Performance Engineering based on UML & SPN's: A software performance tool*

Juan Pablo López-Grao, José Merseguer, Javier Campos
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
Zaragoza, Spain
{jpablo,jmerse,jcampos}@posta.unizar.es

Abstract

The increasing relevance of UML as a semi-formal modelling paradigm has entailed the need for an adjustment of the classical performance evaluation methods within the scope of the new working environment. Under these circumstances, a formal semantics for the UML language and a strong mathematical substratum are required in order to be able to compute performance estimates and validate logical properties in the first stages of the software life-cycle. We believe that stochastic Petri nets are specially suited for this aim. A compositional approach for the translation of several UML diagrams into analyzable Petri net models has therefore been considered in previous papers. Following this approach, we will focus here in the depiction of a model case study from the perspective of our new performance-oriented CASE tool.

1. Introduction

Achieving a balance between the usage of strong, well-known performance formalisms and the usability of the method for non-experienced end-users may be one of the most challenging tasks the software performance engineering (SPE) [18] community has to face nowadays. Performance evaluation often requires wide knowledge in queuing theory [13] and formal performance models. Thus, a successful deployment in industrial terms depends on the degree of integration of this kind of techniques into the regular work of the software analyst or developer.

In line with these considerations, the Unified Modeling Language (UML) [5] has progressively spread as the current universal standard in software modeling. UML is a semi-formal language maintained by the Object Management Group (OMG) [16] consortium and used to specify, visualize and document artifacts of discrete event systems, being particularly suited for software development environments.

Our proposal is based on taking advantage of UML for SPE purposes. Providing a fully UML-complaint modelling framework ensures an efficient communication between software architects whereas performance issues are integrated in their everyday work in a consistent way. In previous papers [12, 11, 4, 9], this approach has been widely presented. According to that work, the performance evaluation process basically takes three steps: extension of UML diagrams with performance annotations, translation of extended UML diagrams to labelled stochastic Petri net modules [1, 6] and a final composition of the modules into a single model representing the whole system behavior. This model can be used either for validation or performance evaluation means, even though this paper focuses in the last perspective.

Currently, three different classes of UML diagrams have already been studied. All these classes belong to the set of behavioral diagrams of UML: UML statecharts (SCs) were studied in [11] by means of the UML State Machines (SMs) package, UML sequence diagrams (SDs) were discussed in [4] (as well as their relationship with SCs) and UML activity diagrams (ADs) were recently analyzed in [9].

Nevertheless, other kinds of UML diagrams may be taken into consideration in the future. That includes behavioral diagrams (as the Use Case diagram) as well as some structural (as the Class diagram) and implementation diagrams (Deployment and Component diagrams), which will allow us to avoid the infinite resource assumption. Collaboration diagrams may not need further study anyhow, as they are isomorphic to SDs.

In addition, previous work has been developed [12] to illustrate the link between SDs and SCs in the context of a real-world case study. In particular, the software retrieval service in the ANTARCTICA system [12] was profusely discussed and compared with other non-agent-based alternatives thanks to a performance analysis based on our pro-

*This work has been developed within the project P084/2001 of the Gobierno de Aragón and the project UZ00-TEC-03 of the Universidad de Zaragoza .

posal.

Our main goal in this paper is to focus in our most recent work, so as to clarify and establish the connection between ADs and the previously studied diagrams through a realistic test problem. Obviously, the whole process will be reflected in this framework.

Furthermore, our new CASE tool prototype will be presented. Its main features will be depicted, as well as the main topics to be considered in future versions. This prototype automatizes the process described below, letting the analyst model performance issues in an intuitive way. That also enables a perfect coordination between the software architect or developer and the performance engineer, establishing well-defined roles. The performance engineer would then be able to perform a profuse analysis based on the system specification without being concerned about details.

This paper is organized as follows: Section 2 recalls the main aspects of the process given in [4, 9]. Section 3 traces the guidelines of the sample case study and applies the concepts stated in the previous section to it. Section 4 presents our CASE tool prototype. Finally, section 5 concludes the paper summarizing its most relevant points besides discussing related and future work.

2. Process

Although the main rules of our SPE process were briefly considered in Section 1, there are a number of points that should be clarified regarding these guidelines. Moreover, we must specify which role is taken by each UML diagram according to the present system description potential. All these issues will be discussed in this section.

As it was previously stated, three steps are taken to obtain directly analyzable models. Firstly, our UML model is (at least) extended with a temporal interpretation of its dynamics, usually based on estimations. In particular, adding performance annotations to UML diagrams lets us define a stochastic interpretation according to their associated semantics. Tagged values will be used to ensure full UML compatibility.

In [9], our proposal for performance annotations in ADs is broadly discussed. Similarly, our performance annotations in SCs and SDs have been studied in previous works, such as the description of the ANTARCTICA Software Retrieval Service [12]. Hence, we will have recourse to the syntax illustrated in those papers.

Secondly, every UML diagram is translated into an analyzable formal model. In particular, we use a specific class of stochastic Petri nets: the labeled Generalized Stochastic Petri Nets (LGSPNs) [6] . LGSPNs are specially suitable for this task, as they allow a compositional (bottom-up) approach to construct a unique analyzable model describing the whole system behaviour.

The reader is assumed to know basic definitions of GSPN and LGSPN systems. Here we follow the notation given in [1, 6]. Note that, with respect to the definition of labeled GSPN system given in [6] both places and transitions can be labeled, moreover, the same label can be assigned to place(s) and to transition(s) since it is not required that L^T and L^P are disjoint. The particular translation rules for SCs, SDs and ADs have already been considered in [4, 11, 9], respectively, and thus they will not be discussed here.

Finally, the whole system is composed from the LGSPNs obtained in the previous step. At our current research status, the system can be described through *n* SDs, *m* SCs and *o* ADs, being $n \in \{0, 1\}$, $m \geq 1$, $n, m, o \in \mathbb{N}$. That means we have $n + m + o$ nets to be composed. Every diagram belonging to a same diagram class must be composed together (as explained on [4, 11, 9]) to form a unique LGSPN system (either by synchronization or fusion of places).

Hence, we will have (at most) three LGSPNs called $\mathcal{LS}_{sd}$, $\mathcal{LS}_{sc}$, $\mathcal{LS}_{ad}$ (for SDs, SCs and ADs, respectively). The diagram corresponding to ADs will then be composed with that for SCs through superposition of places (i.e., fusion of modules) and elimination of some spare *acknowledge* places. The resulting LGSPN system $\mathcal{LS}_{ad-sc}$ describes the whole system behavior.

As far as the SD is concerned, it lets us consider the behavior of the system under certain restrictions. That is, in terms of the resulting LGSPN, it constrains the firable sequences of $\mathcal{LS}_{ad-sc}$ (composition by synchronization). Paper [11] defines two different approaches (the 'full' case and the 'constrained' case), depending on the interpretation given to the scenario described by the SD.

Concretely, SDs are used to model patterns of interaction between classes within a particular scenario (e.g., in the depiction of a use case). Meanwhile, SCs are used to model the life-cycle for instances of a particular class and ADs to model activities performed in a particular state of a SM (that includes ADs and SCs). This latest task could be accomplished by SCs too, but ADs are rather more suitable for activities that are not dependable of external events. Notice that the system is always modelled at class level. Otherwise, we may require other kinds of formal models, as Stochastic Well-formed Nets (SWN), in order to be able to distinguish between different instances of a class (or classes) [12].

It must be noted that, after composing the entire LGSPN, there exist a lot of unnecessary elements on the net. Thus, it would be useful to apply some proper reduction techniques so as to obtain a more compact model. This matter will be a subject of future research.

3. Case study

In order to shine a light on the process outlined in section 2 a test problem will be analyzed below. The analytical

results will be stated in section 4, as well as the facilities our CASE tool provides to obtain them.

The problem consists in modelling a basic mail client. Here we will focus in the first Use Case (UC) showed in figure 1: checking mail from a server using the POP3 [14] protocol.

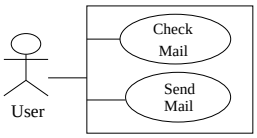

Figure 1. Use Case view of the 'mail client' model

The behavior of the referred client is rather intuitive for this UC. First, the client tries to establish a TCP connection with the server via port 110. If 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 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.

A SC has been used to depict the mail client behavior (MailClient class) for the referred UC, concretely in figure 2. Notice that the resulting GSPN for the SC has been included right below the SC. Similarly, figure 3 illustrates the server host and user behaviors via two SCs describing the POP3Server class and User actor dynamics. Nonetheless, it should be noted that it is not always possible to apply an stochastic interpretation to the user behavior.

In the original proposal [11] usage of guards in UML transitions was avoided. However, not-event-driven decisions have been considered in this example by modelling 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. This has lead to some minor adjustments to the translation rules expressed in [11]. Due

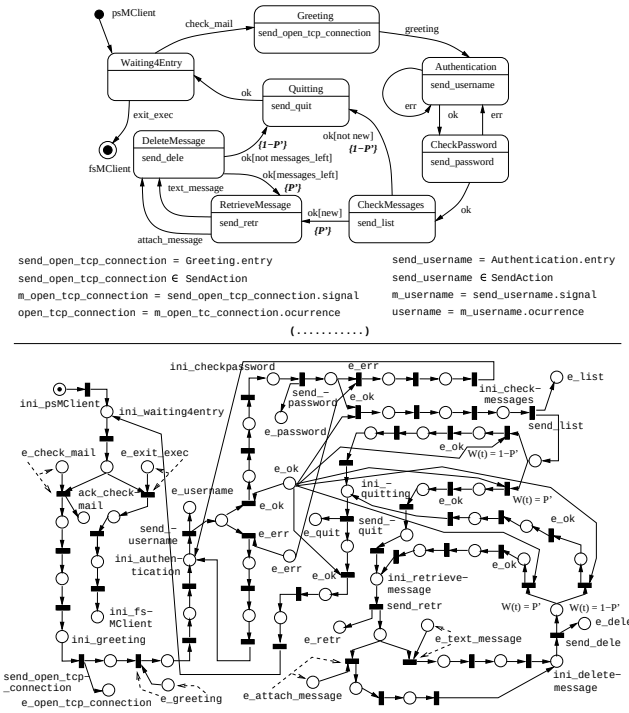

Figure 2. Statechart and resulting LGSPN for the dynamics of the class ClientHost

to the fact that these details do not fit the scope of the paper, they will be commented in future work.

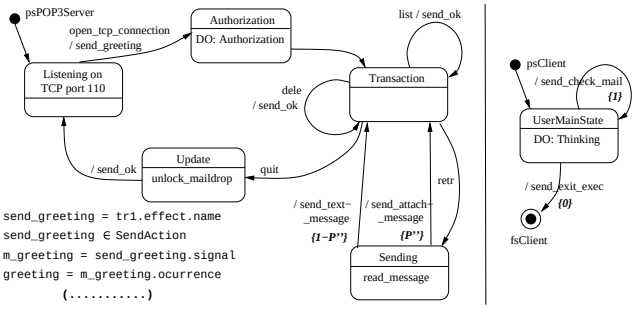

Figure 3. Statecharts for the dynamics of the classes ServerHost and User (actor)

Apart from being necessary to complete system description, the activity Authenticate associated to the state Authorization in the SC for ServerHost (figure 3) is rather relevant to the system performance. Therefore, it is necessary to model the actions performed within. Here we will use an AD (see figure 4), 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 [9] for certain tasks.

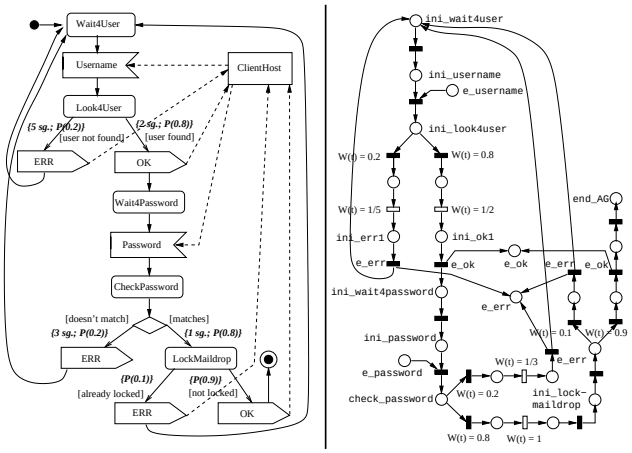

Figure 4. Activity Diagram for POP3ServerHost::Authenticate and resulting LGSPN

Finally, we use SDs to obtain performance analytical measures in a certain context of execution. Figure 5 shows an example of interaction between both server and client. Some results for this particular scenario will be obtained in section 4.

4. Final performance model & analysis

Once the final LGSPN models are obtained (following the composition rules given in [9, 11]) performance estimates can be extrapolated. 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).

Figure 6 shows some results for both cases. The graph on the left 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 on the right represents the time cost of executing the interaction illustrated in figure 5 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 [9, 11]. 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 section 1, our proposal for the establishment of a UML-complaint performance modelling framework was pointed out. The advantages of this approach have been previously discussed in this paper, and they are especially

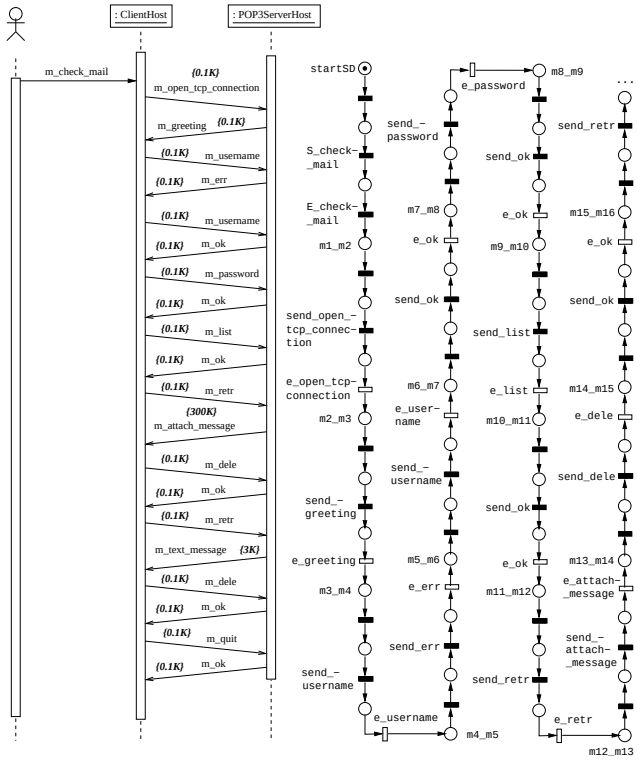

Figure 5. Sequence Diagram describing scenario, and corresponding LGSPN

strong if the process itself is automatizable. To fulfill this objective, a CASE tool prototype based in our own process has been developed.

Some features of this CASE tool (as the possibility to model and translate SDs and SCs) are currently in development phase. However, it is already possible to model ADs (full syntax support) and translate them into GSPNs, as according to [9]. Moreover, importation of models in XMI format is also being implemented. This enables the usage of the tool as a performance analysis front-end for other CASE tools or environments, such as Rational Rose® [17].

Furthermore, tool and files are fully project-oriented (in the sense that every UML element or diagram belongs to a project). This facilitates the construction (and further trans-

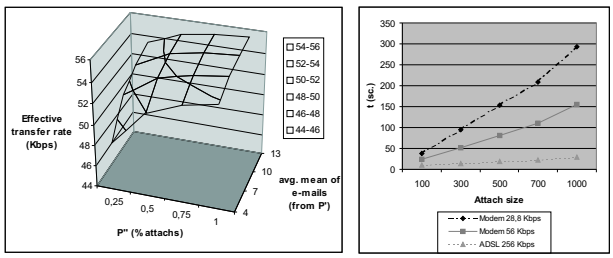

Figure 6. Some analytical results for the presented case study

lation) of complex UML models. The tool also performs basic checking of the diagrams syntax (based on the current UML specification [5]) and provides an intuitive and highly flexible GUI.

The GSPNs generated by the tool are saved in GreatSPN [7] format. These nets are subsequently processed in the referred tool in order to obtain analytical results. Figure 7 shows a snapshot of our tool (the traditional *coffeepot* sample AD, which appears in the UML specification [5], with performance annotations) and its resulting translation in GreatSPN, as it was obtained originally. Note that an special effort has been made to avoid superposition of places and transitions in the resulting nets. Support for other GSPN tools may be part of future work.

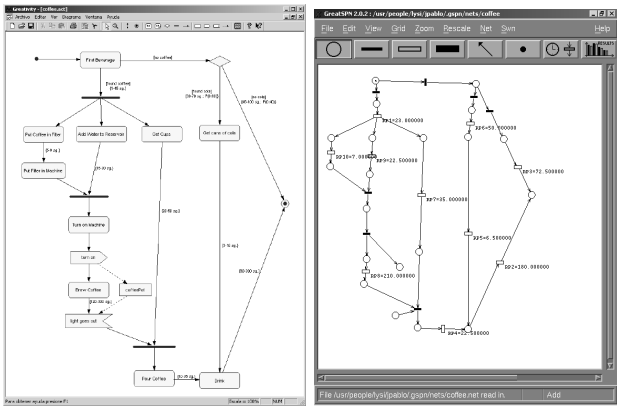

Figure 7. Tool - Extended coffeepot example and results in GreatSPN

5. Concluding remarks

In the present paper, a sample test problem has been studied so as to take a longer view on our UML-based SPE process. Moreover, our new CASE tool prototype has been presented, which supports and automatizes this process. The approach lets the software architect model the system along with performance issues in an easy, consistent fashion whereas performance models can be automatically obtained.

Concerning related work, we are unable to compare our tool due to the fact that, as far as we know, there exist four SPE tools [3, 2, 10, 15] based on UML and none of them uses stochastic Petri nets as performance model. On the other hand, we do not consider DSPNExpress2000 [8] a really UML-based SPE tool as it seems that only rather simple SCs can be used to model the system. Meanwhile, in SimML [3], simulation queuing networks (QN) models [13] for SPE are obtained from UML class diagram and SDs, while in the PERMABASE project [2] models for simulation are obtained from UML SDs and class and deployment diagrams. Finally, PROGRES [15] is a graph rewriting tool that captures XMI descriptions of UML models (using ADs, collaboration and deployment diagrams) and translates them into layered QNs.

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, 1995.

[2] D. Akehurst, G. Waters, P. Utton, and G. Martin. Predictive Performance Analysis for Distributed Systems - PERMABASE position. In *Workshop on Software Performance Prediction*, Heriot-Watt University, November 1999.

[3] L. Arief and N. Speirs. A UML tool for an automatic generation of simulation programs. In *2nd International Workshop on Software and Performance*, pp. 71–76, Ottawa, September 2000. ACM.

[4] S. Bernardi, S. Donatelli, and J. Merseguer. From UML sequence diagrams and statecharts to analysable Petri net models. In *3rd International Workshop on Software and Performance*, Rome, July 2002. ACM. To appear.

[5] G. Booch, I. Jacobson, and J. Rumbaugh. OMG Unified Modeling Language specification, v. 1.4, September 2001.

[6] S. Donatelli and G. Franceschinis. PSR Methodology: integrating hardware and software models. In *LNCS 1091*, pp. 133–152. Springer-Verlag, June 1996.

[7] The GreatSPN tool. http://www.di.unito.it/˜greatspn.

[8] C. Lindemann, A. Thummler, A. Klemm, M. Lohmann, and O. Waldhorst. Quantitative system evaluation with DSPNexpress 2000. In *2nd International Workshop on Software and Performance*, pp. 12–17, Ottawa, September 2000. ACM.

[9] J. P. López-Grao, J. Merseguer, and J. Campos. From UML activity diagrams to stochastic PNs: Application to software performance analysis. Technical report, April 2002.

[10] J. Medina, M. González, and J. M. Drake. MAST-UML: Visual modeling and analysis suite for real-time applications with UML. http://mast.unican.es/umlmast/.

[11] J. Merseguer, S. Bernardi, J. Campos, and S. Donatelli. A compositional semantics for UML state machines aimed at performance evaluation. In *6th International Workshop on Discrete Event Systems*, October 2002. To appear.

[12] J. Merseguer, J. Campos, and E. Mena. Performance evaluation for the design of agent-based systems: A Petri net approach. In M. Pezzé and S. M. Shatz, editors, *Proceedings of the Workshop on Software Engineering and PNs*, pp. 1–20, Aarhus, June 2000.

[13] M. Molloy. *Fundamentals of Performance Modelling*. Macmillan, 1989.

[14] J. Myers and M. Rose. RFC 1725: Post Office Protocol - version 3, November 1994.

[15] D. C. Petriu and H. Shen. Applying the UML performance profile: Graph grammar based derivation of LQN models from UML specifications. In *LNCS 2324*, pp. 159–177. Springer-Verlag, 2002.

[16] Object Management Group. http://www.omg.org.

[17] Rational Software Corporation. http://www.rational.com.

[18] C. U. Smith. *Performance Engineering of Software Systems*. Addison–Wesley, 1990.