply interaction (b), or a single SD (c).

objects that will execute concurrently the SM). The UML-SPT provides the *PAclosedLoad* stereotype, that we attach to the SM initial state, and the *PApopulation* attribute. In Fig. 1a and b, a variable $\$N$ and an integer number (4) have been used, respectively, to define the number of customers and pumps in the system.

System activities, modeled inside the states of the SMs as *do-activities*, represent (computation or thinking) time and they are stereotyped as computational *PAsteps*. See, for example, activities *think* in states *Thinking* in Fig. 1a where the *PAdemand* attribute specifies the duration of the activities as random variables exponentially distributed, which are the ones supported by the SWN formalism.

The system routing rates can be modeled either in the SD or in the IOD by assigning probabilities to the interaction constraints, see the *PAprob* attributes in Fig. 2b and c, or to the interaction occurrences, see the *PAprob* attributes in Fig. 2a.

The delay of the messages exchanged among objects allocated in different physical nodes are annotated in the SD. We consider two alternative ways to model such delay:

- If the amount of delay is known, then we use the attribute *PAdemand* to annotate it in the message. See, for example, *PumpReady* in Fig. 4a.
- Otherwise, we use a combination of attributes: *PAsize*, annotated to the message, and *PAspeed* (communication network speed) annotated in the DD. See, for example, Fig. 12.

Concerning the *performance metrics* (annotated in red[3] color in the UML diagrams along the paper), although some of them are specified in the SMs and others in the SD, or in the IOD, all of them will be computed on the same performance model, the one obtained in the third step of the method.

These metrics are defined as *mean* values. They are computed considering the set of class objects that execute, an infinite number of runs, the SMs and the performance scenario (steady state assumption in the SWN).

We have proposed performance scenarios as an input to the design, then it may be of interest to compute their mean execution time, we call it *scenario response time*. The *scenario response time* is annotated as a *PAstep* with attribute *PArespTime* attached either to the initial state of the IOD or to the most external combined fragment of the SD. In the gas system, Fig. 2a and c illustrates both annotations.

[3] For interpretation of color in all figures, the reader is referred to the web version of this article.

The *sojourn time* in a SM's state is the mean time spent by an object in the state, from its entrance to its exit. The *sojourn time* metric is annotated as *PAstep* with attribute *PArespTime* attached to the SM state of interest, in the example, *Waiting for availability* in Fig. 1a.

Finally, the *throughput* of a SM transition measures the number of its firing per unit of time. It is a mean value calculated considering the SM transition executed by all the instances. The *throughput* metric is annotated as *PAstep* with attribute *PAthroughput* attached to the SM transition of interest.

## 4. Translation of the UML model into SWN

This second step of the method takes as input the UML-SPT design, from the first step, to produce one SWN component for each UML SM and one SWN component for the performance scenario.

### 4.1. Translation of annotated state machine

In Merseguer (2003) we proposed a formal translation, for most of the SM features into GSPN, that has been implemented in the ArgoSPE tool (http://argospe.tigris.org). Concretely, we translated the different kind of states (initial, final, simple states, composite states, history, synchronous), actions (entry, exit, do-activity), transitions and events.

The GSPN component, obtained automatically by ArgoSPE, can be provided with color information to gain the SWN component for the SM. For example, the net depicted in Fig. 3, without blue inscriptions, is actually the GSPN produced by ArgoSPE for the SM of the pumps. The sub-nets enclosed in the dotted areas represent the translation of the states together with their outgoing transitions. Net labels are written in italic and net names in roman. The blue inscriptions (in bold font) are the color information that need to be added to the GSPN to get the SWN component for the SM of the pumps. For readability reasons, we do not show all the labels, names and colors information.

An SWN component is obtained from the GSPN component by defining *the object identities*, *the initial state of the objects*, *the arc expressions* and *the color domain of mailbox places*.

*The object identities* are captured by assigning a token of different color to each object. The objects of class *Pump* are then mapped into a basic color class $\mathbf{Pump} = \{\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}, \mathbf{p_4}\}$, where the cardinality of **Pump** is the value of the *PApopulation* tag associated to state *Unused* of the SM. The basic color class is parameterized if, instead, a variable is assigned to the *PApopulation* tag, as for the SM of the customers in Fig. 1a.

The tag provides also information on *the initial state of the objects*, that corresponds to assign, as initial marking of the SWN, one token per color to place *ini_Unused*. Then, the initial marking of place *ini_Unused* is equal to $\langle \mathbf{S\ Pump} \rangle$, that is the symbolic notation of the formal sum $\langle \mathbf{p_1} \rangle + \langle \mathbf{p_2} \rangle + \langle \mathbf{p_3} \rangle + \langle \mathbf{p_4} \rangle$. The current state of an object is represented by the presence of its corresponding token in a place of color domain **Pump**: such places, let us call them *internal*, are drawn in red in the figure. Observe that, each internal place is shared by all the objects of the same class, this makes the SWN model of a SM scalable with respect to the class population.

By definition of SWN, when the same variable appears in many *arc expressions* related to the same transition, the different occurrences actually denote the same object. Then, the same variable name **#x** is assigned to the input and output arcs connecting a SWN transition to a pair of internal places; this choice guarantees the preservation of the identity of the objects during their lifetime.

In the GSPN component, places that are not internal contain event occurrences related to object communication: they represent mailboxes collecting outgoing and incoming messages that are, respectively, sent and received by the objects. See for example in Fig. 3, the mailbox place *e_PumpReady* in the sub-net *Checking*.

In the SWN component, *the color domain of mailbox places* is defined as the Cartesian product of two basic color classes, where the first component identifies the sender class and the second one identifies the receiver class. An event occurrence is then represented by a colored token $\langle \mathbf{s}, \mathbf{r} \rangle$, where **s** represents the sender object and **r** represents the receiver object. For example, the place *e_PumpReady*, with color domain $\mathbf{Pump} \times \mathbf{Customer}$, represents an outgoing mailbox for the pump. Vice-versa the place *e_SelectGrade*, in the sub-net *Ready To Fill*, with color domain $\mathbf{Customer} \times \mathbf{Pump}$, models an incoming mailbox for the pump. The *arc expressions* related to mailbox places ensure that tokens $\langle \mathbf{s}, \mathbf{r} \rangle$: (1) are added to mailbox places *e_ev* due to the generation of an event occurrence by the object of color **s** and, (2) are removed from a mailbox place *e_ev* either due to event consumption performed by the object of color **r** or due to event loss.

Finally, it is worth noticing that ArgoSPE maps the annotated do-activities, like *filling* in sub-net *Working*, into timed transitions whose firing rate is derived from the value of the *PAdemand* tag. More precisely, the latter specifies the expectation value of the duration of the activity, 100 s. for the *filling* activity. The firing rate of the corresponding timed SWN transition is equal to the inverse of expectation value, $\Omega(t_4) = 0.01/\mathrm{s}$. When the *PAdemand* value is characterized by a variable, then the firing rate of the SWN transition is parameterized.

### 4.2. Translation of annotated sequence diagram

The SD is translated into an SWN that preserves the *causal relation* between the events exchanged by the interacting objects. The SWN is obtained through net composition, where the unit of composition is an SWN sub-net modeling one single message.

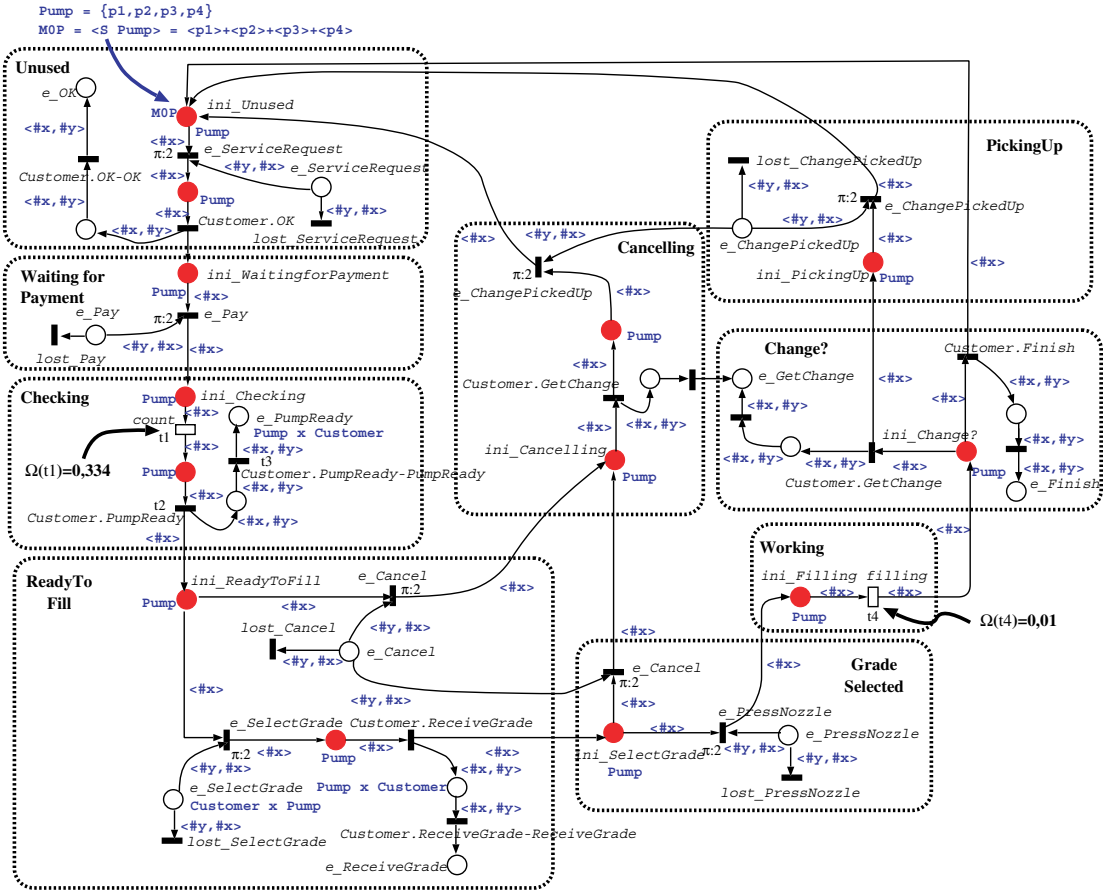


Fig. 3. Component SWN for the pump class.

Let us consider how a single message is translated into an SWN sub-net: Fig. 4a depicts the message *PumpReady* exchanged between two objects of classes pump and customer, respectively. Fig. 4b shows the resulting SWN sub-net, where $t_1$ represents the sending action performed by object $P[j]$, $t_2$ models the message transmission delay, $t_3$ represents the reception of the message by object $C[i]$ and $t_4$ models the message loss.

The value associated to the tag *PAdemand* defines the firing rate of the timed transition, $\Omega(t_2)$. When the transmission delay is modeled using a combination of tags *PAsize* (in the SD) and *PAspeed* (in the DD), then $\Omega(t_2)$ is equal to the ratio between their values.

Transition labels match with the corresponding ones in the SWN models of the SMs, as it can be observed by comparing, for example, the transitions $t_1$ and $t_2$ in Fig. 4b with the transitions $t_2$ and $t_3$, of the sub-net *Checking*, in the SWN model of the *Pump* in Fig. 3.

Places are all of the same type (**Customer** × **Pump**, in the example); their color domain is equal to the Cartesian product of the color classes of the participants, in the order they appear in the SD, from left to right. The arc expressions keep track of the interacting objects by means of the variable names **#x** and **#y**, so matching with the arc expressions of the corresponding transitions in the SWN components of the SMs. The variable **#x** is always assigned

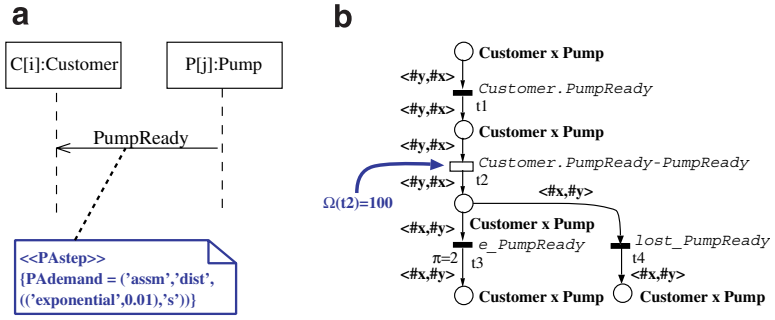


Fig. 4. SWN translation of a message.

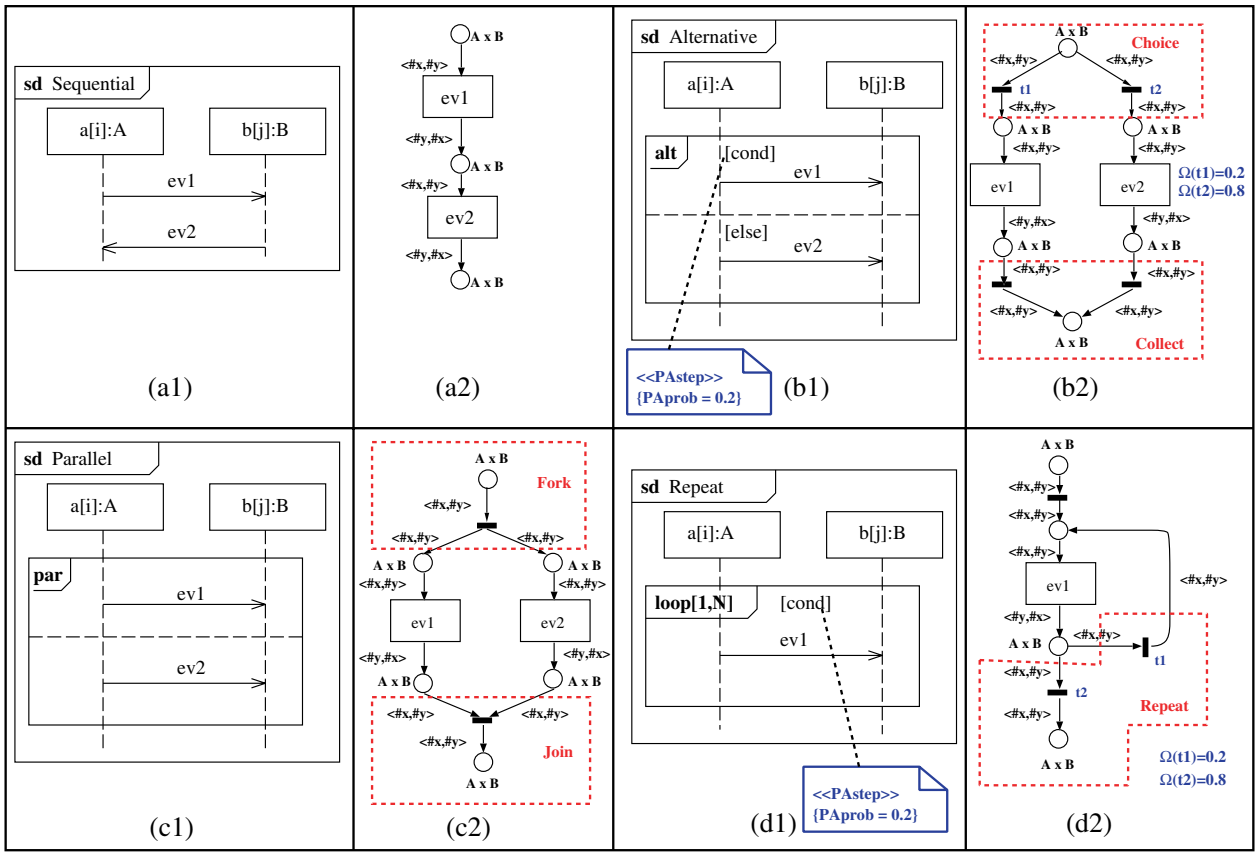


Fig. 5. SWN translation of some basic operators on messages.

to the *active* part in the communication and **#y** is always assigned to the *passive* part. In case of sending action and message transmission, the sender is the active part, while the receiver the passive one. Vice-versa, in case of event consumption, the sender plays the passive role and the receiver plays the active one.

An SD is defined in terms of basic operators on messages. Fig. 5 shows the four main types of SD constructors (sequential, alternative, parallel and loop with "repeat-until" semantics) and their mapping onto SWNs. The sequential operator on messages (a1) corresponds to *causally connect* the SWN sub-nets representing the messages (a2). The translation of the other operators requires the use of additional SWN sub-nets. Fig. 5(b2) shows the SWN sub-net modeling the alternative choice between *ev1* and *ev2*. The additional sub-nets enclosed in the red dotted rectangle *Choice* and *Collect* represent, respectively, the choice between the sending of *ev1* or *ev2*, and the unification of the flow. Note that the choice is probabilistic: the weights of the conflicting transitions $t_1$ and $t_2$ are derived from the tag *PAprob* attached to the constraint *cond*. Fig. 5(c2) depicts the SWN sub-net corresponding to the parallel execution of *ev1* and *ev2*. The two additional sub-nets *Fork* and *Join* represent, respectively, the splitting of the control flow and their subsequent synchronization. Finally, Fig. 5(d2) shows the SWN sub-net modeling a "repeat-until" loop. The sub-net *Repeat* models the itera-

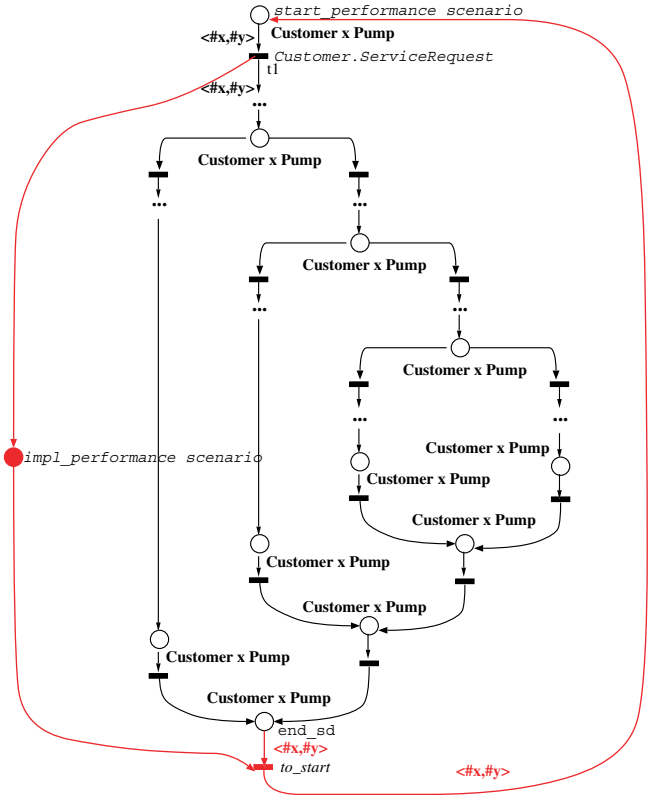


Fig. 6. SWN model of the gas system scenario.

tion of message *ev1*. The loop is probabilistic and the weights of the transitions $t_1$ and $t_2$ are derived from the tag *PAprob* attached to the constraint *cond*.

For metric computation purposes, an implicit neutral place *impl_S* and a transition *to_start* are added to the SWN of a SD *S*. The former keeps track of the number of instances executing the interaction and the latter is introduced to bring the SWN back to its initial state *start_S*. Fig. 6 shows a sketch of the SWN for the SD in Fig. 2c, with the new elements in red.

## 5. Obtention of the performance SWN

The SWNs of the SMs are characterized by interface places, labeled *e_ev*, that represent mailboxes of events *ev*. These SWNs are composed over such interface places to get an intermediate SWN $\mathcal{N}_{\text{SMs}}$ that models the communication between objects via mailboxes.

Fig. 7 shows, as black boxes, the SWNs of the customer and the pump. The figure emphasizes two pairs of interface places that represent two kinds of object interaction: (1) the interaction in which an event *GetChange* is produced by a pump when prompting the customer to pick up the change, and (2) the consequent interaction where an event *ChangePickedUp* is generated by the customer once he/she has picked up the change. For each kind of event, there are two mailbox places with matching labels (one in each SWN): the SWN composition replaces them by a unique place (depicted as dotted red circle) with the same type connecting the two SWNs.

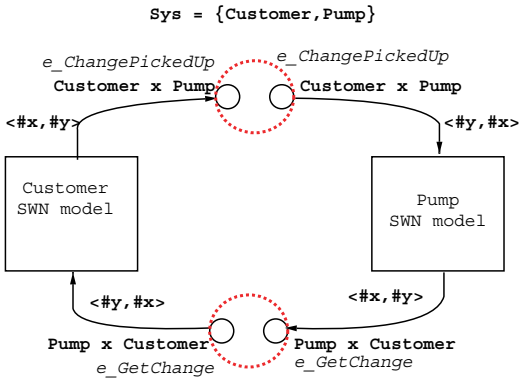


Fig. 7. Composition of Customer and Pump SWN models.

By construction, the $\mathcal{N}_{\text{SMs}}$ model is scalable with respect to the class population and it represents the communication between objects considering their identities. However, it fails to capture the causal relation between events, such as the one between the events *GetChange* and *ChangePickedUp*. In fact, it may happen that the pump *p*[1] generates an event *GetChange* for customer *c*[1] and the latter, once consumed it, answers by producing an event *ChangePickedUp* for another pump, different from *p*[1], which is not the desired behavior.

On the other hand, the SWN model of the SD $\mathcal{N}_{\text{sd}}$, represents the causal relation between events without the objects internal behavior. The composition of $\mathcal{N}_{\text{sd}}$ and $\mathcal{N}_{\text{SMs}}$, over matching label transitions, produces an SWN model $\mathcal{N}$ of the system, that is able to capture both properties.

Let us come back to the example to see how the final SWN $\mathcal{N}$ behaves. Fig. 8 shows the portion of $\mathcal{N}$ related to the exchange of messages *GetChange* and *ChangePickedUp*. The red part comes from the $\mathcal{N}_{\text{sd}}$ model that represents the interaction. Let us suppose that the pump, identified by the token $\langle \mathbf{p1} \rangle$, sends a message *GetChange* to the customer identified by $\langle \mathbf{c1} \rangle$. Then a control token $\langle \mathbf{c1}, \mathbf{p1} \rangle$ is added to place $p_1$. When, after transmission, the message is received by $\langle \mathbf{c1} \rangle$ (firing of transition $t_1$ for the pair $\langle \mathbf{c1}, \mathbf{p1} \rangle$) the control token $\langle \mathbf{c1}, \mathbf{p1} \rangle$ is moved from $p_2$ to $p_3$. The presence of this control token in $p_3$ constraints the object $\langle \mathbf{c1} \rangle$ to send the answer *ChangePickedUp* to pump $\langle \mathbf{p1} \rangle$ (firing of transition $t_2$ for the pair $\langle \mathbf{c1}, \mathbf{p1} \rangle$), then capturing the causality of the two interactions.

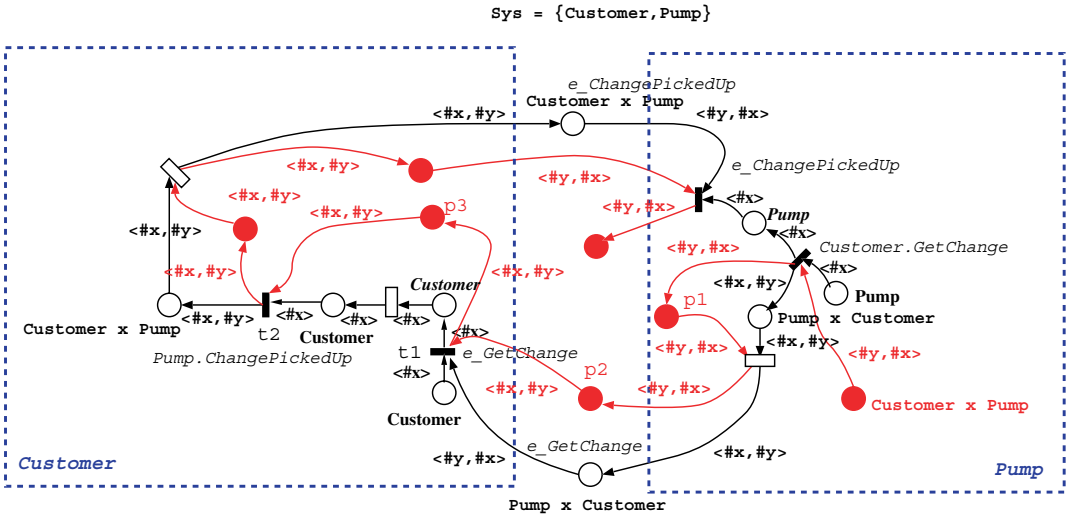


Fig. 8. Partial view of the Customer and Pump SWN model.

### 5.1. Interaction assumptions

An SD $S$ represents a system interaction, but depending on the participating objects, several *concrete* interactions can be interpreted on $S$. The method supports two alternative interpretations, let us call them **A1** and **A2**.

Fig. 9 exemplifies the two assumptions for an interaction between objects of two classes $A$ and $B$ with population $n = 3$ and $m = 2$, respectively. On the assumption **A1** each pair of objects can potentially execute the interaction $S$, then leading to $n \times m = 6$ potential interactions (Fig. 9 on the left).

On the assumption **A2**, a subset of $m$ objects of class $A$ can participate to a concrete interaction $S$ with at most one partner of class $B$. The remaining $n - m$ objects of class $A$ behave as in assumption **A1**, that is each one can potentially execute the interaction $S$ with any object of class $B$. The assumption **A2** leads to $m + (n - m) \times m = 4$ potential interactions (Fig. 9 on the right).

The choice between **A1** and **A2** depends on the system to be analyzed: e.g., in the gas station system both the assumptions are reasonable and they correspond to a different service discipline. Indeed, under **A1**, a customer that arrives to the gas station is not aware of the availability situation of the pumps and then he/she selects probabilistically to be served by a pump; if the pump is available the customer proceeds with the operation, otherwise he/she tries again until he/she succeeds. Under **A2**, customers do not wait for being served when the number of customers does not exceed the number of pumps; indeed, an available pump is always ready and the service requests are processed in parallel.

In the SWN $\mathcal{N}$, each assumption corresponds to assign a different initial marking to the place *start_S*, modeling the beginning of $S$. The set of concrete interactions can be formalized by a relation $\mathscr{R} \subseteq \mathbf{A} \times \mathbf{B}$, where $\mathbf{A}$ and $\mathbf{B}$ are SWN basic color classes associated to the classes $A$ and $B$ respectively, and $\langle \mathbf{a_i}, \mathbf{b_j} \rangle \in \mathscr{R}$ represents a pair of objects that participate in a concrete interaction. Then, the initial marking of the place *start_S* is set to the formal sum of the pairs $\langle \mathbf{a_i}, \mathbf{b_j} \rangle \in \mathscr{R}$:

$$\mathbf{A1}: \quad M0_S = \langle \mathbf{S\ A}, \mathbf{S\ B} \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} \langle \mathbf{a_i}, \mathbf{b_j} \rangle, \tag{1}$$

$$\mathbf{A2}: \quad M0_S = \sum_{i=1}^{m} \langle \mathbf{a_i}, \mathbf{b_i} \rangle + \sum_{i=m+1}^{n} \sum_{j=1}^{m} \langle \mathbf{a_i}, \mathbf{b_j} \rangle. \tag{2}$$

In case of $N$ participant classes, **A1** and **A2** can be straightforward interpreted and $\mathscr{R}$ easily generalized. In particular, under assumption **A1**, $\mathscr{R}$ is equal to the Cartesian product $\prod_{j=1}^{N} \mathbf{C_j}$, where the SWN basic color class $\mathbf{C_j}$ represents the participant class $C_j$.

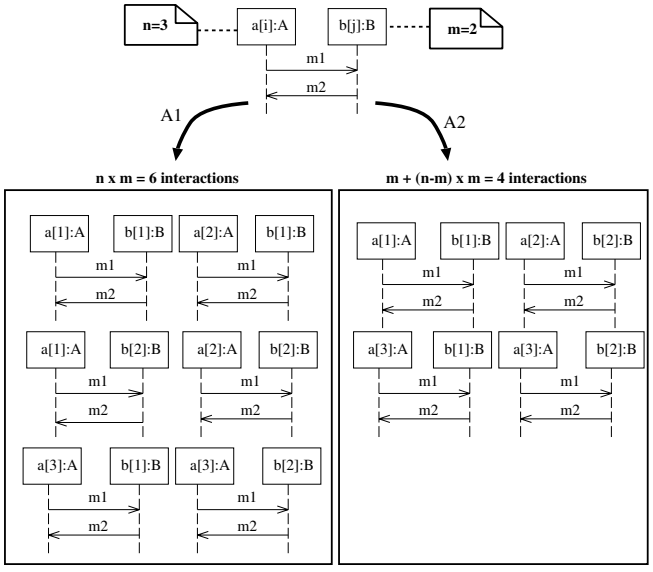


Fig. 9. Concrete interactions according to assumptions **A1** and **A2**.

## 6. Performance analysis

The objective of the performance analysis is to compute the metrics annotated in the UML design and to interpret the values obtained in the system domain. The performance analysis is carried out on the SWN model $\mathcal{N}$, obtained in the third step of the method, where the metrics, annotated in the UML design, are computed. Indeed, each metric is mapped onto an output parameter of the SWN model and all them represent mean values, to be computed under the steady state assumption. They can be of the following types:

- Mean response time of the scenario $S$.
- Mean sojourn time of an object in a given state $A$ of the state machine $M$.
- Throughput of a transition *tr* of the state machine $M$.

These metrics can be computed using the GreatSPN tool (http://www.di.unito.it/~greatspn), then no expertise in SWN modeling and analysis is required. Nevertheless, we consider of interest giving the definition of the metrics in terms of SWN formulas in order to show the traceability of the method. Moreover, the software analysts with expertise in Petri net analysis, once learned how to map such metrics onto SWN output parameters, may undertake the mapping of their own metrics, then improving the usability of the method.

The SWN output parameters corresponding to the first and second types of metrics are defined by using the Little's formula (Lazowska et al., 1984) applied on SWN. In particular, the mean response time of the scenario $S$ is mapped onto an SWN output parameter defined as the ratio between the mean number of tokens in the implicit place *impl_S*, that keeps track of the number of instances executing the interaction, and the throughput of transition *to_ start*, that closes the SWN sub-net of $\mathcal{N}$ representing the scenario $S$ (e.g., see the gas system scenario of Fig. 6).

The definition of the SWN output parameter corresponding to the mean sojourn time in the state $A$ of the state machine $M$ is based on the identification of the SWN sub-net $\mathcal{N}_A$ that represents the state $A$. Such identification is carried out through the labels associated to the places and transitions in $\mathcal{N}$ that match with the names of the elements in $A$.

In the gas example, this metric has been annotated in the state *Waiting for availability* of the customer SM, characterized by the do-activity *count-down*. The SWN sub-net, depicted in the dotted area in Fig. 10 represents the state *Waiting for availability*. It consists of two internal places labeled as *ini_Waiting for availability* and *compl_Waiting for availability*, and the timed transition labeled as *count-down*. The mean sojourn time metric is then defined as $\frac{E[\#p_1]+E[\#p_2]}{X(t_1)+X(t_2)}$, where SWN transitions $t_1$ and $t_2$ may remove tokens from the sub-net. In particular, in the system domain, $t_1$ – labeled *e_OK* – represents the dispatching of the event *OK* and $t_2$ – labeled *Pump.ServiceRequest* – represents the execution of the *ServiceRequest* action.

In general, let $P_A$ be the set of internal places and $T_A$ the set of transitions of $\mathcal{N}_A$, the mean sojourn time in state $A$ is defined as

$$\frac{\sum_{p\in P_A} E[\#p]}{\sum_{t\in P_A^\bullet, t\notin T_A} X(t)},$$

where $E[\#p]$ is the mean number of tokens in place $p \in P_A$. The denominator is the sum of the throughputs $X(t)$ of transitions that have an input place in the set $P_A$ ($t \in P_A^\bullet$) and that do not belong to the sub-net $\mathcal{N}_A$ ($t \notin T_A$).

The throughput of a transition $tr$, of a state machine $M$, can be mapped onto either a SWN transition throughput or a sum of SWN transition throughputs, depending on the presence or absence of trigger events and effect actions.

Let $P_A$ and $P_B$ be the sets of places of the SWN sub-nets representing the source state $A$ and the target state $B$ of $tr$, respectively. The simplest case is when neither trigger event nor action are associated to $tr$; then, the SWN output parameter is defined as the throughput of the SWN transition characterized by an input place in $P_A$ and an output place in $P_B$. When the SM transition $tr$ has a trigger event $ev$, then the SWN output parameter is defined as the sum of the throughputs of the SWN transitions with an input place in $P_A$ and representing the dispatching of the event $ev$:

$$Throu_{tr} = \sum_{t\in P_A^\bullet, \lambda(t)=\ e\_ev} X(t).$$

A similar formula is defined when no event is associated to $tr$ but the latter is characterized by an effect action $act$. In this case the sum of the throughputs is made over the transitions, labeled as $act$, with an input place in $P_A$ and an output place in $P_B$.

The software analyst can compute these metrics by choosing between two main types of SWN solution methods (Chiola et al., 1993; Gaeta and Chiola, 1995; Chiola et al., 1997): numerical, based on the solution of the Markov chain underlying the SWN $\mathcal{N}$, and discrete event simulation. Numerical methods are used to obtain "exact" results, that is results equal to the theoretical values of the corresponding statistical qualifiers, apart from approximation errors. Simulation methods provide results with confidence intervals and, depending on the type of simulation technique adopted, several simulation parameters need to be set before starting the experiment (e.g., confidence level, accuracy). In general, the main drawback of numerical methods is the (time and space) exponential complexity of the state space generation and Markov chain solution. So the choice of the type of method strongly depends on the size of the state space of the SWN.

The GreatSPN tool (http://www.di.unito.it/~greatspn) can deal with both types of solution methods. In particular, concerning numerical techniques, GreatSPN supports the generation of both the ordinary and the symbolic state spaces (Chiola et al., 1997) of an SWN. The symbolic technique can be used only for SWN with symbolic initial marking, this is the case of the assumption **A1**, while it is not for the assumption **A2**.

In the following we describe our experience in the performance analysis of the gas system. It is our intention that the software analysts can gain an insight into facing this step of the method.

### 6.1. Performance analysis of the gas system

We have solved the SWN $\mathcal{N}$, obtained in the third step, with the GreatSPN tool (http://www.di.unito.it/~greatspn) running on a Pentium 4 PC with 2,666 GHz CPU. For a customer population ranging from 1 to 4, we have used numerical techniques to solve $\mathcal{N}$ (with an approximation error of the results equal to $10^{-6}$). On the assumption **A1**, both the ordinary and the symbolic state space construction methods can be applied, while on the **A2** only the ordinary one can be used, due to the (not symbolic) initial marking set to the place *start_S*.

Table 1 shows, on both the assumptions **A1** and **A2**, and for different customer populations $N \in [1,4]$, the size of the state space, the memory space required to store the reachability graph and the time required to solve the SWN.

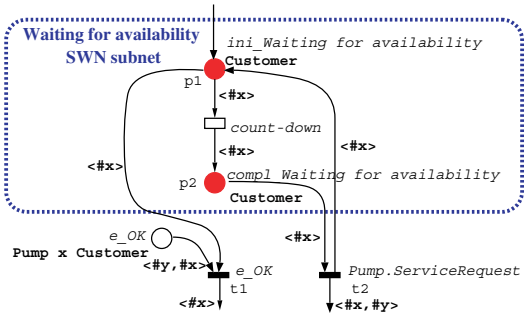


Fig. 10. *Waiting for availability* SWN sub-net.

Table 1
Performance of the numerical techniques on the gas system example

| $N$ | **A1** | | | | | | **A2** | | |
|---|---|---|---|---|---|---|---|---|---|
| | Symbolic | Memory | Time | Ordinary | Memory | Time | Ordinary | Memory | Time |
| 1 | 46 | 15.6KB | 10 ms | 178 | 42KB | 20 ms | 46 | 9.2KB | 7 ms |
| 2 | 518 | 287KB | 30 ms | 10,313 | 3.6MB | 1 s | 891 | 242KB | 15 ms |
| 3 | 3290 | 2.7MB | 11 s | 342,940 | 155MB | 1 m:45 s | 14,036 | 4.6MB | 3 s |
| 4 | 15,247 | 16MB | 9 m:59 s | 6,485,707 | 2GB | 6 h:48 m:20 s | 200,981 | 74MB | 2 m:44 s |

Comparing the size of the state spaces on the assumption **A1** (second and fifth columns), we can observe that the symbolic technique reduces drastically the size of the state space with respect to the ordinary one, then reducing the memory space required to store the reachability graph (third and sixth columns) as well as the time required to solve the SWN (fourth and seventh columns).

Comparing, instead, the size of the ordinary state spaces on the assumptions **A1** and **A2** (fifth and eighth columns), we find that the results on the **A2** are smaller than on the **A1**. This is due to the difference between the number of possible concrete interactions under **A1** and under **A2**. For example, when $N = 4$, they are equal to 16, under **A1**, and equal to 4, under **A2**. When the number of customers is higher than four, the batch simulation technique has been adopted, with confidence level 99% and accuracy $10^{-2}$, to obtain results within a reasonable time. Indeed, the mean time used by GreatSPN to solve the SWN models for $N \in [5, 10]$ has been equal to 1 min and 13 s.

Fig. 11a plots the curves of the mean response time of the performance scenario in Fig. 2c (metric *MRT* in the design), under both the assumptions **A1** and **A2**, for $N \in [1, 4]$. Under **A2**, the *MRT* is constant (equal to 96.47 s), since the customers arriving at the gas station find always an available pump. Under **A1**, instead, the *MRT* increases as the number of customers increases, since there may be the case in which there are available pumps but the customer chooses a pump that is not available, then waiting in a queue.

Fig. 11b plots the curves of the mean time spent by a customer waiting for an available pump (metric *MWT* in the design) on both assumptions. When $N \in [1, 4]$, the gas system behavior slightly differs under the two assumptions: while under **A2**, the *MWT* is zero, under **A1** it is greater than zero when there are more than one customer and reaches 1.46 s for $N = 4$. The two curves are characterized by the same trend when $N > 4$. In particular, they tend to the horizontal asymptote $y(x) = 2$. Indeed, when the number of customers at the gas station grows, the termination of the *count-down* do-activity, without interruptions due to the reception of an *OK* event, becomes more likely. Then, the mean sojourn time in the *Waiting for availability* state becomes equal to the mean duration of the do-activity.

## 7. Case study: software retrieval system

A software retrieval system, like the web site Tucows.com (http://www.tucows.com), provides Internet users with facilities to retrieve and install software. The work in Merseguer et al. (2003) models and analyzes a Tucows-like system. Here, we recall it to illustrate the method: we have updated its specification to UML2, annotated it with the UML-SPT, and obtained an analyzable SWN by applying the steps of the method.

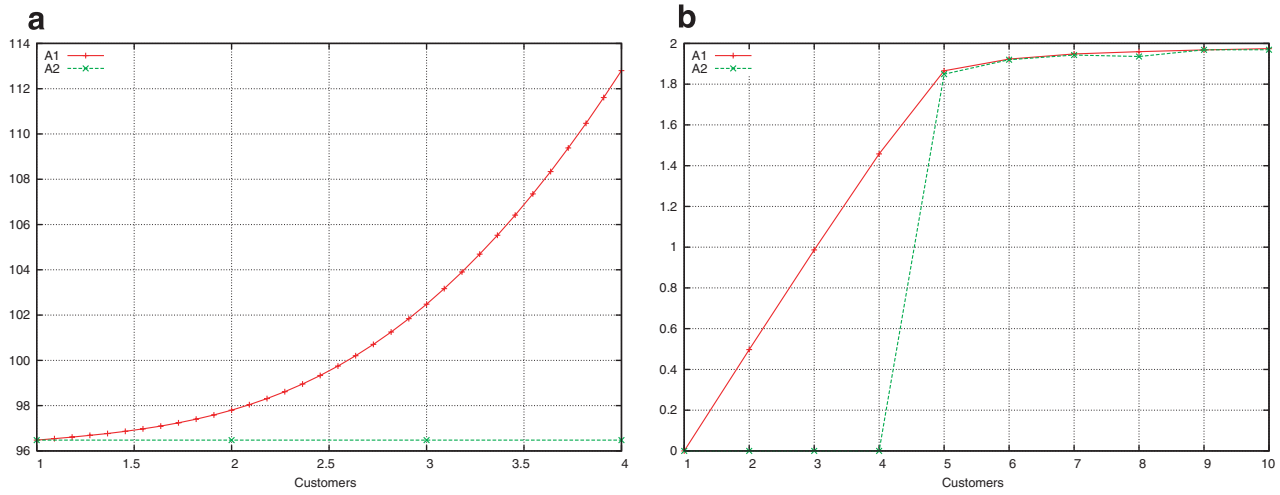


Fig. 11. (a) Mean scenario response time – seconds – and (b) mean sojourn time in state *Waiting for availability* – seconds.

### 7.1. Construction of a UML model for SWN analysis (step 1)

The system allows users to find software by navigating through categories of software programs especially designed to make this task easier. The SD in Fig. 12a illustrates how the system works. The user "clicks" on a browser's category, who requests to the web server for the corresponding HTML page. The web server returns the HTML page to the browser, which presents it to the user. After reading this page, the user can "click" on another link to access either a new web page with other categories or a list of software for the current category. This process is repeated until the user finds a software that fulfills her/his needs. Then the browser requests the selected software, which is downloaded into the user computer.

The system behavior is completed with the UML SMs, of the interacting classes, in Fig. 13: the user (a), the browser (b) and the web server (c). The user aims at installing some software in her/his computer. Initially, she/he is in the *Idle* state and sends the *select_category* event to the browser, so moving in state *Waiting for html page*. The *observe* event, generated by the browser, allows the user to *examine* the HTML page with the available software (state *Examining*). The user can choose to either select another category, then coming back to the state *Waiting for html page*, or to select the desired software, moving in the state *Waiting for download*.

The browser interfaces with the user and the web server to help in the software selection as well as in its downloading. It is waiting for user's requests (*select_category* and *select_sw*). The selection of a category is concurrently processed in the local node by the browser and in the remote node for the web server. The software selection is transformed into a download for the corresponding software by the web server. The web server accepts requests for *select* and *download* the software. For each request, it executes the proper do-activity to serve it, *find_html_page* and *find_file*. While serving a request, other requests coming from other browsers may arrive: in this case, they are deferred until the current activity is not completed. When the activity is completed, the web server sends the corre-

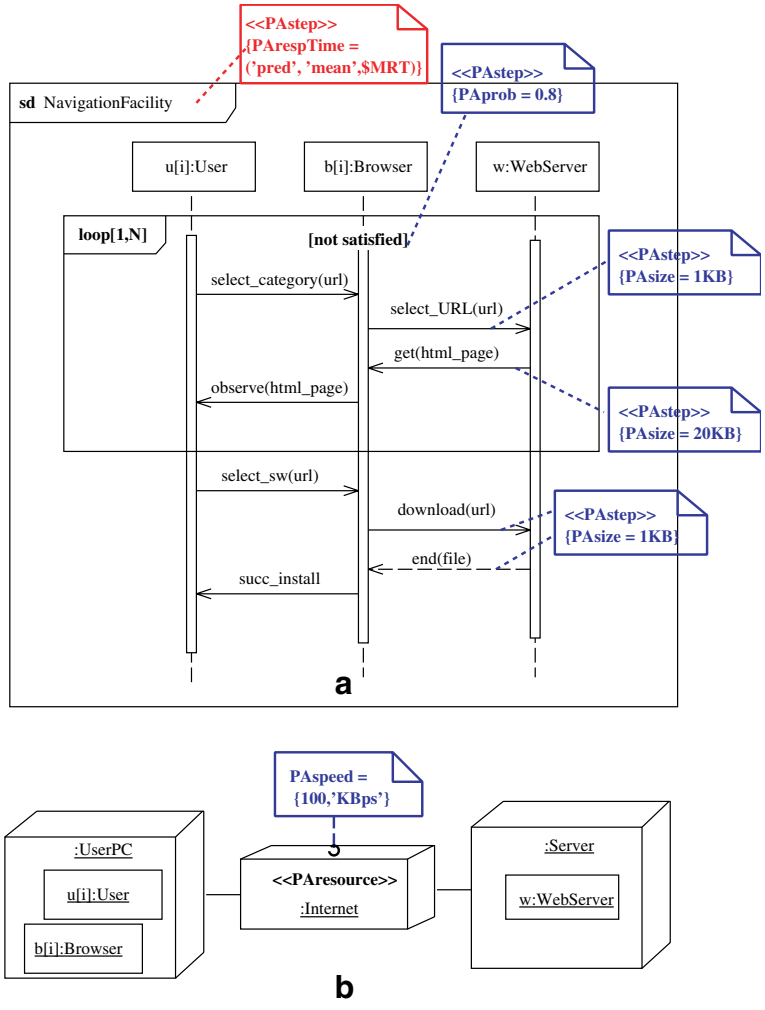


Fig. 12. Navigation facility scenario (a) and system architecture (b).

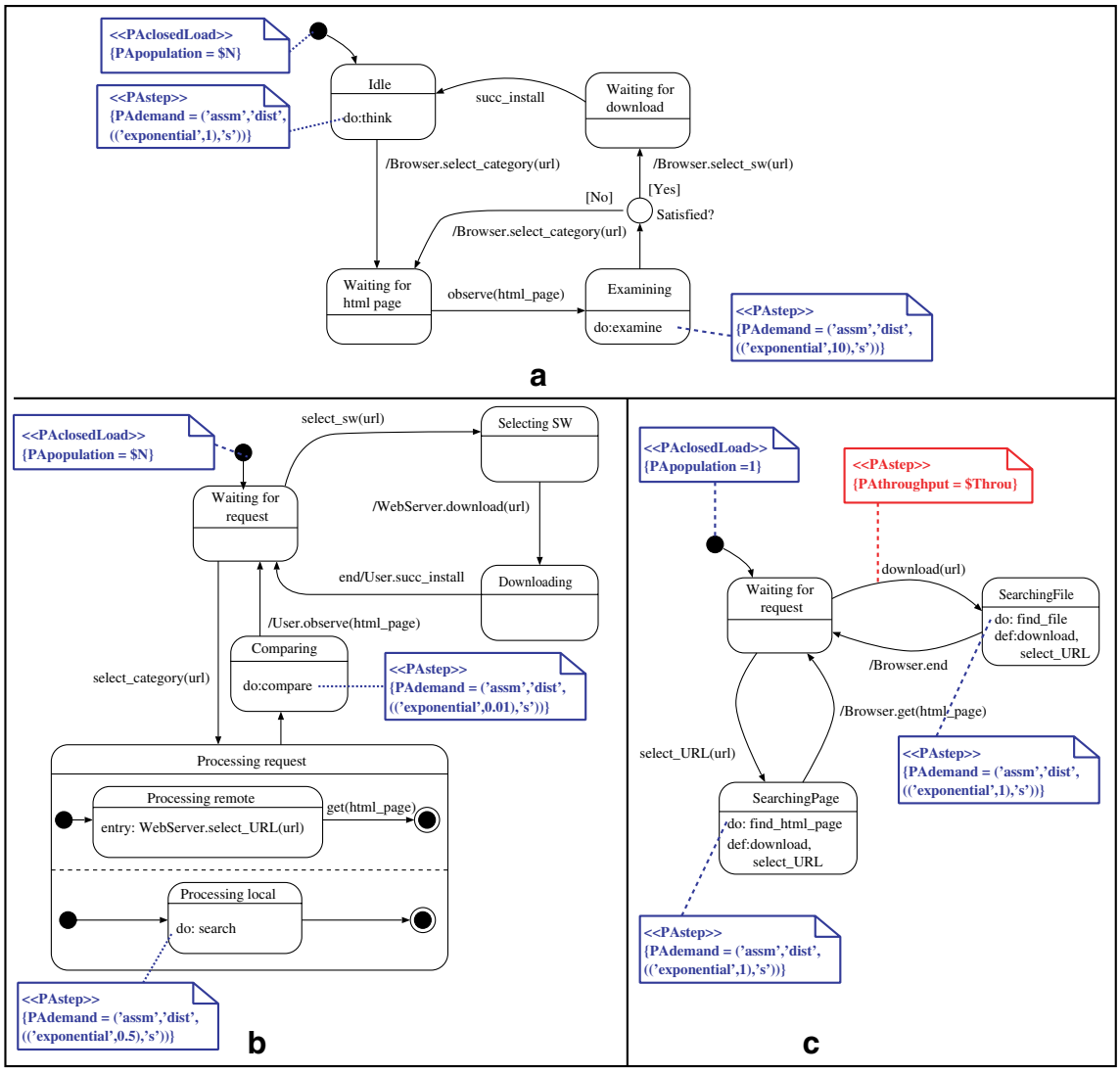


Fig. 13. SMs of the user (a), browser (b) and web server (c).

sponding message to the browser and comes back to its initial state where the deferred requests can be handled as normal events.

In Fig. 12b, the DD models the system architecture. Messages exchanged between the browsers and the web server travel through the Internet, then requiring a certain amount of transmission time. Such time is calculated as the ratio between the network speed (tag *PAspeed* annotated to the node *Internet* in the DD) and their size (tag *PAsize* attached to the messages in the SD). Delays associated to the messages exchanged between the user and the browser are instead assumed negligible.

The rest of the *performance input parameters* are annotated in the SD and the SMs. The system routing rate, annotated in the SD to the interaction constraint *not satisfied* with the tag *PAprob*, models the decision made by the user about to select a category. The class populations, annotated to the initial states of the SMs with the tag *PAclosedLoad*, model the number of users and browsers executing the system (parameter variable $*N*) and the unique web server.

The *performance metrics* of interest are annotated in the performance scenario – Fig. 12a – and in the SM of the web server – Fig. 13c. Concretely, the *scenario response time* and the *throughput* of the transition *download(url)*, that in the problem domain are interpreted as (1) the mean time required to find the software and to download it and (2) the mean number of downloading per seconds, respectively.

### 7.2. Translation of the UML model into SWN (step 2)

We modeled the UML SMs of the user, browser and web server with the ArgoSPE tool, that yielded the GSPNs according to the work in Bernardi et al. (2002). ArgoSPE produces the GSPNs in the file format of the GreatSPN tool, so we use the GreatSPN graphical editor to update them into the component SWNs, following the indication

of the second step of the method, described in Section 4.1. Fig. 14 depicts the user's component SWN, which is used to illustrate such syntactical updating.

- We first define the basic color class of the *User* as **User** = **N**, where **N** is the variable defined in the attribute *PApopulation*. *The object identities* are implicit in this color definition.
- *The initial state of the objects* corresponds with the definition of the initial marking $\mathbf{M0} = \langle \mathbf{S\ User} \rangle$, assigned to place *ini_Idle*. The color domain of the internal places, in red color, is equal to the basic color class **User**. The tokens contained in the internal places represent user objects and the *arc expressions* related to the internal places are defined using the same variable **#x** to guarantee the preservation of the user object identities.
- *The mailbox places* labeled *e_observe* and *e_succ_install*, contain the event occurrences that users receive: then the color domain $\mathbf{Browser} \times \mathbf{User}$ has been associated to them. The other mailbox places, e.g., the places labeled *select_category* and *select_sw*, contain the event occurrences that users generate and, then, their color domain is defined as $\mathbf{User} \times \mathbf{Browser}$.
- Finally, we defined the *arc expressions* related to the mailbox places as either $\langle \#\mathbf{x}, \#\mathbf{y} \rangle$ or $\langle \#\mathbf{y}, \#\mathbf{x} \rangle$ to ensure the preservation of the object identities in the communication.

The performance scenario represented by the SD in Fig. 12a was converted into an SWN following the steps given in Section 4.2. The SWN model $\mathcal{N}_{sd}$, in Fig. 15 bottom-right, was edited with GreatSPN. It is quite easy to get since it consists in the translation of each single message, according to Fig. 4, and causally connect them, except for the loop operator that implies to apply the translation proposed in Fig. 5(d2).

### 7.3. Obtention of the performance SWN (step 3)

The SWN $\mathcal{N}_{SMs}$ was obtained, first, by composing the SWNs of the SMs over interface places. In the case of the user SWN the interface places are the ones labeled as *e_observe*, *e_succ_install*, *e_select_sw*, *e_select_category*. The composition has been performed automatically, by using the algebra tool (Bernardi et al., 2001) that composes SWNs in GreatSPN format just indicating the net names and the place labels.

The algebra tool was also used to compose $\mathcal{N}_{SMs}$ with $\mathcal{N}_{sd}$ over interface transitions, then producing the performance SWN $\mathcal{N}$. In this case, the transition labels provided to algebra were those that match in $\mathcal{N}_{SMs}$ and in $\mathcal{N}_{sd}$. Considering the user SWN, the matching labels are *Browser.select_category*, *Browser.select_category-select_category*, *Browser.select_sw*, *Browser.select_sw-select_sw*, *e_observe*, *lost_observe*, *e_succ_install*, *lost_succ_install*. The SWN $\mathcal{N}$ is depicted in Fig. 15, where we can identify the sub-nets representing the SMs and the performance scenario. To improve visibility, the algebra tool allows the user to hide the arcs connecting the sub-nets.

#### 7.3.1. Interaction assumption

The performance scenario represented in Fig. 12a is executed by $N$ users, each one using its own browser. This interpretation corresponds to the interaction assumption **A2**, where there are $N$ concrete user–browser interactions, executed in parallel, and all them using the (unique) web server.

The initial marking of the place *start_NavigationFacility* (the red place in the *NavigationFacility* sub-net in Fig. 15), representing the beginning of the interaction, is then defined according to the formula (2):

$$\mathbf{M0}_{\mathbf{NavigationFacility}} = \sum_{i=1}^{N} \langle \mathbf{u_i}, \mathbf{b_i}, \mathbf{w} \rangle.$$

### 7.4. Performance analysis (step 4)

We have used the SWN model $\mathcal{N}$ to analyze, with the GreatSPN tool, the behavior of the software retrieval system under different workload assumptions, considering a user population ranging between 1 and 60. In particular,

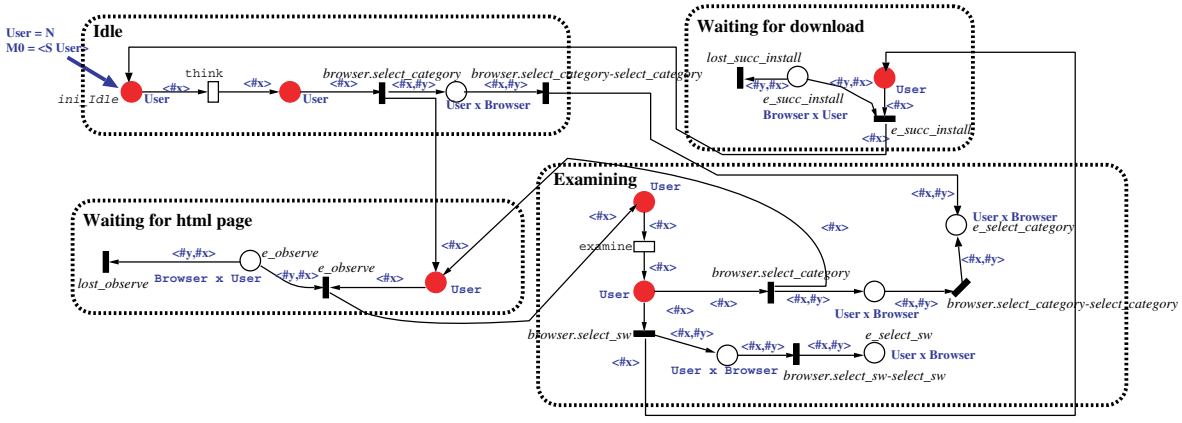


Fig. 14. Component SWN for the user class.

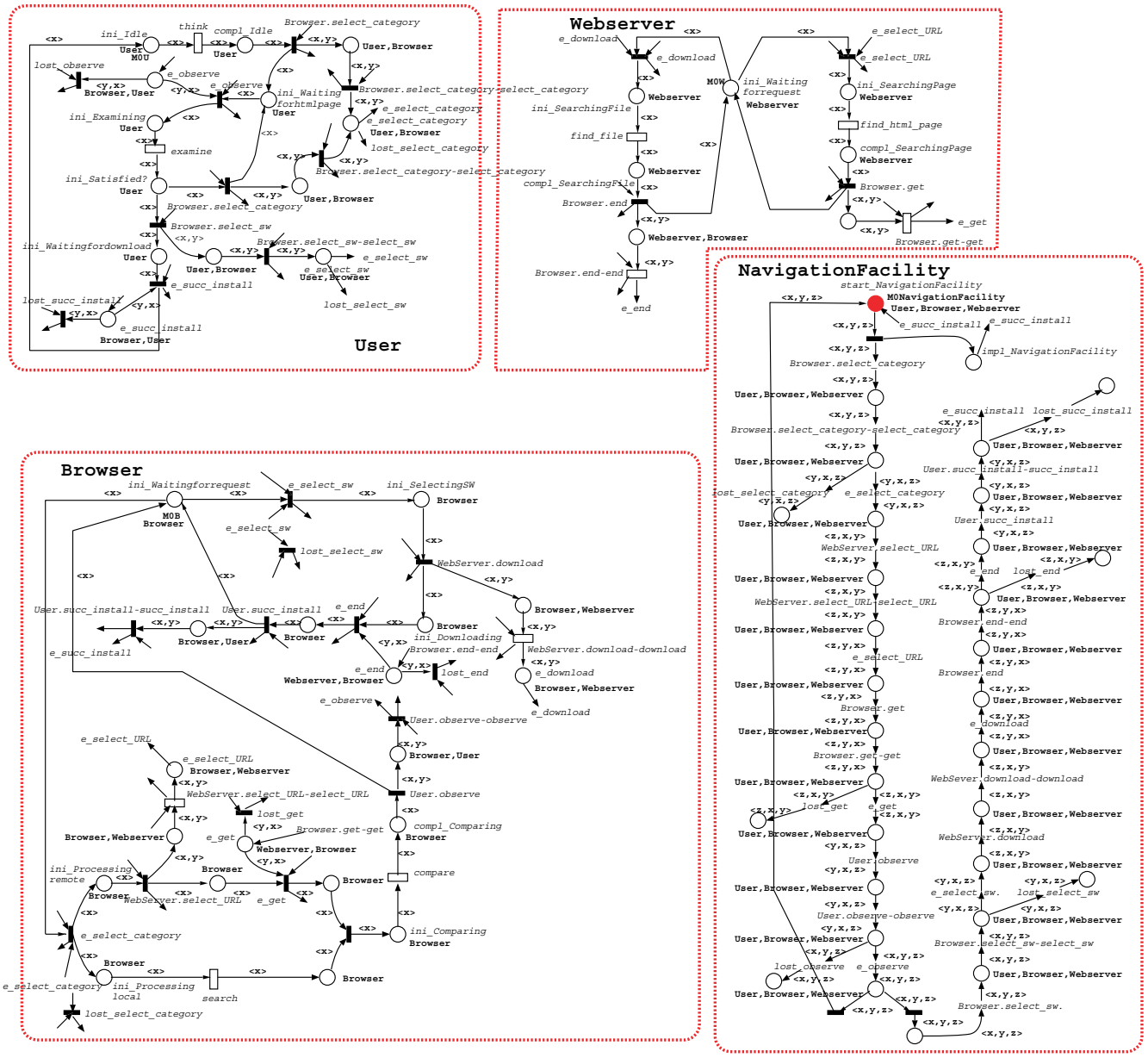


Fig. 15. SWN of the software retrieval system.

the metrics of interest are the mean time to find the software and to download it, and the throughput of the download request. Such metrics correspond, respectively, to the scenario mean response time ($\$MRT$) and the throughput of the web server SM transition *download* ($\$Throu$) in the UML design.

For a population of at most 5 users, we have used numerical techniques (with an approximation error of $10^{-6}$) to solve $\mathcal{N}$. Since the instantiation of the scenario has been carried out under the assumption **A2**, the only possibility is the use of the solution technique based on the construction of the ordinary state space.

Table 2 shows the data collected during the analysis of $\mathcal{N}$, for different user populations. In particular, the size of the state space, the memory space required to store the reachability graph and the time required to solve the SWN when the numerical technique has been applied. The memory space and the time grows exponentially when increasing the number of users, then simulation is the only

Table 2
Performance of the solution techniques

| $N$ | State space size | Memory | Time |
|---|---|---|---|
| 1 | 47 | 12KB | 10 ms |
| 2 | 1129 | 385KB | 20 ms |
| 3 | 23,228 | 9MB | 20 s |
| 4 | 447,685 | 215MB | 5 m:39 s |
| 5 | 8,311,586 | 2TB | 8 h:49 m:41 s |
| 6–60 | [28 s,1 m:21 s], mean time: 50.17 s | | |

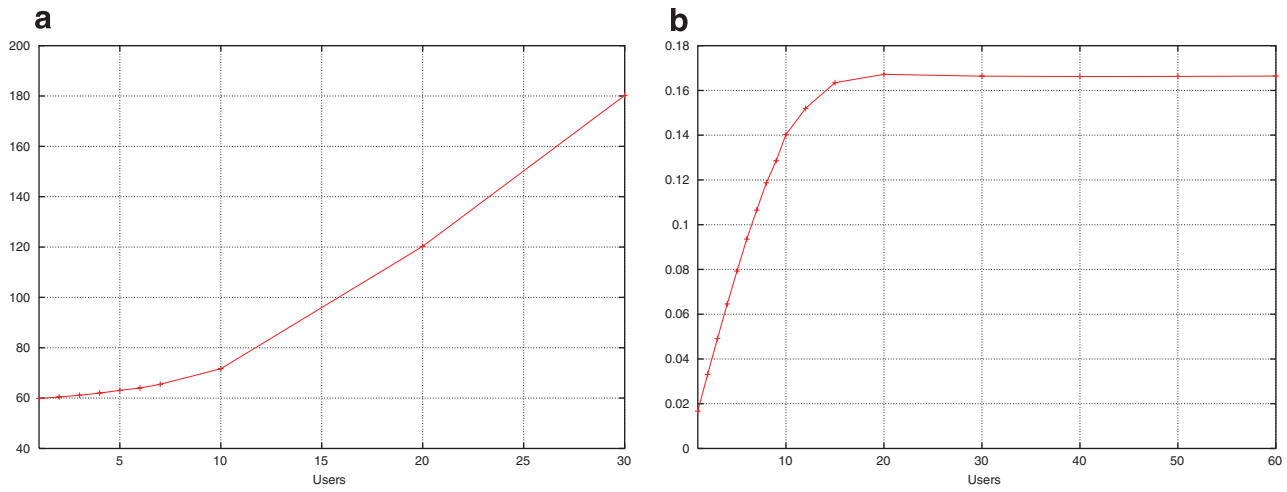


Fig. 16. (a) Mean time to find and download the software (seconds). (b) Throughput of SM transition *download* (1/seconds).

possible available technique when $N > 5$. We have used the batch simulation, with confidence level 99% and accuracy $10^{-2}$. The last line of the table, shows the minimum, maximum and mean times required to solve the model for a user population ranging from 6 to 60.

Fig. 16a plots the curve of the scenario mean response time versus the number of users. Fig. 16b plots, instead, the curve of the throughput of the SM transition *download* versus the number of users.

When the number of users is greater than 20, *MRT* increases with a constant factor and the throughput of the SM transition *download*, in the web server SM, becomes constant (equal to 0.1662/s) indicating that the web server acts as the bottleneck software resource.

## 8. Conclusion

In this paper, we have presented a method for the evaluation of performance requirements in software systems. This method, based on the works (Merseguer et al., 2002; Bernardi et al., 2002) where we proposed to convert UML models into GSPNs, provides software engineers with the ability to compute a set of predefined metrics (*sojourn time*, *throughput and response time*) from the UML-SPT design. The method offers desirable properties such as traceability and object-oriented suitability, and an important advance introduced by the SWN formalism: the capability to analyze, with scalability, systems where the identities of the objects are relevant.

Currently, all the steps of the method but one have tool support: ArgoSPE (http://argospe.tigris.org), algebra (Bernardi et al., 2001) and GreatSPN (http://www.di.unito.it/~greatspn). The remaining step, i.e., the automatic conversion of the GSPN into an SWN, is being now addressed in ArgoSPE. When ArgoSPE implements such conversion, it will compute automatically the metrics for the SWN since now it does for the GSPN, then the whole method will be automatic.

The method does not consider hardware resource contention, then relaying on the *infinite hardware resource* assumption. As a future work, we plan to extend the method with a step that will produce automatically a resource model. Now, software engineers with certain expertise in the Petri net domain could introduce manually in the SWN the places and tokens that represent hardware resources, then gaining an SWN that relaxes this assumption.

The more metrics provided the more useful the method turns. Therefore, it is a must to add new metrics and their computation should be carried out on assuming a transient period or steady state. Currently, the method only supports steady state assumption, nevertheless typical transient measure need to be evaluated such as probability distribution function of a scenario execution.

The most important and challenging future work is to improve the method with a final step addressing *performance assessment*. Indeed, the performance analysis results (from the fourth step) can provide feedback to assess the UML design, then enabling the method to pinpoint the performance bottlenecks. As an example, the software analyst could obtain information for redesigning the concurrency, the deployment or the system workload to meet the desired performance metrics. Moreover, it could be very useful to provide tool assistance to automate this work.

## Appendix A. Performance annotated UML design models

In this appendix, we introduce the annotation approach used by the method to enrich a UML design with performance characteristics, that conforms to the standard Schedulability, Performance and Time UML profile (UML-SPT) (OMG-UML-SPT, 2005).

The UML-SPT is partitioned into sub-profiles defining specific quantitative aspects of software systems. In particular, the *Performance Modeling* sub-profile is addressed to

the performance analysis of UML models and supports the computation of performance indices from a *scenario* point of view. The sub-profile provides a straightforward approach to the annotation of sequence, communication, activity and deployment diagrams. A set of UML *stereotypes* and related tags are defined to characterize the proper diagram elements from a quantitative point of view. Fig. A.1 depicts an example of annotation, where a message is stereotyped as a computation step and its transmission delay is specified by the *PAdemand* tag. However, neither UML state machine (UML SM) nor Interaction Overview diagrams are explicitly considered in the sub-profile.

We have used the annotation approach of the *Performance Modeling* sub-profile by identifying the subset of stereotypes and tags that can be naturally mapped onto input/output parameters of the SWN models. We have extended the usage of some UML-SPT tags to support the performance annotation of the UML SM, according to Merseguer and Campos (2003), and of the UML2.0 Interaction Overview diagrams. We have also added new tags (*PAsize* and *PAspeed*) to existing UML-SPT stereotypes (*PAstep* and *PAresource*, respectively) to allow the user to specify the size of data transmitted via message exchange and the speed of the communication network. Table A.1 shows the subset of annotations used in this work. The first and second columns indicate the stereotype and its tags. The third and fourth columns list the model element (i.e., UML meta-classes) and the UML diagram where the extension is applied. Finally, the last column provides indication on the typical usage of the tag in the performance field.

Each tag in Table A.1 is defined with a type, Table A.2 shows those used in this work, that can be: (a) primitive, such as integer or real; (b) a pair (primitive type, string), where the string describes either a rate or a size unit, such as 'Kbps', 'MB'; (c) complex (*PAperfValue*). Complex type values are specified according to the following format:

$$(\langle <\texttt{source-modifier}\rangle, \langle \texttt{type-modifier}\rangle, \langle \texttt{time-value}\rangle)$$

where the source modifier indicates the origin of the value, e.g., a system requirement (*req*), an assumed input parameter (*assm*), a performance metric to be computed (*pred*). The type modifier defines the statistical meaning, e.g., a mean value (*mean*), a distribution (*dist*). Finally, the time value can be a primitive type value or an expression ((`dist-type,value`), `time-unit`). The transmission delay of the message in Fig. A.1 is an input parameter, characterized by a random variable of an exponential distribution function, with mean 0.01 s. A primitive type value can be a Perl-like expression, even may contain variable names prefixed by the dollar symbol ($). The last column in Table A.2 relates the tag values with the SWN parameters affected in the translation proposed in this work.

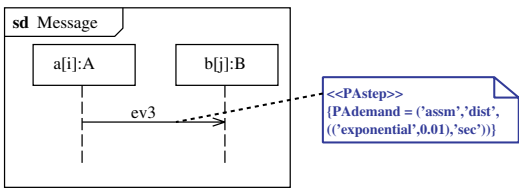


Fig. A.1. UML-SPT annotation.

## Appendix B. Stochastic Well-formed Nets and SWN composition

In this appendix, we introduce the basic concepts of Stochastic Well-formed Net (SWN) and its composition operators. A complete and formal definition of SWN can be found in Chiola et al. (1993), while reader interested in SWN compositional features can see the work (Bernardi et al., 2001).

### B.1. SWN basics

A Stochastic Well-formed Net (SWN) is a high-level Petri net $\mathcal{N} = \langle P, T, C, \mathscr{D}, W^-, W^+, W^h, \Phi, \Pi, \Omega, M_0\rangle$, where $P$ is the set of places, $T$ is the set of transitions, $C = \{C_1, \ldots, C_n\}$ is the set of basic color classes. Basic color classes are finite and disjoint sets, and each class $C_i$ can be partitioned into several static (disjoint) subclasses $C_i = C_i^1 \cup \cdots \cup C_i^{K_i}$ when it is necessary to make a distinction among groups of colors of the class.

Fig. B.1 shows the SWN derived from the SM of the pumps in Fig. 1b.

Place color domains and variable names of the arc expressions are written in bold fonts. The SWN is characterized by two basic color classes **Customer** and **Pump**, the former is only declared while latter is defined as a unique static subclass.

$\mathscr{D}$ is a function that associates a color domain to each place and transition of the net. Color domains are expressed as Cartesian product of basic color classes (repetition of the same class is allowed): tokens in a place $p \in P$ incorporate information and they can be seen as instances of a data structure whose type is the color domain of $p$. In Fig. B.1, places with color domain **Pump** contain tokens representing the pumps together with their identities. Places with color domain **Pump** $\times$ **Customer** contain, instead, tokens modeling messages sent by pump objects: each message is represented by a pair of colors, where the first one is associated to the pump that sends the message and the second one is associated to the customer that receives it.

SWN transitions can be considered as procedures with formal parameters, where the latter range in the *transition color domain*: the classes in the color domain define the types associated with the transition parameters. The color domain of $t \in T$ is implicitly defined by the color domains of its input, output and inhibitor places, and the relation between transition and place color domains is defined through the input, output and inhibitor arc functions

Table A.1
Performance annotations

| Stereotype | Tag | Meta-class | Diag. | Performance concept |
|---|---|---|---|---|
| PAclosedLoad | PApopulation | Initial PseudoState | SM | System closed load |
| PAstep | PAdemand | Message | SD | Transmission delay |
| | | Action(doActivity) | SM | Activity duration |
| | PAsize | Message | SD | Message size |
| | PAprob | InteractionOperand, Int.Constraint | SD | Routing probability |
| | | (inline)Interaction, Int.Occurrence | IOD | Routing probability |
| | PArespTime | State | SM | State sojourn time |
| | | Interaction | SD | Scenario response time |
| | | ActivityInitialNode | IOD | Scenario response time |
| | PAthroughput | Transition | SM | Transition throughput |
| PAresource | PAspeed | Node | DD | Network processing rate |

SD = sequence diagram, IOD = interaction overview diagram, SM = state machine, DD = deployment diagram.

Table A.2
Specification of performance values

| Tag | Type | Type value (expression) | SWN parameter |
|---|---|---|---|
| PApopulation | Integer | $n \in \mathbb{N}$; \$Nuser | Initial marking |
| PAprob | Real | $p \in [0,1]$; \$Prob | Trans. weight |
| PAthroughput | Real | $p \in \mathbb{R}$; \$Throu | Trans. throughput |
| PAspeed | (Real,String) | ($r \in \mathbb{R}$, rateUnit); (\$rate,'Kbps') | Trans. rate |
| PAsize | (Real,String) | ($s \in \mathbb{R}$, sizeUnit); (\$size,'Mb') | Trans. rate |
| PArespTime | PAperfValue | (*pred*, *mean*, \$respTime) | Trans.thr.,pl.marking |
| PAdemand | PAperfValue | (*assm*, *dist*, ((*exponential*, $\delta \in \mathbb{R}$), timeUnit)) | Trans. rate |

$W^-$, $W^+$, $W^h$. A transition $t$ whose formal parameters have been instantiated to actual values is called *transition instance*, denoted as $[t, c]$, where the assignment $c$ is a color *tuple* belonging to the transition color domain of $t$. Only transition instances can fire and their enabling and firing depend on the expression of the arcs connected to the transitions.

An arc expression is a sum of weighted tuples of elementary functions defined on the basic color classes. The simplest elementary function is the *projection* one, used in Fig. B.1, that can be used to select one element of a transition instance color tuple. The variables used for specifying the function can be chosen arbitrarily, e.g., $x$, $y$.

Observe that, when the same variable appears in many arc expressions related to the same transition, the different occurrences actually denote the same object. On the other hand, if the same variable is used in several arc expressions, each related to different transitions, there is no relation between the objects represented by the different variable occurrences.

$\Phi$ is a function that associates to each transition $t \in T$ a guard expression: guards are used to restrict the set of admissible color instances of a transition to those satisfying a given predicate. A predicate is expressed in terms of *standard predicates* and it is a boolean expression. By default, $\Phi(t) = true$ is assumed.

$\Pi$ is the priority function that assigns a priority level to each transition. *Timed* transitions are graphically represented by white tick boxes, such as transition *filling* in Fig. B.1, and they are characterized by zero priority. Priority levels greater than zero are reserved, instead, for immediate transitions, graphically represented as black thin boxes. Conflicting immediate transitions *e_ServiceRequest* and *lost_ServiceRequest* are characterized, for example, by different priorities: the former has higher priority (equal to 2) with respect to the latter (default priority, equal to 1).

$\Omega$ is a function that associates to each timed transition a firing rate, that is the parameter of the exponential probability distribution function characterizing the random firing delay of the transition, and to each immediate transition a weight. Transition weights are used for the probabilistic resolution of conflicts among immediate transitions with the same priority.

Finally, $M_0$ is the initial marking function that assigns to each place either a multi-set over its color domain or a parameter. In Fig. B.1, an initial marking parameter **M0P** is assigned to place *ini_Unused*. **M0P** is equal to the symbolic marking $\langle \mathbf{S\ Pump} \rangle$, which corresponds to the formal sum $\langle \mathbf{p_1} \rangle + \langle \mathbf{p_2} \rangle + \langle \mathbf{p_3} \rangle + \langle \mathbf{p_4} \rangle$. The place *ini_Unused* initially contains four tokens, one per color in the color domain **Pump**.

### B.2. SWN composition features

The SWN composition rules used are based on the concept of *matching labels*, that is transitions and places of a SWN are labeled and pairs of transitions (or places) with matching labels, each one belonging to a different operand, i.e., SWN component, are superposed.

A SWN component is then defined as a triplet $\mathscr{LN} = (\mathscr{N}, \psi, \lambda)$ where $\mathscr{N}$ is a SWN, and $\psi : P \to L_P \cup \{\tau\}$ and $\lambda : T \to L_T \cup \{\tau\}$ are the place and transition labeling functions, respectively. Net objects labeled as $\tau$ are considered non-observable with respect to the composition,

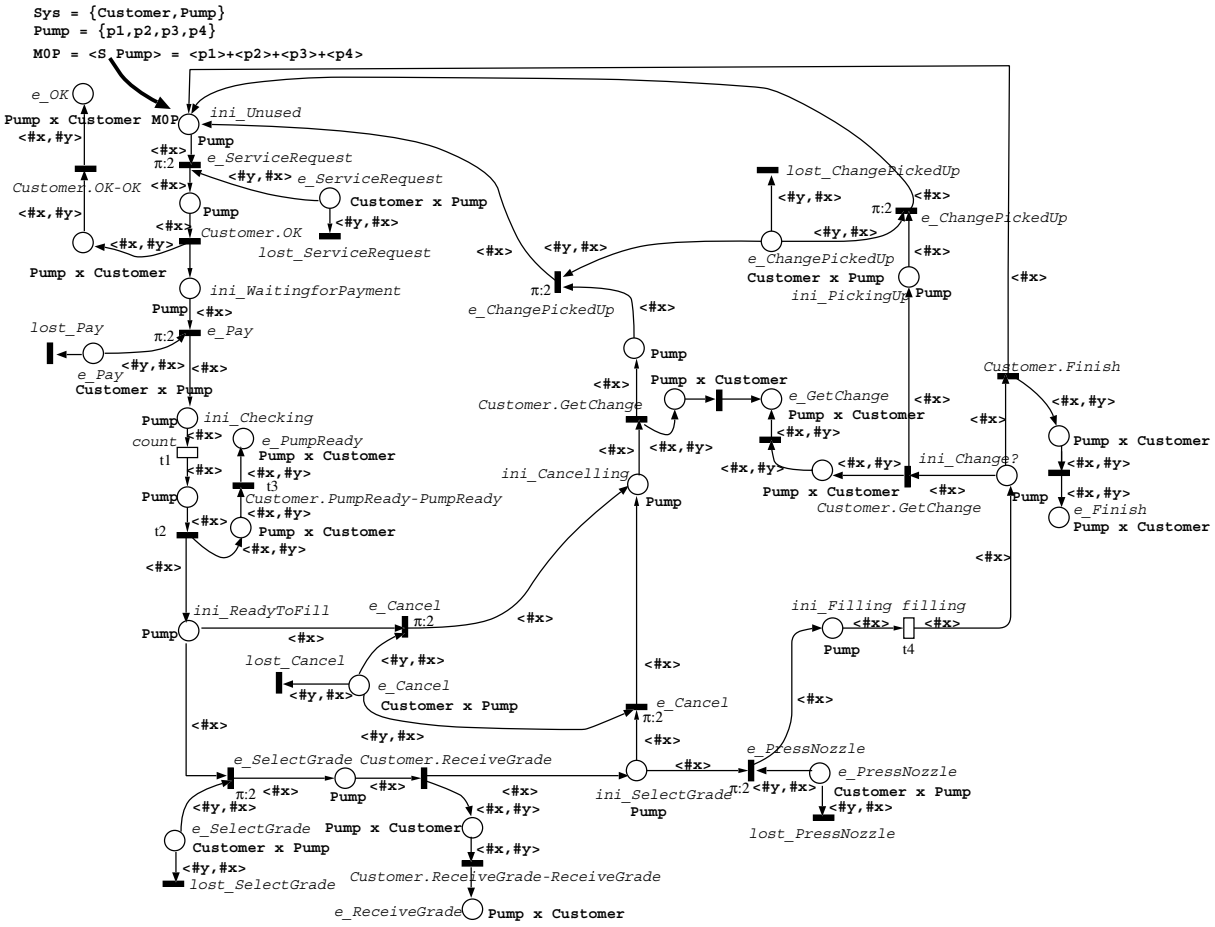


Fig. B.1. A sample SWN.

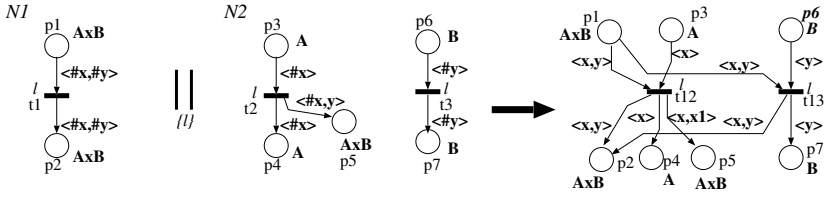


Fig. B.2. SWN transition composition operator.

and those whose labels do not appear in the other operand are not involved in the composition.

An exemplification of how the transition superposition operator works is given in Fig. B.2: the two SWN components $\mathcal{N}_1$ and $\mathcal{N}_2$ are composed over the common labeled transitions (labels are written in italic font). The resulting SWN is shown on the right: it contains the cross product of the transitions of equal label $l \in L_T$, that is $t_{12} \equiv (t_1, t_2)$ and $t_{13} \equiv (t_1, t_3)$. Observe that the variable names prefixed by the # symbol are not renamed during the composition, then allowing to unify values. This is the option we have used in this paper. On the other hand, the variables names which are not prefixed by # are renamed in the composed net; e.g., compare the arc expressions of the arc connecting transition $t_2$ to place $p_5$ in $\mathcal{N}_2$ and the corresponding arc, in the composed SWN, connecting $t_{12}$ to $p_5$. The place superposition operation is the direct counterpart of the transition composition with the additional constraint that the color domain of places to be superposed have to be identical.

## References

Balbo, G., 1995. On the success of stochastic Petri nets. In: Proceedings of the 6th International Workshop of Petri Nets and Performance Models (PNPM95), Durham, NC, USA, pp. 2–9.

Ballarini, P., Bernardi, S., Donatelli, S., 2002. Validation and evaluation of a software solution for fault tolerant distributed synchronization.

Proceedings of International Conference on Dependable Systems and Networks (DSN). IEEE Computer Society, Los Alamitos, CA, USA, pp. 773–782.

Ballarini, P., Capra, L., Franceschinis, G., Pierro, M.D., 2003. Memory fault tolerance software mechanisms: design and configuration support through SWN models. Third International Conference on Application of Concurrency to System Design (ACSD'03). IEEE Computer Society, Los Alamitos, CA, USA, pp. 111–121.

Balsamo, S., Marzolla, M., 2003. A simulation-based approach to software performance modeling. Proceedings of ESEC/FSE. ACM, Helsinky, Finland, pp. 363–366.

Baresi, L., Pezzè, M., 2001. On formalizing UML with high-level Petri nets. In: Agha, G., De Cindio, F., Rozenberg, G. (Eds.), Concurrent Object-Oriented Programming and Petri Nets. State of the Art, Advances in Petri Nets. Lecture Notes in Computer Science (LNCS), vol. 2001. Springer-Verlag, Heidelberg, pp. 276–304.

Bernardi, S., Merseguer, J., 2006. QoS assessment via stochastic analysis. IEEE Internet Computing 10 (3), 32–42.

Bernardi, S., Donatelli, S., Horváth, A., 2001. Implementing compositionality for stochastic Petri nets. International Journal on Software Tools for Technology Transfer (STTT) 3 (4), 417–430.

Bernardi, S., Donatelli, S., Merseguer, J., 2002. From UML sequence diagrams and statecharts to analysable Petri net models. Proceedings of the 3rd Workshop on Software and Performance (WOSP'02). ACM, Roma, Italy, pp. 35–45.

Bobbio, A., Bologna, S., Ciancamerla, E., Franceschinis, G., Gaeta, R., Minichino, M., Portinale, L., 2001. Comparison of methodologies for the safety and dependability assessment of an industrial programmable logic controller. In: Proceedings of European Safety and Dependability Conference, pp. 411–418.

Bondavalli, A., Dal Cin, M., Latella, D., Majzik, I., Pataricza, A., Savoia, G., 2001. Dependability analysis in the early phases of UML-based system design. International Journal of Computer Systems Science & Engineering 16 (5), 265–275.

Bouabana-Tebibel, T., Belmesk, M., 2004. Formalization of UML object dynamics and behavior. In: IEEE International Conference on Systems, Man and Cybernetics, pp. 4971–4976.

Canevet, C., Gilmore, S., Hillston, J., Prowse, M., Stevens, P., 2003. Performance modelling with UML and stochastic process algebras. IEE Proceedings: Computers and Digital Techniques 150 (2), 107–120.

Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S., 1993. Stochastic well-formed coloured nets for symmetric modelling applications. IEEE Transactions on Computers 42 (11), 1343–1360.

Chiola, G., Dutheillet, C., Franceschinis, G., Haddad, S., 1997. A symbolic reachability graph for coloured Petri nets. Theoretical Computer Science B (Logic, Semantics and Theory of Programming) 176 (1–2), 39–65.

Cortellessa, V., Mirandola, R., 2000. Deriving a queueing network based performance model from UML diagrams. Proceedings of the Second International Workshop on Software and Performance (WOSP2000). ACM, Ottawa, Canada, pp. 58–70.

De Miguel, M., Lambolais, T., Hannouz, M., Betge, S., Piekarec, S., 2000. UML extensions for the specification of latency constraints in architectural models. Proceedings of the Second International Workshop on Software and Performance (WOSP2000). ACM, Ottawa, Canada, pp. 83–88.

Franceschinis, G., Bertoncello, C., Bruno, G., Lungo-Vaschetti, G., Pigozzi, A., 2001. SWN models of a contact center: a case study. 9th International Workshop on Petri Nets and Performance Models (PNPM'01). IEEE Computer Society, Los Alamitos, CA, USA, pp. 39–48.

Gaeta, R., Chiola, G., 1995. Efficient simulation of SWN models. Proceedings of the Sixth International Workshop on Petri Nets and Performance Models. IEEE Computer Society, Washington, DC, USA, pp. 137–147.

Grassi, V., Mirandola, R., Sabetta, A., 2005. From design to analysis models: a kernel language for performance and reliability analysis of component-based systems. In: Proceedings of the Fifth International Workshop on Software and Performance (WOSP'05), pp. 25–36.

Gu, G., Petriu, D., 2005. From UML to LQN by XML algebra-based model transformations. Proceedings of the Fifth International Workshop on Software and Performance (WOSP'05). ACM, Palma de Mallorca, Spain, pp. 99–110.

Haugen, O., Husa, K., Runde, R., Stolen, K., 2005. STAIRS: towards formal design with sequence diagrams. Software & System Modeling 4 (4), 255–267.

Hu, Z., Shatz, S.M., 2004. Mapping UML diagrams to a Petri net notation for system simulation. In: Maurer, F., Ruhe, G. (Eds.), Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004), Banff, Alta., Canada, pp. 213–219.

Jansen, D.N., Hermanns, H., Katoen, J.-P., 2003. A QoS-oriented extension of UML statecharts. Proceedings of the 6th International UML Conference. In: Lecture Notes in Computer Science, vol. 2863. Springer, pp. 76–91.

Lazowska, E., Zahorjan, J., Scott Graham, G., Sevcik, C., 1984. Quantitative System Performance: Computer System Analysis Using Queueing Network Models. Prentice-Hall.

López Grao, J.P., Merseguer, J., Campos, J., 2004. From UML activity diagrams to stochastic Petri nets: application to software performance engineering. Proceedings of the Fourth International Workshop on Software and Performance (WOSP'04). ACM, Redwood City, CA, USA, pp. 25–36.

Merseguer, J., 2003. Software performance engineering based on UML and Petri nets. Ph.D. thesis, University of Zaragoza, Spain, March.

Merseguer, J., Campos, J., 2003. Exploring roles for the UML diagrams in software performance engineering. Proceedings of the 2003 International Conference on Software Engineering Research and Practice. CSREA Press, Las Vegas, Nevada, USA, pp. 43–47.

Merseguer, J., Bernardi, S., Campos, J., Donatelli, S., 2002. A compositional semantics for UML state machines aimed at performance evaluation. In: Silva, M., Giua, A., Colom, J.M. (Eds.), WODES02: 6th International Workshop on Discrete Event Systems. IEEE Computer Society, Zaragoza, Spain, pp. 295–302.

Merseguer, J., Campos, J., Mena, E., 2003. Analysing Internet software retrieval systems: modeling and performance comparison. Wireless Networks: The Journal of Mobile Communication, Computation and Information 9 (3), 223–238.

OMG-UML, 2005. Unified Modeling Language: Superstructure Object Management Group, version 2.0, formal/05-07-04. http://www.omg.org, July 2005.

OMG-UML-SPT, 2005. UML Profile for Schedulability, Performance and Time. Object Management Group, version 1.1, formal/05-01-02, January 2005.

Pettit IV, R., Gomaa, H., 2004. Modeling behavioral patterns of concurrent software architectures using Petri nets. 4th Working IEEE/IFIP Conference on Software Architecture (WICSA). IEEE Computer Society, Oslo, Norway, pp. 57–68.

Saldhana, J., Shatz, S., 2000. UML diagrams to object Petri net models: an approach for modeling and analysis. Twelfth International Conference on Software Engineering and Knowledge Engineering. Knowledge Systems Institute, Chicago, IL, USA, pp. 103–110.

Whittle, J., 2006. Specifying precise use cases with use case charts. In: Bruel, J.-M. (Ed.), Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica, October 2–7, 2005, Lecture Notes in Computer Science, vol. 3844. Springer.

Woodside, M., Petriu, D., Petriu, D., Shen, H., Israr, T., Merseguer, J., 2005. Performance by unified model analysis (PUMA). Fifth International Workshop on Software and Performance (WOSP'05). ACM, Palma, Illes Balears, Spain, pp. 1–12.

**Simona Bernardi** is a researcher at the University of Torino, Italy. Her research interests include software performance and dependability engineering, stochastic modeling, and UML. She has a BS, an MS in mathematics, and a PhD in computer science, both from the University of

Torino. She has served as a referee for international journals and conferences.

**José Merseguer** is an assistant professor in the Department of Computer Science and Systems Engineering at the University of Zaragoza, Spain. His research interests include performance and dependability analysis of software systems, UML semantics, and object-oriented software engineering. He has a BS and an MS in computer science from the Technical University of Valencia and a Ph.D. in computer science from the University of Zaragoza. He has served as a referee for international journals and as a program committee member for several international conferences and workshops.