On the Integration of UML and Petri Nets in Software Development*

Javier Campos and José Merseguer

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

Abstract. Software performance engineering deals with the consideration of quantitative analysis of the behaviour of software systems from the early development phases in the life cycle. This paper summarizes in a semiformal and illustrative way our proposal for a suitable software performance engineering process. We try to integrate in a very pragmatic approach the usual object oriented methodology —supported with UML language and widespread CASE tools— with a performance modelling formalism, namely stochastic Petri nets. A simple case study is used to describe the whole process. More technical details should be looked up in the cited bibliography.

1 Introduction

The design and implementation of complex systems is a difficult engineering task. In the last years the modelling, validation, performance evaluation and implementation of such systems has been usually tackled with the help of formal models.

Petri nets (PNs) [1] is an adequate formal paradigm to support the whole life-cycle engineering of a complex discrete event system. They have been used for the modelling and evaluation of flexible manufacturing systems [2], multiprocessor architectures [3], communication systems [4], and also for the writing of reliable and efficient concurrent programs [5].

Software systems are complex systems, probably the most complex construction tasks that humans undertake. Functional requirements of software applications are obviously important, but they are not the only concern. Performance objectives are also important: the degree to which a software system meets its objectives for timeliness is important in many cases and it is critical in some real-time applications.

Being software engineering a relatively young discipline, the importance of the use of well established methodologies, even formal methods, languages and tools has already been detected and assumed. Unfortunately, performance objectives

* This work has been developed within the projects TIC2003-05226 of the Spanish Ministry of Science and Technology and IBE2005-TEC-10 of the University of Zaragoza.

S. Donatelli and P.S. Thiagarajan (Eds.): ICATPN 2006, LNCS 4024, pp. 19–36, 2006.

are still not usually included in first stages of the software life cycle. Being performance requirements critical to the success of today's software systems, many final software products fail to meet those requirements. The usual practice, as nicely summarized in [6], consists on *make it run, then make it run right, and finally make it run fast*. But this practice is in many cases too expensive, because fixing performance problems can oblige to modify the initial design.

Just to illustrate the previous statement with a single famous crisis, we recall the Denver airport baggage system story. It was planned for the United Airlines terminal, but during the development it was enlarged to support all the airport terminals but without considering the new system's workload. As a result of the inadequate performance characteristics of the system, the airport was opened 16 months later with a loss of hundreds of million dollars [7].

Therefore, many researchers defend the principle that performance should be included in the software design process from the very beginning. This principle is one of the main goals of the *International Workshop on Software and Performance*, a forum for researchers interested in the intersection of software engineering and performance evaluation that started in 1998 and has already held five editions. The research field that deals with the goal of building software with predictable performance by specifying and analysing quantitative behaviour from the early development phases of a system throughout its entire life cycle has been coined with the term *Software Performance Engineering* (SPE) [8].

The Unified Modelling Language (UML) [9] combined with an object oriented methodology, such as [10], is nowadays the most widely used approach in the software engineering community. Thus, most of Computer Aided Software Engineering (CASE) tools support OO methodology and use UML as the design language.

Since performance goals are not included in the usual practice of software engineers, we think that there exists a need of integrating performance modelling with the existing software development methodologies and tools. Markov models, queueing networks, stochastic process algebras and stochastic PNs are probably the best studied performance modelling paradigms. Among all of them, we bet on PNs due to its special adequacy to model parallel and distributed systems, its mathematical simplicity, its modelling generality, its adequacy for expressing all basic semantics of concurrency, its locality both of states and actions, its graphical representation, its well-developed qualitative and quantitative analysis techniques, and the existence of analysis tools.

This paper tries to present in a semiformal and illustrative way, some of our experience in the process of integration of performance modelling within software development process. In section 2 we summarize the main phases of a suitable software performance engineering process based on UML and stochastic PNs. Section 3 is devoted to a more detailed presentation of the process by means of a simple case study. A basic client for checking mail from a server using POP3 protocol is described. The system is modelled using UML language. In particular, use case diagram, statecharts diagrams, sequence diagram, activity diagram and the deployment diagram are considered. Each diagram is annotated with timing

information according to the UML *Profile for Schedulability, Performance and Time* [11] of the Object Management Group (OMG). For each of the annotated UML diagram, the highlights of the translation into a corresponding stochastic PN is presented. Also some examples of performance figures that can be derived from the analysable model are presented. The paper ends with some concluding remarks in section 4.

2 Software Performance Process

Several works have been proposed to combine UML and a performance modelling formalism to analyze quantitative aspects of software systems. All of them share some basic principles. Then it could be argumented that there exists a widely accepted *process* among the SPE community:

- The behaviour and architecture of the system is described by a (set of) UML diagram(s), that make up the *system design*.
- This UML design is annotated according to a standard OMG profile. Then gaining the *annotated design*.
- The annotated design is converted into a *performance modelling formalism*.
- A qualitative analysis will be carried out, if the formal model allows it.
- The formal model is analyzed using quantitative analysis techniques already developed for the target formalism.

Now, we reveal some key aspects of the process.

Use cases (UCs) use to be the starting point to describe the system behaviour. They are used to specify the requirements of a system, subsystem or class and for the specification of their functionality. Sequence diagrams (SDs) or activity diagrams (ADs) are the common choice to detail UCs. SDs specify a set of partially ordered messages, each one defining a communication mechanism as well as the roles to be played by the sender and the receiver. Thus SDs represent patterns of interaction between objects. ADs represent internal control flow of processes. SDs and ADs are useful to conduct performance-based scenario analysis. Few approaches introduce the statecharts (SCs) to complement them. SCs, as ADs, are used to describe the behaviour of a model element. The software architecture completes the *system design* by means of the deployment diagram (DD), then modeling the distribution of the software components in the hardware platform.

Workloads, utilizations, response times or throughput characterize the performance view of the system design, then gaining an *annotated design*. The UML profile for schedulability, performance and time specification (UML-SPT) [11] is the widely used OMG standard to annotate them. The OMG QoS profile [12] is also used for these purposes. Actually, the convergence of both profiles is expected.

Three paradigms have been mainly used as *performance modeling formalisms* in the SPE process: Stochastic Petri nets (SPNs), stochastics process algebras (SPA) and (layered) queuing networks (LQN). Several methods have been proposed to translate the UML syntax into the syntax of the target formalism.

Among the translations using SPNs as target are [13, 14, 15], using SPA [16], using QN [17, 18] and LQN [19, 20]. Such methods use different approches and technologies: Customization Rules [21], XSL transformations [19], algebra-based transformations [22], direct formalization [23] or Graph Grammars [20]. For a survey of such translations see [24].

Tools are essential for SPE. The preferred approach is to integrate performance aspects into existing CASE tools. Figure 1 depicts the OMG framework to develop performance evaluation tools following the *SPE process*.

During the last decades researchers developed techniques to analyze performance models. They were implemented in analysis tools [25, 26, 27] that the engineer uses to analyze the resulting models. Moreover these analysis tools are used as the core of the CASE tools [28, 29, 30, 31, 32] that follow the proposal in Figure 1.

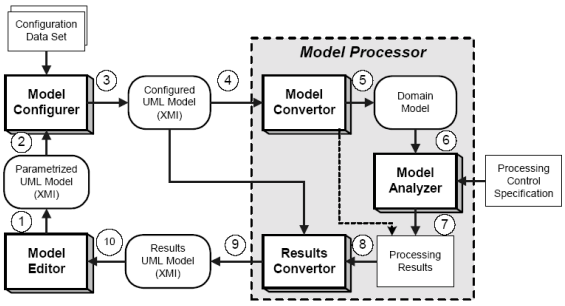

Fig. 1. OMG architectural framework for SPE tools

The authors have developed some work in the SPE field following the process so far outlined. Now, we recall how each process step is considered in the proposal here recalled:

- Concerning the UML diagrams, we consider: Use cases, sequence diagrams, state machines, activity diagrams and the deployment diagram [33].
- They are annotated according to the UML-SPT. Table 1 summarizes the subset of annotations currently supported.
- SPNs are the target of the translation methods proposed in [34] for SMs, in [35] for SDs and in [36] for ADs.
- ArgoSPE [37] provides tool support for the proposal.

This proposal is illustrated, in section 3, through the development of a case study. It is worth to notice that today a big SPE challenge is about how to integrate the bunch of UML diagramatic notations and the performance formalisms. So, our approach has evolved and now fits in the PUMA framework [32]. It defines an intermediate performance model, the core scenario model (CSM) [38], to encompass this challenge. Other works complement the PUMA approach [39].

Table 1. Performance annotations

Stereotype	Tag	Diagram	Kind*; Type	Comment
PAcontext	-	SD,SC,AD	-;-	Performance model
PAclosedLoad	PArespTime	SD	pred.;time-value	Execution time
		SC		Object lifetime
		AD		Execution time
	PApopulation	SC	-; nat-number	Number of instances
PAstep	PAdemand	SD,SC,AD	assm.; time-value	Msg. transm. delay
		SC,AD		Activity duration
	PArespTime	SC	pred.;time-value	State exec. time
		AD		Trans. exec. time
	PAprob	SD	-; real	Condit. msg; Branch
		SC,AD		Guarded transition
	PArep	SD	-; nat-number	Iterated msg
	PAsize	SD,SC,AD	assm.; nat-numb.	Msg size
	PAthroughput	SC	pred.; -	Transition throughput
PAcommun.	PAspeed	DD	assm.; Kbytes	Network speed
PAinitialCond.	PAinitialState	SC	bool	Initial state
GRMcode	-	DD	-	Resident classes
(*)predicted or assumed				

3 Case Study

The process outlined in section 2 has been applied in different software domains. For example in [40] to test the design of a mobile agents system or in [41] to study the QoS of fault tolerant systems distributed over Internet. In this paper, we recall a case study [42] that was useful to show the feasibility of the proposal. Now, we come back to that work to illustrate some aspects of the SPE process.

The case study models a basic mail client. Here we will focus in the first Use Case (UC) showed in Figure 2: checking mail from a server using the POP3 [43] protocol.

The behavior of the referred client is rather intuitive for this UC. The SC in Figure 3 depicts the behaviour of a mail client, it can be useful to get a first view of the general system behaviour. The client tries to establish a TCP connection with the server via port 110. If it succeeds (reception of a greeting message), both (client and server) begin the authentication (authorization) phase. The client sends the username and his/her password through a USER and PASS

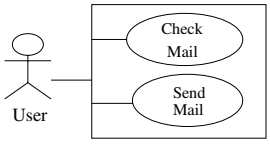

Fig. 2. Use Case view of the 'mail client' model

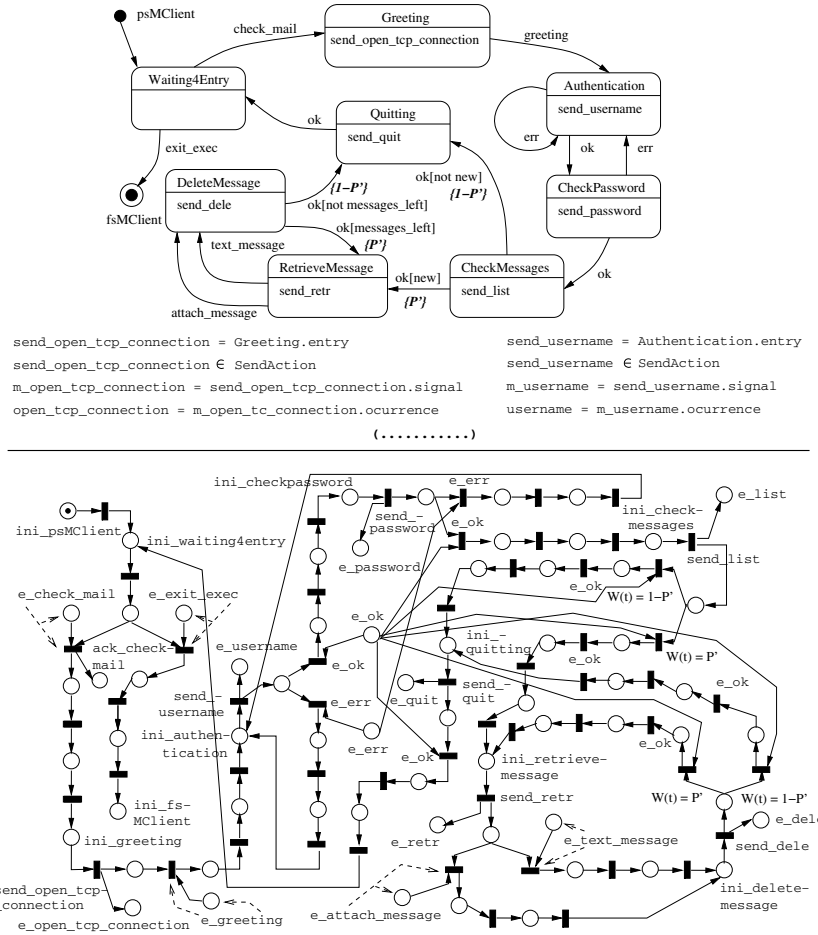

Fig. 3. Statechart and resulting LGSPN for the dynamics of the class ClientHost

command combination. For the sake of simplicity, usage of the APOP command has not been contemplated here.

If the server has answered with a positive status indicator ("+OK") to both messages, then the POP3 session enters the transaction state (phase). Otherwise (e.g., the password doesn't match the one specified for the username), it returns to the beginning of the authorization phase.

In the transaction phase, the client checks for new mail using the LIST command. If there is any, the client obtains every e-mail by means of the RETR and DELE commands. It must be noted that, for simplicity, potential errors have not been considered here; thus, no negative status messages ("-ERR") are modelled.

Once every e-mail has been downloaded, the mail client issues a QUIT command to end the interaction. This provokes the POP3 server to enter the update

state and release any resource acquired during the transaction phase. The protocol is ended with a goodbye ("+OK") message.

3.1 Statechart Modeling

The mail client behavior (MailClient class) has been modeled using an SC, see Figure 3. Similarly, Figure 4 illustrates the server host and user behaviors via two SCs describing the POP3Server class and User actor dynamics.

UML statecharts are used in this context to describe the behavior of a model element, concretely classes. 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 [44]. Studies about their semantics can be found in [34, 45].

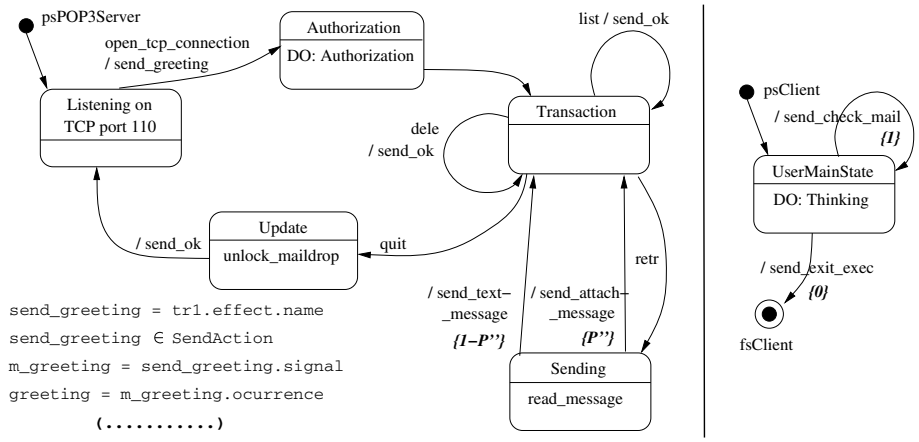

Fig. 4. Statecharts for the dynamics of the classes ServerHost and User (actor)

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. See state Waiting4Entry in Figure 3, where the mail client is waiting either for the event check_mail or the event exit_exec. An event is a noteworthy occurrence that may trigger a state transition [44]. A simple transition is a relationship between two states indicating that an object in the first state will enter the second state.

The approach taken to translate a statechart into an Labelled Generalized Stochastic Petri Net [46] (LGSPN) consists in the translation of each element in the metamodel of the state machines into an LGSPN subsystem. Just to take the flavor of the translation, Figure 5 depicts the LGSPNs subsystems obtained by the translation of a simple state.

An important aspect of the translation concerns with the activities. Since they consum processing time, then they will be translated as timed transitions. The

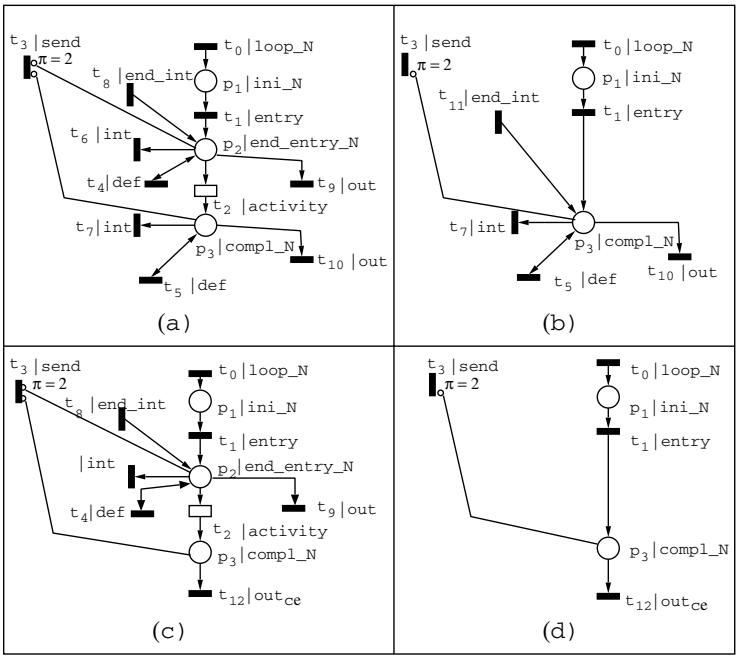

Fig. 5. Different labelled "basic" systems for a simple state N: (a) with activity and no immediate outgoing transition; (b) no activity and no immediate outgoing transition; (c) with activity and immediate outgoing transition; (d) no activity and with immediate outgoing transition

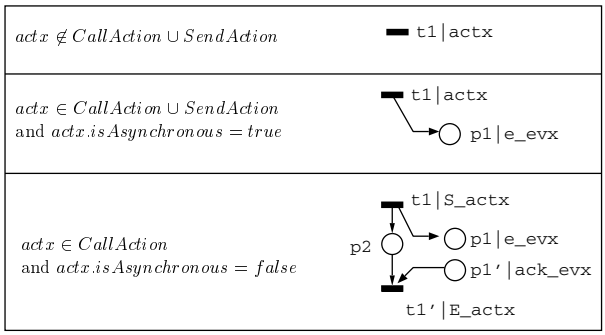

Fig. 6. Translation of the different types of actions

delay of such transitions, i.e. the mean of a exponentially distributed ramdom variable, is calculated as the inverse of such processing time. The translation of the actions is given in Figure 6. The translation of the other elements in the metamodel can be found in [23, 34].

Not-event-driven decisions are modelled using guards with its success probability. Concretely, a combination of guards and events has been used in some SC transitions. That probability will be represented in the Petri net using an immediate 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 a composition operator defined in [46]. Basically, this operator composes two LGSPNs into a third one by fusing the places and transitions with equal labels. The details of the composition method can be found also in [23, 34].

Figure 3 shows the LGSPN for the MailClient SC. It has been obtained applying the transformation method for SCs given in [23] and briefly recalled here. The LGSPNs for the ServerHost and the User are not given to avoid a cumbersome presentation.

3.2 Activity Diagram

An in-depth study of our mail client showed that the activity Authenticate, associated to the state Authorization in the SC for ServerHost (Figure 4), was relevant to the system performance. Moreover, it would be necessary a more accurate modelling of its behaviour to complete the system description. To fill this gap, the SPE process proposes to model the actions performed within the states of the SC, concretely by using the activity diagram (AD).

Thus, we used an AD (see Figure 7) to model the behaviour of the Authenticate activity. Although it may be more useful in cases where there is not such a strong external event dependence (e.g., 'internal' operations). The activity could have been described extending the SC but, in general, ADs provide some additional expresiveness for certain tasks.

Activity diagrams (ADs) represent UML activity graphs and are just a variant of UML state machines (see [44]), 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.

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. See the annotations PAprob and PArespTime in Table 1.

Moreover, the AD will be annotated with the size of the messages, PAsize tag, when it models event-driven behaviour. The closed load will be modelled attaching to the initial state the PApopulation tag. Table 1 summarizes the annotations proposed for the AD.

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 [34].

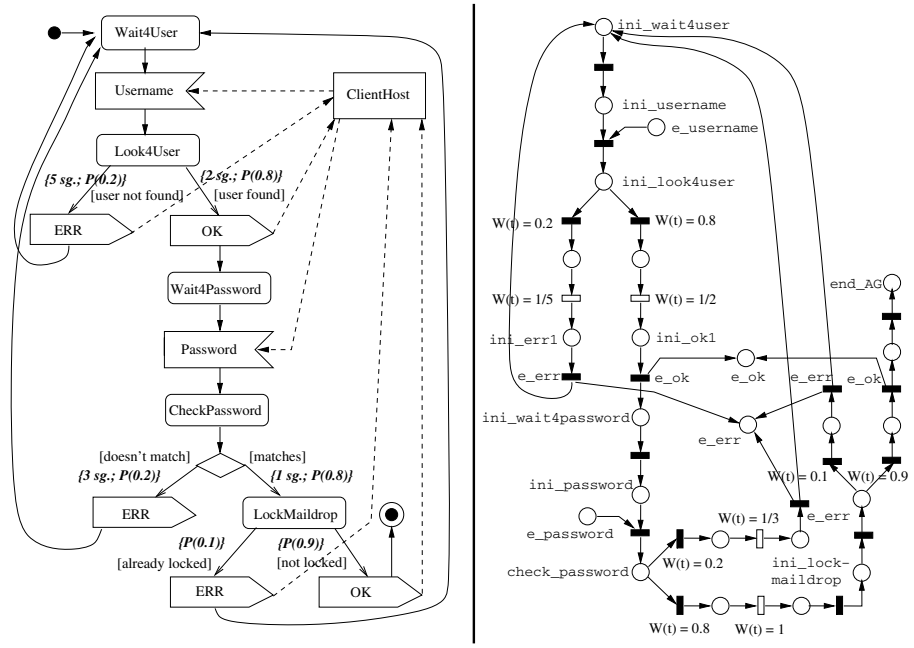

Fig. 7. Activity Diagram for POP3ServerHost::Authenticate and resulting LGSPN

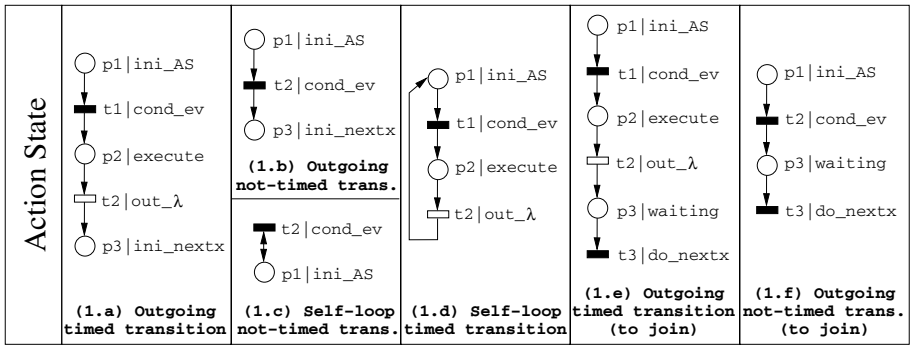

Fig. 8. Translation of the Action States in ADs

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.

The performance model obtained from an AD in terms of LGSPNs as proposed in [36] 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 full translation process can be found in [36], Figure 8 offers its flavour by depicting the action states translation.

Other interpretations for the AD propose it as a high level modeling tool, that of the workflow systems, but the SPE approach presented here does not consider this role.

3.3 Sequence and Deployment Diagrams

A sequence diagram (SD) 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 [44]. The deployment diagram (DD) models the distribution of the software components in the hardware platform and the operative system resources.

In this SPE process 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. While the DD will be useful to deploy each class (that has been modeled with a SC) in a hardware node.

From the performance point of view the SD and the DD have relevant elements and constructions. They will be annotated, according to Table 1. In the following these elements and constructs are explained.

Objects can be executed in the same hardware or in different ones 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 *travelling through the net*, it is considered that they spend time, then representing a load for the system that should be annotated in this diagram. In the second case, it is also possible to annotate in the SD the size of the message and in the DD the bit rate at which the network operates. Then, the load for each message can be easily obtained from the SD and the DD annotations. See Figure 9 for the size annotation. Figure 10 annotates the bit rate with a variable $TR. This transfer rate will take different values in the analysis step.

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.

In [35], the translation process of a given sequence diagram *sd* into its corresponding LGSPN $\mathcal{LS}_{sd}$ can be found.

Finally, we use SDs to obtain performance analytical measures in a certain context of execution. Figure 9 shows an example of interaction between both server and client. Some results for this particular scenario will be obtained in the next subsection.

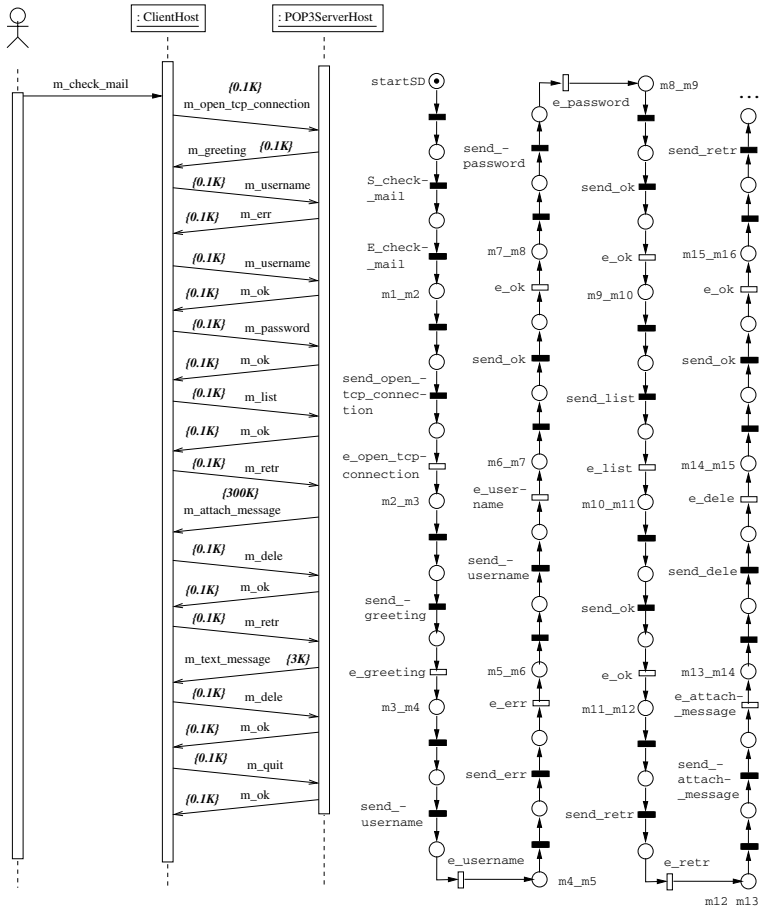

Fig. 9. Sequence Diagram describing scenario, and corresponding LGSPN

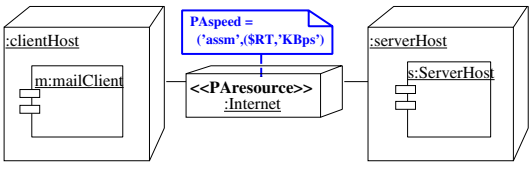

Fig. 10. Deployment Diagram

3.4 Analysis

Once the final LGSPN models are obtained (following the composition rules detailed in [34, 35]) performance estimates can be computed. These figures can be related to either the whole system behavior (somehow unrestricted) or the system behavior in a concrete scenario (thus adjusted to certain restrictions).

Therefore, 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:

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

The works in [34, 36, 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:

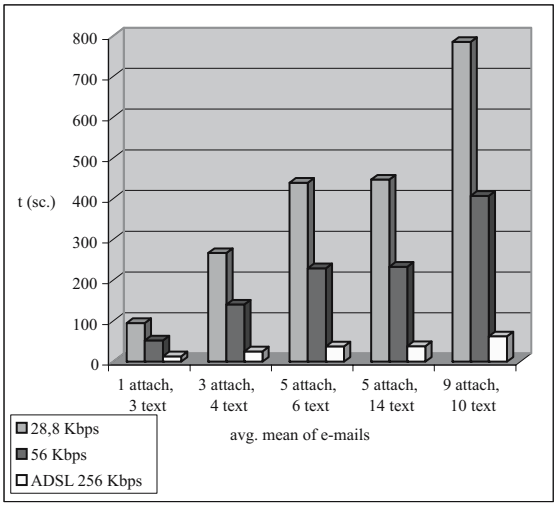

Fig. 11. Effect of number of mails and attached file proportion on the downloading time for different connection speed

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

The works in [35, 47] detail the composition method.

Figure 11 shows the effect of the number of mails and different proportion of them with attached files on the downloading time for different connection speeds. Bars labelled with "a attach - b text" correspond with the case where the average number of downloaded mails is $a + b$ and an average of a of them have attached files.

The graph in Figure 12 represents the effective transfer rate of the client when checking mail (maximum transfer rate: 56 Kbps). Note that higher amounts of data minimize the relative amount of time spent by protocol messages. The analysis has been taken considering the whole system behavior (that is, using the net obtained by composition of the ones corresponding to the SCs and the AD).

Meanwhile, the graph in Figure 13 represents the time cost of executing the interaction illustrated in figure 9 in function of different attach file sizes and maximum network speeds. The analysis has been taken using the SD to construct the net for the constrained case [35]. In general, SDs can be extremely useful to check the behaviour of the system for a particular use case. Moreover, analysts may use them to model test conditions in an easy way.

In another paper included in this volume [37], ArgoSPE tool is presented. It implements the features explained here for the software performance modelling process. So, the system is modeled as a set of UML diagrams, annotated according to the UML-SPT, which are translated into LGSPN. The UML diagrams used to obtain a performance model by means of ArgoSPE are those considered in our process: statecharts, activity diagrams and sequence diagrams. The class and the implementation diagrams (components and deployment) are used to collect some system parameters (system population or network speed).

ArgoSPE has been implemented as a set of Java modules, that are plugged into the open source ArgoUML CASE tool [48]. It follows the software architecture

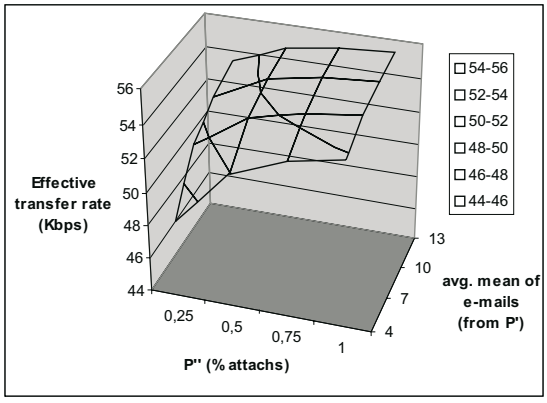

Fig. 12. Effective transfer rate of the client when checking mail

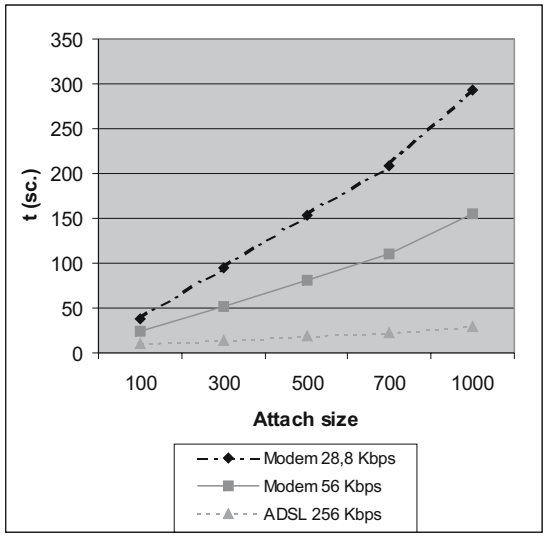

Fig. 13. Some analytical results for the presented case study

proposed in the UML-SPT, see Figure 1. ArgoSPE has been used to model and analyze software fault tolerant systems [41] and mobile agents software [40].

4 Conclusions

This paper has presented the application of a SPE process through a case study. Such SPE process has as relevant features the use of some UML diagrams and stochastic Petri nets. The UML behavioral diagrams (statecharts, activity and sequence) togheter with the deployment diagram allow to model system functionality and describe system performance characteristics. While the stochastic Petri nets are used as a performance modelling formalism to analyze quantitative aspects of the system.

The SPE process proposes a translation of the UML models into the performance model. These final models represent 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). The SPE process has support through a CASE tool.

References

1. Murata, T.: Petri nets: Properties, analysis, and applications. Proceedings of the IEEE **77**(4) (1989) 541–580
2. DiCesare, F., Harhalakis, G., Proth, J.M., Silva, M., Vernadat, F.: Practice of Petri Nets in Manufacturing. Chapman & Hall, London (1993)

3. Ajmone Marsan, M., Balbo, G., Conte, G.: Performance Models of Multiprocessor Systems. MIT Press, Cambridge, Massachussetts (1986)
4. Billington, J., Diaz, M., Rozenberg, G., eds.: Application of Petri Nets to Communication Networks. Number 1605 in Lecture Notes in Computer Science, Advances in Petri Nets. Springer-Verlag (1999)
5. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. John Wiley Series in Parallel Computing - Chichester (1995)
6. Smith, C., Williams, L.: Software Performance Engineering. In: UML for Real: Design of Embedded Real-Time Systems. Kluwer (2003)
7. Gibbs, W.: Trends in computing: Software's chronic crisis. Scientific American **271**(3) (1994) 72–81
8. Smith, C.U.: Performance Engineering of Software Systems. The Sei Series in Software Engineering. Addison–Wesley (1990)
9. Booch, G., Jacobson, I., Rumbaugh, J.: The Unified Modeling Language. Addison Wesley (1999)
10. Jacobson, I., Christenson, M., Jhonsson, P., Overgaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992)
11. Object Management Group `http:/www.omg.org`: UML Profile for Schedulabibity, Performance and Time Specification. (2005)
12. Object Management Group `http:/www.omg.org`: UML Profile for Modeling Quality of Service and Fault Tolerant Characteristics and Mechanisms. (2004)
13. Bondavalli, A., Dal Cin, M., Latella, D., Majzik, I., Pataricza, A., Savoia, G.: Dependability analysis in the early phases of UML-based system design. International Journal of Computer Systems Science & Engineering **16**(5) (2001) 265–275
14. Pettit IV, R., Gomaa, H.: Modeling Behavioral Patterns of Concurrent Software Architectures Using Petri Nets. In: 4th Working IEEE / IFIP Conference on Software Architecture (WICSA), Oslo, Norway, IEEE Computer Society (2004) 57–68
15. Saldhana, J., Shatz, S.: UML Diagrams to Object Petri Net Models: An Approach for Modeling and Analysis . In: Twelfth International Conference on Software Engineering and Knowledge Engineering, Chicago, IL, USA, Knowledge Systems Institute (2000) 103–110
16. Canevet, C., Gilmore, S., Hillston, J., Kloul, L., Stevens, P.: Analysing UML 2.0 activity diagrams in the software performance engineering process. In: Proceedings of the 4th International Workshop on Software Performance (WOSP'04), Redwood Shores, California, USA, ACM (2004) 74–78
17. Cortellessa, V., Mirandola, R.: Deriving a queueing network based performance model from UML diagrams. In: Proceedings of the Second International Workshop on Software and Performance (WOSP2000), Ottawa, Canada, ACM (2000) 58–70
18. Smith, C.U., Williams, L.: Performance Solutions. Addison–Wesley (2002)
19. Gu, G., Petriu, D.: XSLT transformation from UML models to LQN performance models. In: Proceedings of the Third International Workshop on Software and Performance (WOSP2002), Rome, Italy, ACM (2002) 25–34
20. Petriu, D., Shen, H.: Applying the UML performance profile: Graph grammar-based derivation of LQN models from UML specifications. In Field, T., Harrison, P.G., Bradley, J., Harder, U., eds.: Computer Performance Evaluation, Modelling Techniques and Tools 12th International Conference, TOOLS 2002. Volume 2324 of Lecture Notes in Computer Science., London, UK, Springer (2002) 159–177

21. Baresi, L., Pezzè, M.: 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. Volume 2001 of Advances in Petri Nets. Lecture Notes in Computer Science, (LNCS). Springer-Verlag, Heidelberg (2001) 276–304
22. Gu, G., Petriu, D.: From UML to LQN by XML algebra-based model transformations. In: Proceedings of the Fifth International Workshop on Software and Performance (WOSP2005), Palma, Spain, ACM (2005) 99–110
23. Merseguer, J.: Software Performance Engineering based on UML and Petri nets. PhD thesis, University of Zaragoza, Spain (2003)
24. Balsamo, S., Di Marco, A., Inverardi, P., Simeoni, M.: Model-based performance prediction in software development: a survey. IEEE Transactions on Software Engineering **30**(5) (2004) 295–310
25. The GreatSPN tool (`http://www.di.unito.it/~greatspn`)
26. The TimeNET tool (`http://pdv.cs.tu-berlin.de/~timenet/`)
27. The Möbius Tool (`http://www.mobius.uiuc.edu/`)
28. ArgoSPE: (`http://argospe.tigris.org`)
29. Argo Performance: (`http://argoperformance.tigris.org`)
30. Cortellessa, V., Gentile, M., Pizzuti, M.: Xprit: An xml-based tool to translate uml diagrams into execution graphs and queueing networks. In: 1st International Conference on Quantitative Evaluation of Systems (QEST 2004). (2004) 342–343
31. Marzolla, M., Balsamo, S.: UML-PSI: the UML Performance SImulator. (In: 1st International Conference on Quantitative Evaluation of Systems (QEST 2004)) 340–341
32. Woodside, M., Petriu, D., Petriu, D., Shen, H., Israr, T., Merseguer, J.: Performance by unified model analysis (PUMA). In: Fifth International Workshop on Software and Performance (WOSP'05), Palma, Spain, ACM (2005) 1–12
33. Merseguer, J., Campos, J.: Exploring roles for the UML diagrams in software performance engineering. In: Proceedings of the 2003 International Conference on Software Engineering Research and Practice (SERP'03), Las Vegas, Nevada, USA, CSREA Press (2003) 43–47
34. Merseguer, J., Bernardi, S., Campos, J., Donatelli, S.: A compositional semantics for UML state machines aimed at performance evaluation. In Giua, A., Silva, M., eds.: Proceedings of the 6th International Workshop on Discrete Event Systems, Zaragoza, Spain, IEEE Computer Society Press (2002) 295–302
35. Bernardi, S., Donatelli, S., Merseguer, J.: From UML sequence diagrams and statecharts to analysable Petri net models. In: Proceedings of the Third International Workshop on Software and Performance (WOSP2002), Rome, Italy, ACM (2002) 35–45
36. López-Grao, J., Merseguer, J., Campos, J.: From UML activity diagrams to stochastic Petri nets: Application to software performance engineering. In: Proceedings of the Fourth International Workshop on Software and Performance (WOSP'04), Redwood City, California, USA, ACM (2004) 25–36
37. Gómez Martínez, E., Merseguer, J.: ArgoSPE: Model-based software performance engineering. In: 27th International Conference on Application and Theory of Petri Nets and Other Models Of Councurrency, LNCS (2006) In this volume.
38. Petriu, D., Woodside, M.: A metamodel for generating performance models from uml designs. In: Proc. UML 2004. Volume 3273 of LNCS., Lisbon, Portugal, Springer-Verlag (2004) 41–53

39. Grassi, V., Mirandola, R., Sabetta, A.: 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). (2005) 25–36
40. Merseguer, J., Campos, J., Mena, E.: Analysing internet software retrieval systems: Modeling and performance comparison. Wireless Networks: The Journal of Mobile Communication, Computation and Information **9**(3) (2003) 223–238
41. Bernardi, S., Merseguer, J.: QoS assesment of fault tolerant applications via stochastics analysis. IEEE Internet Computing (2006) To appear.
42. López-Grao, J., Merseguer, J., Campos, J.: Performance engineering based on UML and SPNs: A software performance tool. In: Proceedings of the Seventeenth International Symposium On Computer and Information Sciences (ISCIS XVII), Orlando, Florida, USA, CRC Press (2002) 405–409
43. Myers, J., Rose, M.: RFC 1725: Post Office Protocol - version 3 (1994)
44. Object Management Group `http:/www.omg.org`: OMG Unified Modeling Language Specification. (2003) version 1.5.
45. Domínguez, E., Rubio, A., Zapata, M.: Dynamic semantics of UML state machines: A metamodelling perspective. Journal of Database Management **13** (2002) 20–38
46. Donatelli, S., Franceschinis, G.: PSR Methodology: integrating hardware and software models. In Billington, J., Reisig, W., eds.: Proceedings of the 17th International Conference on Application and Theory of Petri Nets. Volume 1091 of Lecture Notes in Computer Science., Osaka, Japan, Springer (1996) 133–152
47. Bernardi, S.: Building Stochastic Petri Net models for the verification of complex software systems. PhD thesis, Dipartimento di Informatica, Università di Torino (2003)
48. ArgoUML project: (`http://argouml.tigris.org`)