Software Performance Modeling using UML and Petri nets*

José Merseguer and Javier Campos

Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
C/María de Luna, 1, 50018 Zaragoza, Spain
{jmerse, jcampos}@unizar.es

Abstract. Software systems are today one of the most complex artifacts, they are simultaneously used by hundred-thousand of people sometimes in risk real time operations, such as auctions or electronic commerce. Nevertheless, it is a common practice to deploy them without the expected performance. Software Performance Engineering has emerged as a discipline to complement Software Engineering research in order to address this kind of problems. In this work, we survey some recent contributions in the field of Software Performance Engineering. The approach surveyed has as main features that it uses the UML diagrams to specify the functional and performance requeriments of the system and the stochastic Petri nets formalism to analyse it.

1 Introduction

The measurement of any system or device developed in the framework of any engineering field should be cleary identified as a stage in the life cycle of the product. Traditionally, three methods have been proposed, sometimes in complementary ways, to reveal how a system performs: direct measurement, simulation and analythical techniques. Several reasons have pointed the last two methods as the most significant, among others, for engineers because they allow to test the device before its development and for researchers since mathematical tools can be applied to model and experiment such devices at a lower cost. Both methods share the goal of creating a performance model of the system/device that accurately describes its load, routing rates and activities duration. Performance models are often described in some stochastic formalism that provides the required analysis and simulation capabilities, e.g. queuing network models [20], stochastic Petri nets [29] or stochastic process algebra [17]. On the other hand, the success of these methods and performance models has been mostly attached to the available tools to apply and assist them.

In contrast with these common engineering practices, software systems are sometimes deployed without the performance expected by clients, being usual

* This work has been developed within the projects P084/2001 of the Aragonese Government and TIC2002-04334-C03-02 and TIC2003-05226 of the Spanish Ministry of Science and Technology.

among developers to test the performance of their systems only when they have been implemented, following the well-known "fix-it-later" approach. It has been common to believe that powerful hardware at minor extra cost will solve performance problems when the software is deployed. The inadequacy of these actitudes has been justified by the youth of the software engineering discipline.

In the last years, software performance engineering [40] (SPE) has grown as a field to propose methods and tools to effectively overcome these problems attached to software development. The emergence of such a discipline becames a necessity when thinking that today complex distributed systems, most of them operating in the Internet, are avid of high quality requirements of security, dependability, performance or responsiveness.

The state of the research proposals of the SPE field suggests that they cannot be considered today to be applied in the industry. It is true that some of these advances have been applied and a variety of examples can be found about their effective application, but in our opinion they are not stablished as a systematic approach. The software industry, being concerned about the problems involving software quality, has actively participated throughout the OMG [30] consortium in the development of a standard to channel research advances into its industrial application. We refer to the UML Profile for Schedulability, Performance and Time Specification [31], the Profile later on.

The election of the Unified Modeling Language [32] (UML) to build the Profile on it, is not a coincidence, since the SPE community decided in its first international workshop [44] to consider the UML as the reference notation to describe software systems and their performance requirements. Among other goals, the Profile tries to:

- Enable the construction of models that could be used to make quantitative predictions regarding these characteristics.
- Facilitate communication of design intent between developers in a standard way.
- Enable inter operability between various analysis and design tools.

This paper surveys some recent contributions in the field of SPE, also it briefly describes the state of our current work and the lines that will be followed in a near future. In these last years, we have been working in the development of a SPE proposal, actually a method and a tool, to evaluate performance estimates of software systems in the early stages of the software life cycle.

Despite the fact that we started the development of the approach surveyed here previously to the definition of the Profile, it follows the most of the directions stated by the Profile. Moreover in some of the directions that the approach diverges from the Profile, we have tried to rechannel them to follow it.

Among the changes promoted in this approach to follow the Profile, it can be noticed that the tagged valued language and the metamodel to describe performance requirements is now followed instead of the previous *pa-UML* [27, 28].

In other aspects the proposal differs significantly from the Profile. In [25] a SPE role for the use case and statechart diagrams is defined in the context of object oriented approach to describe the life of system classes, that clearly differs

from the Profile, that is based on an scenario-based approach using activity and sequence diagrams.

In the following the big picture of the approach is given. The use case diagram is proposed to calculate, among other estimates, the system usage by actors. The statechart diagram is used to model the life of each class in the system and to describe its performance requirements. The actions in the statechart are described at a lower level by means of the activity diagram. Statecharts and activity diagrams are translated into stochastic Petri nets to conform a performance model representing the whole system. The sequence diagram, representing patterns of interactions among objects, is translated togheter with the statechart of the corresponding objects into a stochastic Petri net representing a performance model of a concrete execution of the system. Other diagrams such as the deployment or classes are now being studied to be coupled with the others.

At the same time we have been working in the development of a tool, at a prototype level, to support this proposal. Starting from the ArgoUML [36] CASE tool, it can be enriched with the capabilities to describe performance requeriments according to the Profile. Moreover, a module that translates the UML diagrams into the corresponding Petri nets has been developed and attached to the tool. The obtained Petri net files are sufficient to feed the GreatSPN [16] tool that can simulate or analyze them in order to obtain the desired estimates.

As a new work, some steps are announced in this paper towards the integration of the results in SPE surveyed here with previous results in the efficient performance analysis of stochastic Petri nets [7–9, 33]. We consider interesting to merge both fields since it is expected to reach performance analysis techniques for the models obtained from the translation that contribute to palliate the state explosion problem inherent to the Petri net formalism.

The rest of the article is structured as follows. In section 2, an introduction to the notations and formalisms related with the proposal is given, assuming the reader is familiar with both UML and Petri nets terminologies and concepts. Section 3 depicts our proposal of a method for SPE and travels around each UML diagram in the proposal surveying its performance role under our interpretation and in some cases its translation into the Petri net formalism. Section 4 is devoted to envisage the future steps of our work in the field of the efficient analysis of the obtained performance models. Related work is explored in section 5. Finally conclusions of the work are obtained in section 6.

2 Context of the work

The work surveyed in this article relies on a number of fields and formalisms, they are addressed in this section. We start by recalling the main proposals of the SPE. After, the UML notation is introduced and then we study the part of the Profile of interest for this work. Finally, the stochastic formulation of the Petri net formalism used in the resulting performance models is described, as well as some issues concerned with their analysis.

2.1 Software Performance Engineering

The term software performance engineering (SPE) was first introduced by C.U. Smith in 1981 [41]. Several complementary definitions have been given in the literature to describe the aim of the SPE. Among them it can be remarked the following:

- In [40], SPE is proposed as a method (a systematic and quantitative approach) for constructing software systems to meet performance objectives, taking into account that SPE augments other software engineering methodologies but it does not replace them.
- SPE is defined in [38] as a collection of methods for the support of the performance-oriented software development of application systems throughout the entire software development process to assure an appropriate performance related product quality.
- Finally, in [42] new perspectives for SPE are devised, then proposing that SPE must provide principles, patterns [26, 15] and antipatterns [43] for creating responsive software, the data required for evaluation, procedures for obtaining performance specifications and guidelines for the types of evaluation to be conducted at each development stage.

It is important to remark that the previous definitions emphasize that SPE cannot be placed outside the context of software engineering. This fact contrasts with other engineering fields, such as telecommunication, where performance practices have been applied successfully in "isolation", i.e. not explicitly while developing the engines. Moreover SPE, as pointed out in [38], reuses and enlarges concepts and methods from many other disciplines such as: Performance management, performance modeling, software engineering, capacity planning, performance tunning and software quality assurance.

The SPE process proposed in [40] still remains as a reference for a very general proposal to establish the basic steps that a SPE process should consider. Firstly, the goals or quantitative values must be defined, obviously changing from one stage of the software life cycle to other. Data *gathering* is accomplished by defining the proper scenarios interpreting how the system will be typically used and its possible deviations (defining upper and lower bounds when uncertainties are present). The construction and evaluation of the performance model associated with the system is one of the fundamentals in SPE. Validation and verification of the performance model are on-going activities of the SPE process.

The common paradigms of stochastic models used in SPE are the queuing network models [20], stochastic process algebras [17] and stochastic Petri nets [29].

2.2 The Unified Modeling Language

The Unified Modeling Language [32] (UML) is a semi formal language developed by the OMG to specify, visualize and document models of software systems and non-software systems too. UML defines twelve types of diagrams, divided into

three categories: static diagrams, behavioral diagrams and diagrams to organize and manage application modules. Behavioral diagrams (sequence, collaboration, use case, statechart and activity) constitute a major aim in this work since the most performance issues of systems can be represented by means of them.

The semantics of the behavioral UML diagrams is specified in the *Behavioral Elements* package, which is decomposed into the following subpackages: Common behavior, collaborations, use cases, state machines and activity graphs. The last four subpackages give semantics to each one of the UML behavioural diagrams.

In this work a translation of each UML behavioural diagram into a stochastic Petri net is proposed. The translation is based on the UML metamodel defined for each subpackage. Figure 1 shows the UML metamodel for the state machines subpackage, the rest of the metamodels can be found in [32].

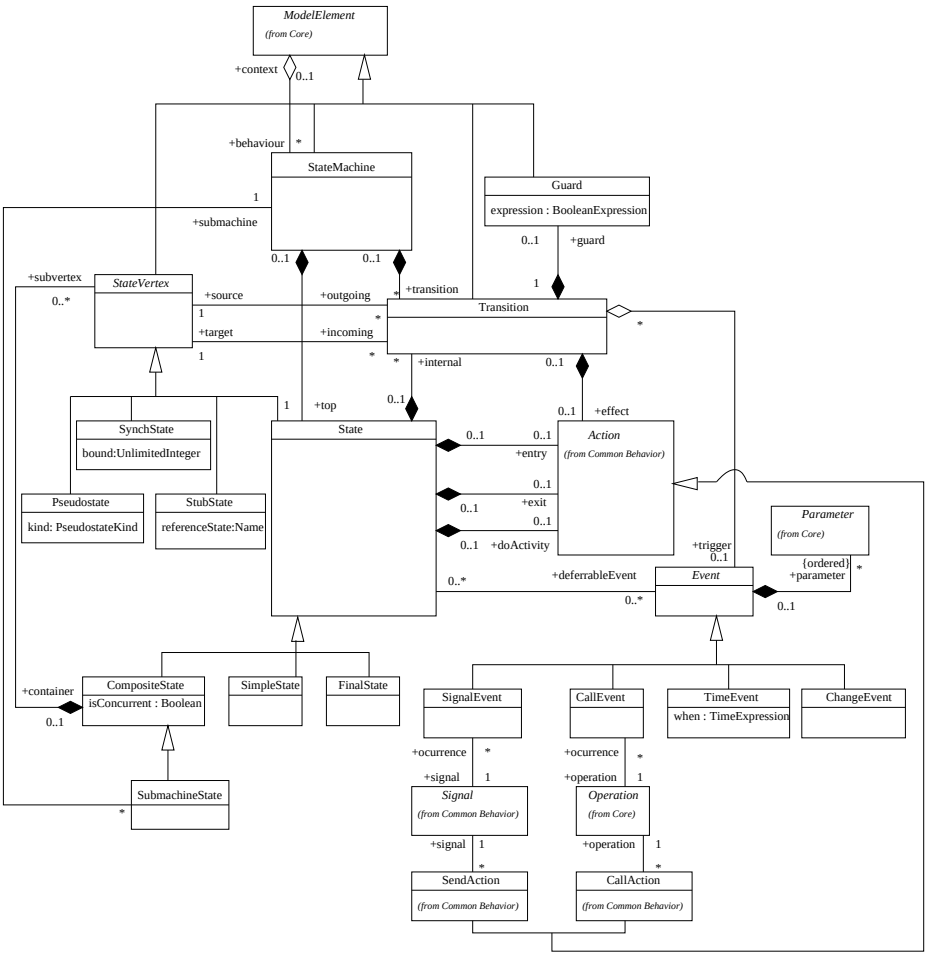

Fig. 1. UML state machines metamodel.

2.3 UML Profile for Schedulability, Performance and Time Specification

The adoption of this Profile [31] by the OMG in March 2002 was of major importance for the SPE community. Its main purposes are:

- to encompass different real-time modeling techniques,
- to annotate UML real-time models to predict timeliness, performance and schedulability characteristics based on the analysis of these software models.

Although the profile is oriented to real time systems, the annotations proposed in the *performance sub-profile* remain valid for more general purposes, such as the specification of distributed software systems in non real time environments, which are the ones of interest in this work.

The sub-profile extends the UML metamodel with stereotypes, tagged values and constraints to attach performance annotations to a UML model. It provides facilities for:

1. capturing performance requirements within the design context,
2. associating performance-related QoS characteristics with selected elements of a UML model,
3. specifying execution parameters which can be used by modeling tools to compute predicted performance characteristics,
4. presenting performance results computed by modeling tools or found in testing.

In order to meet these objectives the performance sub-profile extends the UML metamodel with the following abstractions. The QoS requirements are placed on *scenarios*, which are executed by *workloads*. The workload is *open* when its requests arrive at a given rate and *closed* when there are a fixed number of potential users executing the scenario with a "think time" outside the system. The scenarios are composed by *steps*, i.e. elementary operations. *Resources* are modeled as servers and have *service time*. *Performance measures* (utilizations, response times, ...) can be defined as required, assumed, estimated or measured values.

The SPE approach surveyed in this work proposes a method and a tool following the objectives given by the sub-profile and the guidelines of the SPE summarized in section 2.1. The main difference with the sub-profile is that it proposes an scenario-based approach (using collaborations and activity diagrams), while this approach tries to identify the role of each UML diagram (use cases, interactions, statecharts and activity diagrams) in the performance process under an object-oriented perspective. In both cases, the final objective is to obtain a set of annotated UML diagrams that should be the input to create a performance model in terms of some performance modeling paradigm.

2.4 Stochastic Petri nets

A Petri net [39] is a mathematical tool aimed to model a wide-range of concurrent systems. Formally:

Definition 1. *A Petri net is a 5th-tuple* $\mathcal{N} = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$ *such that,*
P *is a set of places,*
T *is a set of transitions,*
$P \cap T = \emptyset$,
$\mathbf{Pre} : P \times T \to \mathbb{N}$ *is the input function,*
$\mathbf{Post} : T \times P \to \mathbb{N}$ *is the output function,*
$\mathbf{m_0} : P \to \mathbb{N}$ *is the initial marking.*

Stochastic Petri nets (SPNs) were proposed [29] as a non deterministic model, then associating with each transition a random firing time. In this work an extension of the SPNs is considered, the class of Generalized Stochastic Petri Nets (GSPNs) [1]:

Definition 2. *A GSPN system is a 8th-tuple* $\langle P, T, \Pi, \mathbf{Pre}, \mathbf{Post}, H, W, \mathbf{m_0} \rangle$ *where,*
$P, T, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0}$ *is a PN as in Def. 1,*
$\Pi : T \to \mathbb{N}$ *is the priority function that maps transitions onto priority levels,*
$H : P \times T \to \mathbb{N}$ *is the inhibition function,*
$W : T \to \mathbb{R}$ *is the weight function that assigns rates (of negative exponential distribution) to timed transitions and weights to immediate transitions.*

The GSPN have been extended to label places and transitions:

Definition 3. *A labeled GSPN system (LGSPN) is a triplet* $\mathcal{LS} = (S, \psi, \lambda)$ *where,*
S *is a GSPN system as in Def. 2,*
$\psi : P \to L^P \cup \tau$ *is a labeling function for places,*
$\lambda : T \to L^T \cup \tau$ *is a labeling function for transitions,*
L^T, L^P *and* τ *are sets of labels,*
τ*-labeled net objects are considered to be "internal", not visible from the other components.*

The LGSPN formalism introduces an operator to *compose* ordinary GSPNs models (i.e., $\mathbf{Pre}(p,t) \in \{0,1\}$ and $\mathbf{Post}(t,p) \in \{0,1\}$), that is of great importance in this work:

Notation 1 (Place and transition superposition of two ordinary LGSPNs).
Given two LGSPN ordinary systems $\mathcal{LS}_1 = (S_1, \psi_1, \lambda_1)$ *and* $\mathcal{LS}_2 = (S_2, \psi_2, \lambda_2)$*, the LGSPN ordinary system* $\mathcal{LS} = (S, \psi, \lambda)$*:*

$$\mathcal{LS} = \mathcal{LS}_1 \underset{L_T, L_P}{||} \mathcal{LS}_2$$

is the composition over the sets of (no τ*) labels* $L_T \subseteq L^T$ *and* $L_P \subseteq L^P$*.*

The definition of this operator can be found in [4]. Figure 2 depicts an informal representation of this operator. This operator is used in this work, among others, to compose, using the labels of some places and transitions, LGSPNs that represent system modules in order to obtain an LGSPN that represents the whole system.

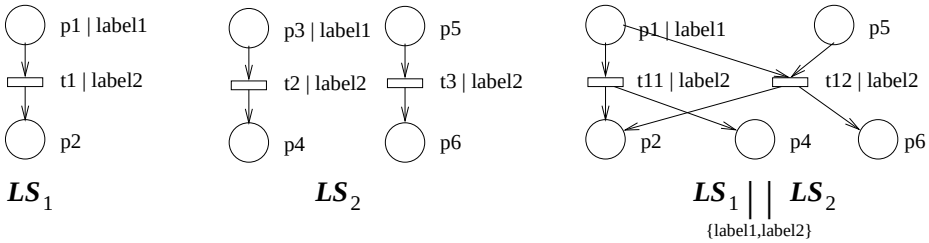

Fig. 2. Superposition of places and transitions

2.5 Analysis of stochastic models

The performance models obtained by the application of the SPE method proposed in this work can be used to estimate *performance measures*, therefore they should be simulated or analyzed, but only the second approach has been applied so far.

Traditionally, techniques for the analysis (computation of performance measures or validation of logical properties) of Petri nets are classified in three complementary groups, enumeration, transformation, and structural analysis:

- *Enumeration* methods are based on the construction of the reachability graph (coverability graph in case of unbounded models), but they are often difficult to apply due to their computational complexity, the well-know *state explosion problem*.
- *Transformation* methods obtain a Petri net from the original one belonging to a subclass easier to analyze but preserving the properties under study, see [5].
- *Structural analysis* techniques are based on the net structure and its initial marking, they can be divided into two subgroups: *Linear programming* techniques, based on the state equation and *graph based* techniques, based on "ad hoc" reasoning, frequently derived from the firing rule.

A complementary classification based on the quality of the obtained results is: exact, approximation and bounding techniques.

- *Exact* techniques are mainly based on algorithms for the automatic construction of the infinitesimal generator of the isomorphic Continuous Time Markov Chain (CTMC). Refer to [1] for numerical solutions of GSPN systems.
- *Approximation* techniques do not obtain the exact solution but an approximation. Some of them substitute the computation of the isomorphic CTMC by the solution of smaller components [33].
- Finally, *bounds* [7, 9] are techniques that offer the further results from the reality. Nevertheless, they can be useful in the early phases of the software life-cycle, in which many parameters are not known accurately.

3 Towards a SPE method

"Large software systems are the most complex artifacts of human civilization" [6]. It is widely accepted that complex systems need formal models to be properly modeled and analyzed, also when the kind of analysis to accomplish is with performance evaluation purposes.

Today still remains the gap among the classical performance evaluation techniques and the classical proposals for software development [37, 19]. Neither of these proposals for software development deal with performance analysis, at most they propose to describe some kind of performance requirements. So, it could be argued that there does not exist an accepted *process* to model and study system performance in the software development process. The Profile [31] should be the starting point to bridge this gap, at least by providing the concepts and notations for the vocabulary of performance requeriments and the guides in the development of performance tools. Then a process should propose the steps to follow to obtain a performance model and it should give trends to analyze this model.

The process presented in this section follows the directions pointed by SPE and the Profile, recalled in sections 2.1 and 2.3, respectively. It emphasizes the previous matters: formal modeling and the need of connecting software and performance well-know practices.

The process to evaluate software performance should be "more or less transparent" for the software designer. By "transparent" is meant that the software designer should be concerned as less as possible to learn new processes since the tasks of analysis and design already imply the use of a process. Therefore, ideally the process to evaluate performance of software systems should not exist, it should be integrated in the common practices of the software engineer. It is our intention that the proposal presented here can be used together with any software life-cycle process, and that the performance model can be semi-automatically obtained as a *by-product* of the software life-cycle. Another important issue for the process is that the performance model should be obtained in the early stages of the software life-cycle. In this way proper actions to solve performance problems take less effort and less economical impact.

Figure 3 gives the big picture of the steps followed by this process, that are the following ones:

1. From the problem domain the software requirements should be modeled using the desired software life cycle paradigm and the UML notation. The meaning of the statecharts will be the description of the behavior of the active classes of the system. The activity diagrams will specify the refinement of the activities in the statecharts. The sequence diagrams will be used to model concrete executions of interest in the context of the system. While developing the models, performance requirements will be modeled by means of the Profile. The role of each diagram in this step and the proposed annotations are described later in this section.
2. In this step functions that translate each UML diagram into an LGSPN should be applied. From these models and applying the composition rules

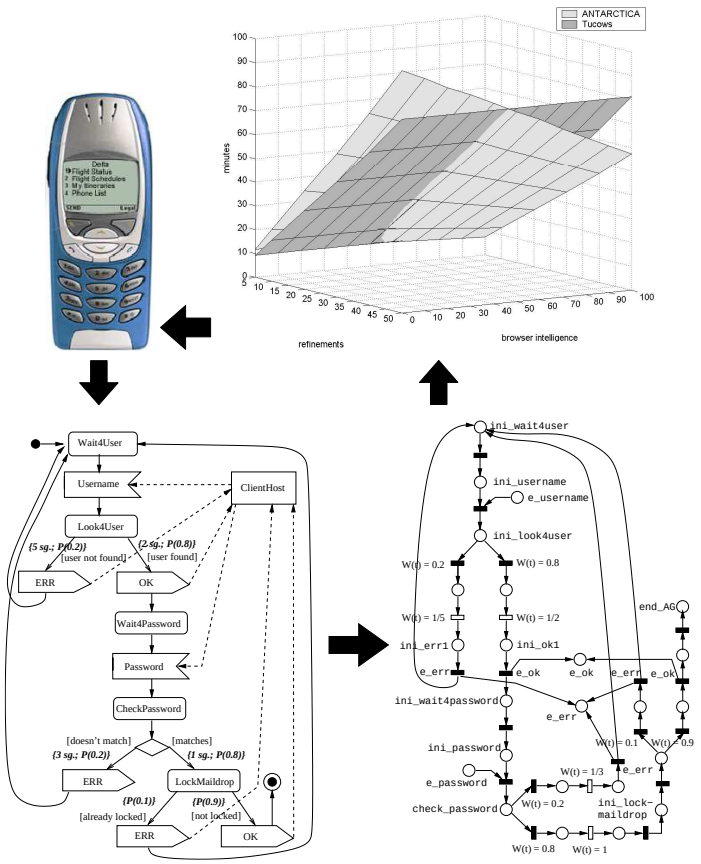

Fig. 3. SPE process.

given in section 3.5 two different performance models can be obtained in terms of LGSPNs: a performance model representing the whole system or representing a concrete execution of the system. Translation and composition rules have been embedded in an augmented CASE tool [22].

3. Finally, the parameters to be computed and the experiments to test the system should be defined. Also, analytical or simulation techniques to solve the formal model should be applied. This phase can be made easier by integrating the augmented CASE tool with stochastic Petri net analysers.

The following sections (from 3.1 to 3.4) address the first and second steps of the proposal. Therefore, they revise the role that each UML diagram should play (under the prespective of the approach) in the SPE process, as well as the kind of performance requirements it is able to gather. Concerning the functions that translate the diagrams into LGSPN, due to space limitations, just the flavor

for those of the statechart is given. The third step of the approach is partially addressed in section 4 by revising efficient techniques to analyze the model.

3.1 Use case diagram

In UML a use case diagram shows actors and use cases together with their relationships. The relationships are associations between the actors and the use cases, generalizations between the actors, and generalizations, extends and includes among the use cases [32].

A use case represents a coherent unit of functionality provided by a system, a subsystem or a class as manifested by sequences of messages exchanged among the system (subsystem, class) and one or more actors together with actions performed by the system (subsystem, class). The use cases may optionally be enclosed by a rectangle that represents the boundary of the containing system or classifier [32].

In the use case diagram in Figure 4 we can see: two actors, three use cases and four associations relationships between actors and use cases, like that represented by the link between the actor1 and the UseCase1.

Role of the use case diagram concerning performance The use case diagram allows to model the usage of the system for each actor. We propose the use case diagram with performance evaluation purposes to show the use cases of interest to obtain performance figures. Among the use cases in the diagram a subset of them will be of interest and therefore marked to be considered in a performance evaluation process.

The role of the use case diagram is to show the use cases that represent executions of interest in the system. Then, a performance model can be obtained for each execution (use case) of interest, that should be detailed by means of the sequence diagram [4].

The existence of a use case diagram is not mandatory to obtain a performance model. In [24] it was shown how a performance model for the whole system can be obtained from the statecharts that describe it.

It is important to recall the proposal in [11], that consists in the assignment of a probability to every edge that links a type of actor to a use case, i.e. the probability of the actor to execute the use case. The assignment induces the same probability to the execution of the corresponding set of sequence diagrams that describes it. Since we propose to describe the use case by means of only one sequence diagram, we can express formally our case as follows.

Let us suppose to have a use case diagram with m users and n use cases. Let $p_i (i = 1, \ldots, m)$ be the i-th user frequency of usage of the software system and let P_{ij} be the probability that the i-th user makes use of the use case $j (j = 1, \ldots, n)$. Assuming that $\sum_{i=1}^{m} p_i = 1$ and $\sum_{j=1}^{n} P_{ij} = 1$, the probability of a sequence diagram corresponding to the use case x to be executed is:

$$P(x) = \sum_{i=1}^{m} p_i \cdot P_{ix} \tag{1}$$

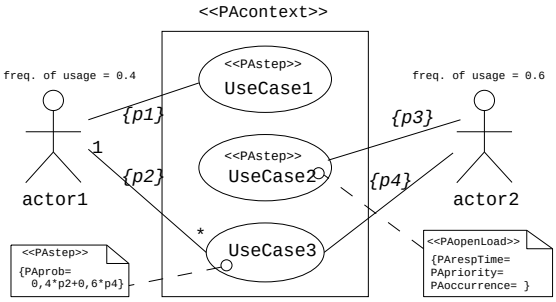

Fig. 4. Use case diagram with performance annotations.

The previous formula, taken from [11], is important because it allows to assign a "weight" to each particular execution of the system. As an example, see in Figure 4 the annotations attached to UseCase3.

The relationships between the actors themselves, and between the use cases themselves are not considered with performance evaluation purposes.

Performance annotations The use case diagram should represent a *Performance Context*, since it specifies one or more scenarios that are used to explore various dynamic situations involving a specific set of resources. Then, it is stereotyped as ≪PAcontext≫. Since there is not a class or package that represents a use case diagram (just the ≪useCaseModel≫ stereotype) the base classes for ≪PAcontext≫ are not incremented.

Each use case used with performance evaluation purposes could represent a *step* (no predecessor neither successor relationship is considered among them). Then, they are stereotyped as ≪PAstep≫, therefore the base classes for this stereotype should be incremented with the class *UseCase*. A load (≪PAclosedLoad≫ or ≪PAopenLoad≫) can be attached to them. Obviously each one of these *steps* should be refined by other *Performance Context*, i.e. a *Collaboration*.

The probabilities attached to each association between an actor and a use case, although not consistently specified, represent the frequencies of usage of the system for each actor (see p1, p2, p3 and p4 in Figure 4). They are useful to calculate the probability for each *Step*, i.e. use case, using equation (1).

3.2 Statechart diagram

A UML statechart diagram can be used to describe the behavior of a model element such as an object or an interaction. Specifically, it describes possible sequences of states and actions through which the element can proceed during its lifetime as a result of reacting to discrete events. A statechart maps into a UML state machine that differs from classical Harel state machines in a number of points that can be found in [32]. Recent studies of their semantics can be found in [24, 13].

A state in a statechart diagram is a condition during the life of an object or an interaction during which it satisfies some condition, performs some action, or waits for some event. A simple transition is a relationship between two states indicating that an object in the first state will enter the second state. An event is a noteworthy occurrence that may trigger a state transition [32].

A composite state is decomposed into two or more concurrent substates (regions) or into mutually exclusive disjoint substates [32].

Role of the statechart concerning performance The *profile* proposes determining system's performance characteristics using scenarios, described by collaborations or activity graphs. By contrast, we have explored an alternative that consists in determining those characteristics from an object's life viewpoint. In order to take a complete view of the system behavior, it is necessary to understand the life of the objects involved in it, being the statechart diagram the adequate tool to model these issues. Then, it is proposed to capture performance requirements at this level of modeling: for each class with relevant dynamic behavior a statechart will specify its routing rates and system usage and load.

The performance requeriments gathered by modeling the statecharts for the system are sufficient enough to obtain a performance model [24]. In this case, all the statecharts togheter represent a *Performance Context* where to explore all the dynamic situations in the system. A particular (and strange) situation arises when only one statechart describes all the system behaviour, then it becomes a *Performance Context*.

Moreover, the statecharts that describe the system (or a subset of them) togheter with a sequence diagram constitute a *Performance Context* that can be used to study parameters associated to concrete executions [4].

In a statechart diagram the useful model elements from the performance evaluation viewpoint are the activities, the guards and the events.

Activities represent tasks performed by an object in a given state. Such activities consume computation time that must be measured and annotated. Activity graphs are adecuate to refine this level of the statechart.

Guards show conditions in a transition that must hold in order to fire the corresponding event. Then they can be considered as system's routing rates.

Events labeling transitions correspond to events in a sequence diagram showing the server or the client side of the object. Objects can reside in the same machine or in different machines for the case of distributed systems. In the first case, it can be assumed that the time spent to send the message is not significant in the scope of the modeled system. Of course, the actions taken as a response of the message can spend computation time, that should be modeled. For the second case, for those messages that travel through the net, we consider that they spend time, then they represent a load for the system that should be modeled.

Performance annotations As proposed in [31] for the activity-based approach, the open or closed workload (induced by the object in our case) is asso-

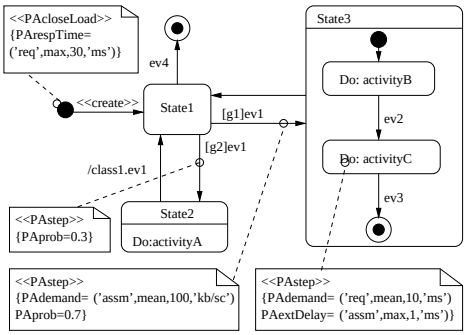

Fig. 5. Statechart with performance annotations.

ciated with the first step in the diagram, in this case the transition stereotyped ≪create≫, see Figure 5.

In pa-UML [23] the annotations for the duration of the activities show the time needed to perform them. If it is necessary, a minimum and a maximum values could be annotated. If different durations must be tested for a concrete activity then a variable can be used. Examples of these labels are {1sec}, {0.5sec..50sec} or {time1}. Using the *profile*, an activity in a state will be stereotyped ≪PAstep≫, then the expressivity is enriched by allowing to model not only response time but also its demand, repetition, intervals, operations or delay, see Figure 5. The successor/predecessor relationship inherent to the ≪PAstep≫ stereotype is not stablised in this case (causing that the probability atribute is not used), firstly because in a state at most one activity can appear [32], but also because it is not of interest to set order among all the activities in the diagram.

By stereotyping the transitions as ≪PAstep≫, it is possible:

A *To consider the guards as routing rates.* The probability of event success represents routing rates in pa-UML by annotating such probability in the guard. Using the *profile*, the attribute probability could be used also to avoid nondeterminism (i.e. transitions labeled with the same event and outgoing from the same state), be aware that this attribute does not provoke a complete order among the succesor *steps* (transitions).

As an example, see Figure 5. The succesor steps of the ≪create≫ transition will be transitions ev4, [g1]ev1 and [g2]ev2, while the predecessor *steps* of transition [g1]ev1 are transitions ev3, ≪create≫ and /class1.ev1. Therefore, it is only necessary to assign probabilities to transitions [g1]ev1 and [g2]ev2.

B *To model the network delays caused by the load of the events (messages) that label them.* The annotations for the load of the messages in pa-UML are attached to the transitions (outgoing or internal) by giving the size of the message (i.e. {1K..100K} or {1K}). Using the *profile*, the size of the message can be specified as a *step* and the network delay as a demand, see transition [g1]ev1 in Figure 5.

Translation into LGSPN The approach taken to translate a statechart into an LGSPN consists in the translation of each element in the metamodel of the state machines (see Figure 1) into an LGSPN subsystem. Just to take the flavor of the translation, Figure 6 depicts the LGSPNs subsystems obtained by the translation of a simple state. The translation of the other elements in the metamodel can be found in [23, 24].

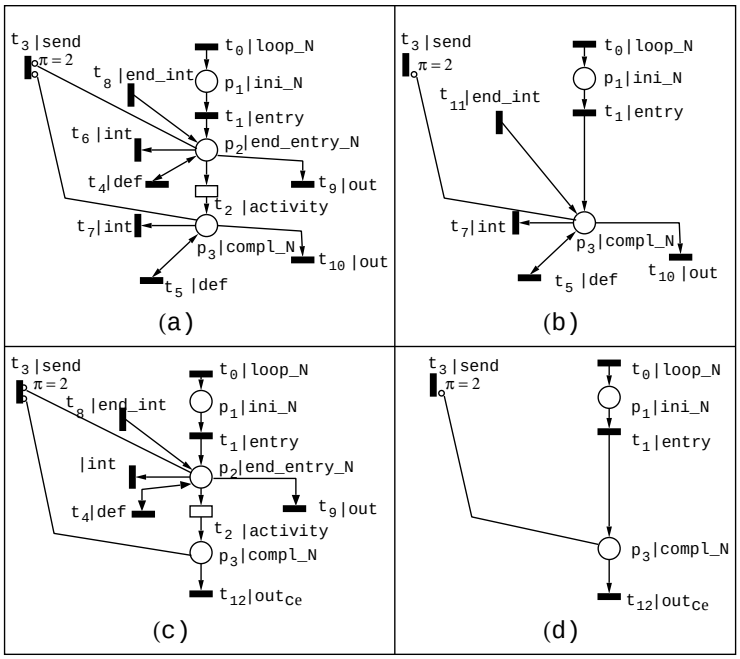

Fig. 6. Different labelled "basic" systems for a simple state N: (a) with activity and no immediate outgoing transition; (b) no activity and no immediate outgoing transition; (c) with activity and immediate outgoing transition; (d) no activity and with immediate outgoing transition.

In order to obtain an LGSPN ($\mathcal{LS}_{sc}$) for a given statechart *sc*, the subsystems that represent its metamodel elements are composed using the operator in Notation 1. The details of the composition method can be found also in [23, 24].

3.3 Activity diagram

Activity diagrams (ADs) represent UML activity graphs and are just a variant of UML state machines (see [32]), in fact, a UML activity graph is a specialization of a UML state machine. The main goal of ADs is to stress the internal control flow of a process in contrast to statecharts, which are often driven by external events.

Role of the activity diagram concerning performance Considering the fact that ADs are suitable for internal flow process modeling, they are relevant to describe activities performed by the system, usually expressed in the statechart as *doActivities* in the states. The AD will be annotated with the information to model routing rates and the duration of the basic actions.

Other interpretations for the AD propose it as a high level modeling tool, that of the workflow systems, but at the moment we do not consider this role in our SPE approach.

According to the UML specification most of the states in an AD should be an action or subactivity state, so most of the transitions should be triggered by the ending of the execution of the entry action or activity associated to the state. Since UML is not strict at this point, then the elements from the SMs package could occasionally be used with the interpretation given in [24].

As far as this issue is concerned, our decision is not to allow other states than action, subactivity or call states, and thus to process external events just by means of call states and control icons involving signals, i.e. signal sendings and signal receipts. As a result of this, events are always deferred (as any event is always deferred in an action state), so an activity will not ever be interrupted when it is described by an AD.

Performance annotations To annotate routing rates and action durations, we consider the stereotype ≪PAstep≫ together with its tag definitions PAprob and PArespTime, respectively.

PArespTime = (<source-modifier>,'max',(n,'s.')), or PArespTime = (<source-modifier>,'dist',(n-m,'s.')). Where <source-modifier>::= 'req'|'assm'|'pred'|'msr'; 'dist' is assumed to be an exponential negative distribution and n-m expresses a range of time.

Annotations will be attached to transitions in order to allow the assignment of different action durations depending on the decision. It implies that the class *Transition* should be included as a base class for the stereotype ≪PAstep≫.

Translation into LGSPN The performance model obtained from an activity diagram in terms of LGSPNs as proposed in [22] can be used with performance evaluation purposes with two goals: A) just to obtain some performance measures of the model element they describe or B) to compose this performance model with the performance models of the statecharts that use the activity modeled in order to obtain a final performance model of the system described by the referred statecharts. The translation process can be found in [22].

3.4 Sequence diagram

A sequence diagram describes a communication pattern performed by instances playing the roles to accomplish a specific purpose, i.e. an interaction. The semantics of the sequence diagram is provided by the collaboration package [32].

Role of the sequence diagram concerning performance For our purposes a sequence diagram should detail the functionality expressed by a use case in the use case diagram, by focusing in the interactions among its participants.

We consider the following relevant elements and constructions of the sequence diagram from the performance point of view to model the load of the system.

Objects can reside in the same machine or in different machines in the case of distributed systems. In the first case it can be assumed that the time spent to send the message is not significant in the scope of the modeled system. Of course the actions taken as a response of the message can spend computation time, but it will be modeled in the statechart diagram. For the second case, those messages that *travel through the net*, it is considered that they spend time, then representing a load for the system that should be modeled in this diagram.

A *condition* can be attached to each message in the diagram, representing the possibility that the message could be dispatched. Even multiple messages can leave a single point each one labeled by a condition. From the performance point of view it can be considered that routing rates are attached to the messages.

A set of messages can be dispatched multiple times if they are enclosed and marked as an *iteration*. This construction also has its implications from the performance point of view.

Performance annotations A deeper explanation of some of the annotations given in the following can be found in [3].

The sequence diagram should be considered as a *performance context* since it will be used to obtain a performance model in terms of LGSPN. Then it will be stereotyped as ≪PAcontext≫.

Each message will be considered as a ≪PAstep≫, it allows that:

- as in the case of statecharts and ADs the *conditions* of the messages are represented by probabilities, then with tagged value PAprob;
- *iterations* will be represented by the tagged value PArep;
- to represent the time spent by a message *travelling through the net*, its size (e.g. 160 Kbytes) and the speed of the net (e.g. 8 Kbytes/sec) should be considered. For this example, the delay of the message is 20 sec., it will be represented by the tagged value PArespTime as follows ('assm','mean',(20,'sec')).

Translation into LGSPN In [4], the translation process of a given sequence diagram *sd* into its corresponding LGSPN $\mathcal{LS}_{sd}$ can be found.

The LGSPN $\mathcal{LS}_{sd}$ will be composed (using the operator in Notation 1) with the LGSPNs that represent the statecharts involved in the interaction. Therefore obtaining a performance model that represents a concret execution of the system, obviously that execution described by the sequence diagram.

3.5 LGSPN performance models

In this section we address the topic of how to obtain a performance model in terms of LGSPNs from a system description in terms of a set of UML diagrams.

Two different kind of performance models can be obtained using the translations surveyed in the previous subsections.

A Supose a system described by a set of UML statecharts $\{sc_1, \ldots, sc_k\}$ and a set of activity diagrams refining some of their *doActivities*, $\{ad_1, \ldots, ad_l\}$. Then $\{\mathcal{LS}_{sc_1}, \ldots, \mathcal{LS}_{sc_k}\}$ and $\{\mathcal{LS}_{ad_1}, \ldots, \mathcal{LS}_{ad_l}\}$ represent the LGSPNs of the corresponding diagrams.
$\mathcal{LS}_{sc_i - ad_j}$ will represent an LGSPN for the statechart sc_i and the activity diagram ad_j.
Then a performance model representing the whole system can be obtained by the following expression:

$$\mathcal{LS} = \mathop{||}_{Labels}^{i=1,\ldots,kj=1,\ldots,l} \mathcal{LS}_{sc_i - ad_j}$$

The works in [24, 22, 23] detail the composition method.

B If a concrete execution of the system $\mathcal{LS}$ in [A] is described by a sequence diagram sd, $\mathcal{LS}_{sd}$ represents its corresponding LGSPN.
Then a performance model representing this concrete execution of the system can be obtained by the following expression:

$$\mathcal{LS}_{execution} = \mathcal{LS} \mathop{||}_{Labels} \mathcal{LS}_{sd}$$

The works in [4, 3] detail the composition method.

4 Analysis of the system

As stated in section 2.5, a fundamental question in the use of stochastic PN models for performance evaluation, even under Markovian stochastic interpretation, is the so called *state explosion problem*. A general approach to deal with (computational) complexity is to use a *divide and conquer* (D&C) strategy, what requires the definition of a decomposition method and the subsequent composition of partial results to get the full solution. On the other hand, the trade-off between computational cost and accuracy of the solution leads to the use of approximation or bounding techniques instead of the computation of exact values. In this context, a pragmatic compromise to be handled by the analyzer of a system concerns the definition of faithful models, that may be very complex to exactly analyse (what may lead to the use of approximation or just bounding techniques), or simplified models, for which exact analysis can be, eventually, accomplished. D&C strategies can be used with exact, approximate, or bounding techniques.

The D&C techniques for performance evaluation present in the literature consider either *implicit or explicit decomposition* of PN models.

In [7], an *implicit* decomposition into P-semiflows is used for computing throughput *bounds* in polynomial time on the net size for free choice PN's with arbitrary pdf of time durations. A generalization to arbitrary P/T models (and

also to coloured models) is presented in [9]. These techniques that use an implicit decomposition are directly applicable to the PN models obtained from UML diagrams (section 3). The bounds are computed from the solution of proper linear programming problems, therefore they can be obtained in polynomial time on the size of the net model, and they depend only on the mean values of service time associated to the firing of transitions and the routing rates associated with transitions in conflict and not on the higher moments of the probability distribution functions of the random variables that describe the timing of the system. The idea is to compute vectors $\boldsymbol{\chi}$ and $\overline{\boldsymbol{\mu}}$ that maximize or minimize the throughput of a transition or the average marking of a place among those verifying several operational laws and other linear constraints that can be easily derived from the net structure. The detailed linear programming problem to solve can be found in [9] and it has been implemented in the *GreatSPN* software tool [16].

In [8], an *explicit* decomposition of a general P/T net into *modules* (connected through buffers) is defined by the analyzer or provided by model construction. The modules are complemented with an abstract view of their environment in the full model. A computational technique is also presented in [8] that uses directly the information provided by the modules to compute the *exact* global limit probability distribution vector without storing the infinitesimal generator matrix of the whole net system (by expressing it in terms of a tensor algebra formula of smaller matrices). In [33], the same decomposition used in [8] is applied to compute a throughput *approximation* of the model through an *iterative technique* looking for a fixed point.

In both [8] and [33], the model decomposition and, eventually, the solution composition process should be net-driven (i.e., derived at net level). Fortunately, the process that is surveyed in section 3 leads to a global PN system that consists on the composition of PN modules (those explained in section 3.5) through fusion of places, thus both the tensor algebra based technique [8] and the approximation technique [33] can be considered to deal with the state explosion problem.

We start by reviewing the decomposition technique for general PN systems that was proposed in [8] (more details can be found there).

An arbitrary PN system can always be observed as a set of *modules* (disjoint simpler PN systems) that asynchronously communicate by means of a set of *buffers* (places).

Definition 4 (SAM). [8] *A strongly connected PN system,* $\mathcal{S} = \langle P_1 \cup \ldots \cup P_K \cup B, T_1 \cup \ldots \cup T_K, \mathbf{Pre}, \mathbf{Post}, \mathbf{m_0} \rangle$*, is a* System of Asynchronously Communicating Modules*, or simply a SAM, if:*
1. $P_i \cap P_j = \emptyset$ *for all* $i, j \in \{1, \ldots, K\}$ *and* $i \neq j$*;*
2. $T_i \cap T_j = \emptyset$ *for all* $i, j \in \{1, \ldots, K\}$ *and* $i \neq j$*;*
3. $P_i \cap B = \emptyset$ *for all* $i \in \{1, \ldots, K\}$*;*
4. $T_i = P_i^{\bullet} \cup {}^{\bullet}P_i$ *for all* $i \in \{1, \ldots, K\}$*.*
The net systems $\langle \mathcal{N}_i, \mathbf{m_{0}}_i \rangle = \langle P_i, T_i, \mathbf{Pre}_i, \mathbf{Post}_i, \mathbf{m_{0}}_i \rangle$ *with* $i \in \{1, \ldots, K\}$ *are called* modules *of* $\mathcal{S}$ *(where* $\mathbf{Pre}_i$*,*$\mathbf{Post}_i$*, and* $\mathbf{m_{0}}_i$ *are the restrictions of* $\mathbf{Pre}$*,* $\mathbf{Post}$*, and* $\mathbf{m_0}$ *to* P_i *and* T_i*). Places in* B *are called* buffers*. Transitions belonging*

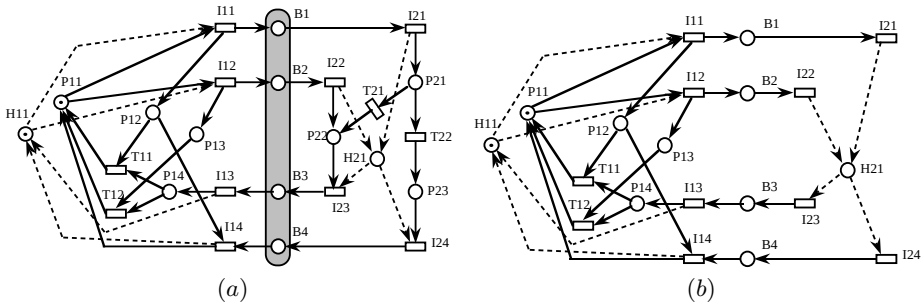

Fig. 7. (a) A SAM and (b) its LS1.

to the set TI $= {}^\bullet B \cup B^\bullet$ *are called* interface *transitions. Remaining ones* $((T_1 \cup \ldots \cup T_K) \setminus \text{TI})$ *are called* internal *transitions.*

A SAM with two modules (the subnets generated by nodes whose tag starts with P_i or T_i for $i = 1, 2$) and four buffers (B_1, B_2, B_3, B_4) is depicted in Fig. 7.a.

All the strongly connected PN systems belong to the SAM class with the only addition of a *structured view* of the model (either given by construction or decided after observation of the model). In the case of PN's obtained from the SPE process introduced in section 3, the structured view is obtained from the net components that correspond to each superposed LGSPN.

In [8], a reduction rule has been introduced for the *internal behaviour* of modules of a SAM. Each module is decomposed into several pieces and each piece is substituted by a set H_i^j of new special places called *marking structurally implicit places* (MSIP's). Later, using that reduction, the original model can be decomposed into a collection of *low level systems* ($\mathcal{LS}_i$ with $i = 1, \ldots, K$) and a *basic skeleton* ($\mathcal{BS}$). In each $\mathcal{LS}_i$, only one module is kept while the internal behaviour of the others is reduced. In [8], the $\mathcal{LS}_i$ and the $\mathcal{BS}$ are used for a tensor algebra-based exact computation of the underlying CTMC.

An algorithm for the computation of the set H_i^j of MSIP's was proposed in [8]. A place is *implicit*, under interleaving semantics, if it can be deleted without changing the firing sequences. Each MSIP of H_i^j added to $\mathcal{N}$ needs an initial marking for making it implicit. In [10], an efficient method for computing such marking is presented.

The next step for the definition of the $\mathcal{LS}_i$ and the $\mathcal{BS}$ is to define an *extended system* ($\mathcal{ES}$). Consider, for instance, the SAM given in Fig. 7.a. The original net system $\mathcal{S}$ is the net without the places H_{11} and H_{21}. These places are the MSIP's computed to summarise the internal behaviour of the two modules. Place H_{11} summarises the module 1 (the subnet generated by nodes whose tags begin with P_1 or T_1), and place H_{21} summarises the module 2. The $\mathcal{ES}$ is the net system of Fig. 7.a (adding to $\mathcal{S}$ the places H_{11} and H_{21}).

From the $\mathcal{ES}$, we can build the $\mathcal{LS}_i$ and $\mathcal{BS}$. In each $\mathcal{LS}_i$ all the modules $\mathcal{N}_j$ with $j \neq i$, are reduced to their interface transitions and to the implicit places that were added in the $\mathcal{ES}$, while $\mathcal{N}_i$ is fully preserved. Systems $\mathcal{LS}_i$ represent different low level views of the original model. In the $\mathcal{BS}$ all the modules are reduced, and it constitutes a high level view of the system. In Fig. 7.b. the $\mathcal{LS}_1$ of Fig. 7.a is depicted. The $\mathcal{BS}$ is obtained by deleting from Fig. 7.b the nodes whose tags begin with P_1 and T_1.

Then, in [8] is presented how the rate matrix of a stochastic SAM can be expressed in terms of matrices derived from the rate matrix of the $\mathcal{LS}_i$ systems.

Let $\mathbf{Q}$ be the infinitesimal generator of a stochastic SAM. We can rewrite $\mathbf{Q}$ as $\mathbf{Q} = \mathbf{R} - \mathbf{\Delta}$, where $\mathbf{\Delta}$ is a diagonal matrix and $\mathbf{\Delta}[i,i] = \sum_{k \neq i} \mathbf{Q}[i,k]$. The same definition holds for the $\mathcal{LS}_i$ components: $\mathbf{Q}_i = \mathbf{R}_i - \mathbf{\Delta}_i$.

If states are ordered according to the high level state $\mathbf{z}$ (a state in the reachability set of $\mathcal{BS}$), then matrices $\mathbf{Q}$ and $\mathbf{R}$ (respectively, $\mathbf{Q}_i$ and $\mathbf{R}_i$) can be described in terms of blocks $(\mathbf{z}, \mathbf{z}')$. We shall indicate them with $\mathbf{Q}(\mathbf{z}, \mathbf{z}')$ and $\mathbf{R}(\mathbf{z}, \mathbf{z}')$ ($\mathbf{Q}_i(\mathbf{z}, \mathbf{z}')$ and $\mathbf{R}_i(\mathbf{z}, \mathbf{z}')$, respectively).

Diagonal blocks $\mathbf{R}_i(\mathbf{z}, \mathbf{z})$ have non null entries that are due *only* to the firing of transitions in $T_i \setminus \mathrm{TI}$ (internal behaviour), while blocks $\mathbf{R}_i(\mathbf{z}, \mathbf{z}')$ with $\mathbf{z} \neq \mathbf{z}'$ have non null entries due *only* to the firing of transitions in TI.

Let $\mathrm{TI}_{\mathbf{z},\mathbf{z}'}$ with $\mathbf{z} \neq \mathbf{z}'$, be the set of transitions $t \in \mathrm{TI}$ such that $\mathbf{z} \xrightarrow{t} \mathbf{z}'$ in the basic skeleton $\mathcal{BS}$. From a matrix $\mathbf{R}_i(\mathbf{z}, \mathbf{z}')$, with $\mathbf{z} \neq \mathbf{z}'$ we can build additional matrices $\mathbf{K}_i(t)(\mathbf{z}, \mathbf{z}')$, for each $t \in \mathrm{TI}_{\mathbf{z},\mathbf{z}'}$, according to the following definition:

$$\mathbf{K}_i(t)(\mathbf{z}, \mathbf{z}')[\mathbf{m}, \mathbf{m}'] = \begin{cases} 1 & \text{if } \mathbf{m} \xrightarrow{t} \mathbf{m}' \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{m}$ and $\mathbf{m}'$ are two of the states with a high level view equal to $\mathbf{z}$ and $\mathbf{z}'$ respectively: $\mathbf{m}|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}$, and $\mathbf{m}'|_{H_1 \cup \ldots \cup H_K \cup B} = \mathbf{z}'$.

Matrices $\mathbf{G}(\mathbf{z}, \mathbf{z}')$ can then be defined as:

$$\begin{aligned} \mathbf{G}(\mathbf{z}, \mathbf{z}) &= \bigoplus_{i=1}^{K} \mathbf{R}_i(\mathbf{z}, \mathbf{z}) \\ \mathbf{G}(\mathbf{z}, \mathbf{z}') &= \sum_{t \in \mathrm{TI}_{\mathbf{z},\mathbf{z}'}} w(t) \bigotimes_{i=1}^{K} \mathbf{K}_i(t)(\mathbf{z}, \mathbf{z}') \end{aligned} \tag{2}$$

The steady-state distribution of a stochastic SAM can be computed using the $\mathbf{G}$ matrix given in equation (2).

Instead of using the above result to compute exact indices, the same decomposition technique for a SAM can be used for an approximate throughput computation. The method is, basically, a *response time preservation algorithm*. In each $\mathcal{LS}_i$ a unique module of $\mathcal{S}$ with all its places and transitions is kept. So, in $\mathcal{LS}_i$ interface transitions of module j (for $j \neq i$) approximate the response time of module j. The algorithm is the following [33]:

select the modules
derive the reachability graph of $\mathcal{LS}_i$ for $i = 1, \ldots, K$ and the one of $\mathcal{BS}$
give an initial service rate $\boldsymbol{\mu}_i^{(0)}$ for $i = 2, \ldots, K$

$j := 0$ {counter for iteration steps}
repeat
  $j := j + 1$
  for $i := 1$ **to** K **do**
    solve $\mathcal{LS}_i$: In: $\boldsymbol{\mu}_l^{(j)}$ for $l < i$ and $\boldsymbol{\mu}_l^{(j-1)}$ for $i < l \leq K$
      Out: initial rates $\boldsymbol{\mu}_i$ and thr. $\boldsymbol{\chi}_i^{(j)}$ of $\mathrm{TI} \cap T_i$
    solve $\mathcal{BS}$: In: $\boldsymbol{\mu}_l^{(j)}$ for $l < i$ and $\boldsymbol{\mu}_l^{(j-1)}$ for $i < l \leq K$
      $\boldsymbol{\mu}_i$ and $\boldsymbol{\chi}_i^{(j)}$ for transitions in $\mathrm{TI} \cap T_i$
      Out: actual rates $\boldsymbol{\mu}_i^{(j)}$ $\mathrm{TI} \cap T_i$
until convergence of $\boldsymbol{\chi}_1^{(j)}, \ldots, \boldsymbol{\chi}_K^{(j)}$

The gain of state space decomposition techniques with respect to the classical exact solution algorithm (solution of the underlying CTMC of the original model) is in both memory and time requirements. With respect to space, the infinitesimal generator of the CTMC of the whole system is never stored. Instead of that, the generator matrices of smaller subsystems are stored. With respect to time complexity, in the case of the above iterative approximation algorithm we do not solve the CTMC isomorphous to the original system but those much smaller isomorphous to the derived subsystems.

5 Related work

There exist several works in SPE that inspired the work presented in this article. They share similarities such as: SPE is driven by models, they were proposed before or in the first specific conference of the SPE [44], they began to devise UML as the design notation. Summarizing, they were the first attempts to position SPE as we understand it today. Concretely, we should mention: the Permabase project [45], the works developed at Carleton University [47, 48] and the works by Pooley and King [21, 35].

Another set of works can be considered as contemporary to the one presented here. They explicitly consider UML as the design language for the SPE process and they are more mature than those previously cited since they appeared later in time. Some of them were presented in [46, 18] and others even in posterior relevant conferences or journals. Among them, the following stand out: the works of Cortellessa [11], the software architecture approaches [2], the work of De Miguel [12] and finally the research work at Carleton University [34].

6 Conclusion

In this paper we surveyed recent contributions in the field of SPE using UML and stochastic Petri nets. The UML behavioral diagrams were studied in order to:

1. find the possible roles that they can play in a SPE process,

2. describe the performance annotations that each one can support under these roles,
3. translate them into stochastic Petri nets.

The result of the proposed translation consists in a performance model that represents either the whole system (composition of the LGSPN representing each SC in the system) or a concrete execution of the system (composition of the LGSPN representing a SD togheter with the LGSPNs of the involved SCs). Since the performance model can suffer the state space explosion problem, we also surveyed here several works on efficient analysis techniques that can be used to solve the model.

References

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis, *Modelling with generalized stochastic Petri nets*, John Wiley Series in Parallel Computing - Chichester, 1995.
2. F. Aquilani, S. Balsamo, and P. Inverardi, *Performance analysis at the software architectural design level*, Performance Evaluation **45** (2001), 147–178.
3. S. Bernardi, *Building Stochastic Petri Net models for the verification of complex software systems*, Ph.D. thesis, Dipartimento di Informatica, Università di Torino, April 2003.
4. S. Bernardi, S. Donatelli, and J. Merseguer, *From UML sequence diagrams and statecharts to analysable Petri net models*, Proceedings of the ThirdInternational Workshop on Software and Performance (WOSP2002) (Rome, Italy), ACM, July 2002, pp. 35–45.
5. G. Berthelot, *Transformations and decompositions of nets*, Advances in Petri Nets 1986 - Part I (W. Brauer, W. Reisig, and G. Rozenberg, eds.), Lecture Notes in Computer Science, vol. 254, Springer-Verlag, Berlin, 1987, pp. 359–376.
6. F.P. Brooks, *Essence and accidents of software engineering*, IEEE Computer **20** (1987), no. 4, 10–19.
7. J. Campos, G. Chiola, and M. Silva, *Properties and performance bounds for closed free choice synchronized monoclass queueing networks*, IEEE Transactions on Automatic Control **36** (1991), no. 12, 1368–1382.
8. J. Campos, S. Donatelli, and M. Silva, *Structured solution of asynchronously communicating stochastic modules*, IEEE Transactions on Software Engineering **25** (1999), no. 2, 147–165.
9. G. Chiola, C. Anglano, J. Campos, J. M. Colom, and M. Silva, *Operational analysis of timed Petri nets and application to the computation of performance bounds*, Proceedings of the 5^{th} International Workshop on Petri Nets and Performance Models (Toulouse, France), IEEE-Computer Society Press, October 1993, pp. 128–137.
10. J. M. Colom and M. Silva, *Improving the linearly based characterization of P/T nets*, Advances in Petri Nets 1990 (G. Rozenberg, ed.), Lecture Notes in Computer Science, vol. 483, Springer-Verlag, Berlin, 1991, pp. 113–145.
11. V. Cortellessa and R. Mirandola, *Deriving a queueing network based performance model from UML diagrams*, in Woodside et al. [46], pp. 58–70.

12. M. de Miguel, T. Lambolais, M. Hannouz, S. Betge, and S. Piekarec, *UML extensions for the specification of latency constraints in architectural models*, in Woodside et al. [46], pp. 83–88.
13. E. Domínguez, A.L. Rubio, and M.A. Zapata, *Dynamic semantics of UML state machines: A metamodelling perspective*, Journal of Database Management **13** (2002), 20–38.
14. R. Dumke, C. Rautenstrauch, A. Schmietendorf, and A. Scholz (eds.), *Performance engineering. state of the art and current trends*, Lecture Notes in Computer Science, vol. 2047, Springer-Verlag, Heidelberg, 2001.
15. H. Gomaa and D. Menascé, *Design and performance modeling of component interconnection patterns for distributed software architectures*, in Woodside et al. [46], pp. 117–126.
16. *The GreatSPN tool*, `http://www.di.unito.it/~greatspn`.
17. H. Hermanns, U. Herzog, and Katoen J.P., *Process algebra for performance evaluation*, Theoretical Computer Science **274** (2002), no. 1-2, 43–87.
18. P. Inverardi, S. Balsamo, and Selic B. (eds.), *Proceedings of the Third International Workshop on Software and Performance*, Rome, Italy, ACM, July 24-26 2002.
19. I. Jacobson, M. Christenson, P. Jhonsson, and G. Overgaard, *Object-oriented software engineering: A use case driven approach*, Addison-Wesley, 1992.
20. K. Kant, *Introduction to computer system performance evaluation*, Mc Graw-Hill, 1992.
21. P. King and R. Pooley, *Using UML to derive stochastic Petri nets models*, Proceedings of the Fifteenth Annual UK Performance Engineering Workshop (J. Bradley and N. Davies, eds.), Department of Computer Science, University of Bristol, July 1999, pp. 45–56.
22. J. P. López-Grao, J. Merseguer, and J. Campos, *From UML activity diagrams to stochastic PNs: Application to software performance engineering*, Proceedings of the Fourth International Workshop on Software and Performance (WOSP'04) (Redwood City, California, USA), ACM, January 2004, To appear.
23. J. Merseguer, *Software performance engineering based on UML and Petri nets*, Ph.D. thesis, University of Zaragoza, Spain, March 2003.
24. J. Merseguer, S. Bernardi, J. Campos, and S. Donatelli, *A compositional semantics for UML state machines aimed at performance evaluation*, Proceedings of the 6th International Workshop on Discrete Event Systems (Zaragoza, Spain) (A. Giua and M. Silva, eds.), IEEE Computer Society Press, October 2002, pp. 295–302.
25. J. Merseguer and J. Campos, *Exploring roles for the UML diagrams in software performance engineering*, Proceedings of the 2003 International Conference on Software Engineering Research and Practice (SERP'03) (Las Vegas, Nevada, USA), CSREA Press, June 2003, pp. 43–47.
26. J. Merseguer, J. Campos, and E. Mena, *A pattern-based approach to model software performance*, in Woodside et al. [46], pp. 137–142.
27. ———, *A performance engineering case study: Software retrieval system*, in Dumke et al. [14], pp. 317–332.
28. ———, *Analysing internet software retrieval systems: Modeling and performance comparison*, Wireless Networks: The Journal of Mobile Communication, Computation and Information **9** (2003), no. 3, 223–238.
29. M. K. Molloy, *Performance analysis using stochastic Petri nets*, IEEE Transactions on Computers **31** (1982), no. 9, 913–917.
30. Object Management Group, `http:/www.omg.org`.
31. Object Management Group, `http:/www.omg.org`, *UML Profile for Schedulabibity, Performance and Time Specification*, March 2002.

32. Object Management Group, `http:/www.omg.org`, *OMG Unified Modeling Language specification*, March 2003, version 1.5.
33. C. J. Pérez-Jiménez and J. Campos, *On state space decomposition for the numerical analysis of stochastic Petri nets*, Proceedings of the 8^{th} International Workshop on Petri Nets and Performance Models (Zaragoza, Spain), IEEE Computer Society Press, September 1999, pp. 32–41.
34. D. Petriu and H. Shen, *Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications*, Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002 (London, UK) (Tony Field, Peter G. Harrison, Jeremy Bradley, and Uli Harder, eds.), Lecture Notes in Computer Science, vol. 2324, Springer, April 14-17 2002, pp. 159–177.
35. R. Pooley and P. King, *The Unified Modeling Language and performance engineering*, IEE Proceedings Software, IEE, February 1999, pp. 2–10.
36. ArgoUML project, `http://argouml.tigris.org`.
37. J. Rumbaugh, M. Blaha, W. Premerlani, E. Frederick, and W. Lorensen, *Object oriented modeling and design*, Prentice-Hall, 1991.
38. A. Schmietendorf and A. Scholz, *Aspects of performance engineering - An overview*, in Dumke et al. [14], pp. IX–XII.
39. M. Silva, *Las redes de Petri en la automática y la informática*, Editorial AC, Madrid, 1985, In Spanish.
40. C. U. Smith, *Performance engineering of software systems*, The Sei Series in Software Engineering, Addison–Wesley, 1990.
41. C.U. Smith, *Increasing information systems productivity by software performance engineering*, Proceedings of the Seventh International Computer Measurement Group Conference (New Orleans, LA, USA) (D.R. Deese, R.J. Bishop, J.M. Mohr, and H.P. Artis, eds.), Computer Measurement Group, December 1-4 1981, pp. 5–14.
42. ———, *Origins of software performance engineering: Highlights and outstanding problems*, in Dumke et al. [14], pp. 96–118.
43. C.U. Smith and Ll.G. Williams, *Software performance antipatterns*, in Woodside et al. [46], pp. 127–136.
44. C.U. Smith, M. Woodside, and P. Clements (eds.), *Proceedings of the First International Workshop on Software and Performance*, Santa Fe, New Mexico, USA, ACM, October 12-16 1998.
45. P. Utton and B. Hill, *Performance prediction: an industry perspective*, Performance Tools 97 Conference (St Malo), June 1997, pp. 1–5.
46. M. Woodside, H. Gomaa, and D. Menascé (eds.), *Proceedings of the Second International Workshop on Software and Performance*, Ottawa, Canada, ACM, September 17-20 2000.
47. M. Woodside, C. Hrischuck, B. Selic, and S. Bayarov, *A wideband approach to integrating performance prediction into a software design environment*, in Smith et al. [44], pp. 31–41.
48. ———, *Automated performance modeling of software generated by a design environment*, Performance Evaluation **45** (2001), 107–123.