Exploring roles for the UML diagrams in software performance engineering[*]

José Merseguer, Javier Campos
Dpto. de Informática e Ingeniería de Sistemas
Universidad de Zaragoza, Zaragoza, Spain
{jmerse,jcampos}@unizar.es

Abstract

It is not an overstatement to say that the gap between software design and performance evaluation techniques has caused the misuse of the last ones by software engineers. The UML profile for schedulability, performance and time [13] arose from the intention to close both fields, software engineering and performance analysis. Nevertheless the gap remains, since it is difficult for software engineers to devise which parts of their designs are suitable to represent performance requirements. The profile has started to study this problem from a scenarios viewpoint. In this work, we explore other viewpoints to deal with performance requirements at software design level.

keywords: UML, software performance engineering, software design.

1. Introduction

The study of the software time efficiency (response time, delays, throughput) requires from the software designer the most accurately possible description of the system load, system usage and the routing rates. In general, software engineers are not familiar with the notation of performance modeling, moreover this notation is too far from the artifacts they use to model software systems.

In the last years, the software performance engineering (SPE) community [14, 15, 6] has dedicated great efforts to incorporate to the software specification languages the abilities to describe performance requirements in understandable terms for software engineers. These efforts have culminated in the adoption of the UML profile for schedulability, performance and time [13]. The section of this profile that deals with performance modeling, maps concepts in the performance analysis domain into concepts in the UML language [12], then bridging the gap between software designers and performance analysts. Other languages have been used with these purposes [7].

The profile proposes "determining a system's performance characteristics" using scenarios, that can be modeled using collaborations or activity graphs. In this paper, we explore an alternative approach to scenarios to determine software system's performance characteristics, attending at the object's life viewpoint [9].

Concretely, in this work we have focussed in the use case diagram and in the statechart diagram. We propose to capture performance requirements for each class with relevant dynamic behavior in the system, by specifying these requirements in the statechart that models its life. Also, it will be identified which model elements in the diagrams are suitable to describe performance aspects and its mapping into concepts of the profile. The activities in the statechart diagram can be refined using activity diagrams for a most accurate description of the requeriments. But the activity and sequence diagrams will be subject of future research to find theirs roles in SPE under this perspective.

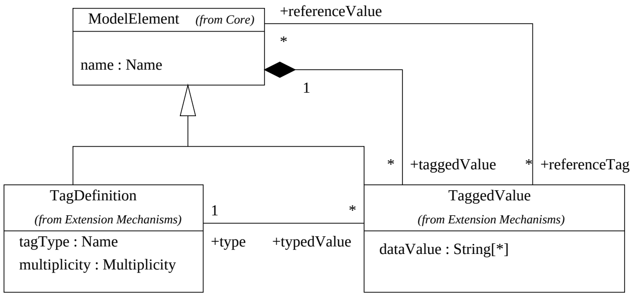
The paper is organized as follows. Section 2 revises the performance modeling component of the UML profile [13] and briefly recalls our proposal to specify performance requirements [9]. Sections 3 and 4 study the use case diagram and the statechart diagram, respectively. For each diagram a short description taken from the UML specification [12] is given to put the reader in context (if more information is needed, we refer to the manual); after that, the role of the diagram concerning performance goals is explored; and finally, the performance annotations suggested for the diagram are given. The paper ends by giving some conclusions in section 5.

[*]This work has been developed within the projects TIC2002-04334-C03-02 of the Spanish CICYT and P084/2001 of the Gobierno de Aragón.

	ANNOTATION	KIND*	REFERENCED VALUE
USE CASE DIAGRAM	Probability that an actor executes a use case	A	Association link
SEQUENCE DIAGRAM	Probability of success of a message	B	Message
	Message size	C	Message
STATECHART DIAGRAM	Activity duration	C	Action (with doActivity role)
	Probability of success of a message	B	Transition[1]
	Message size	C	Event
ACTIVITY DIAGRAM	Activity duration	C	Timed transition
	Probability to take the transition	B	Transition (timed or immediate)

*A=System usage, B=Routing rate, C=System load.
[1]It is not attached to an event since the transition can be automatic.

Table 1. Summary of the annotations proposed in pa-UML .



```
TaggedValue.name = performance annotation
TaggedValue.dataValue = {a concrete annotation}
TaggedValue.type.name = performance annotation
TaggedValue.type.multiplicity = 1
TaggedValue.type.tagType = [system usage | system load |
 routing rate]
```

Figure 2. Metamodel of the annotations proposed in pa-UML .

2 UML profile for schedulability, performance and time

The UML profile for schedulability, performance and time [13], among other purposes, enables the construction of software models that could be analyzed to make quantitative predictions early in the development process.

When studying which performance requeriments should be or can be represented by a given UML diagram (i.e. its role from the performance analysis viewpoint) to get a notation to specify them becames necessary. The *Performance Modeling* component of the UML profile [13], we will refer to this component as the *profile* later on, provides facilities for this task and also for specifying QoS characteristics and execution parameters. The following paragraph and the Figure 1 briefly recall some performance concepts as proposed in the *profile*.

A *performance context* specifies *scenarios* that are used to explore dynamic situations. An scenario is composed by *steps* which are executions that take finite time and use *resources*. The resources can be *passive*, then representing devices or logical entities, or *active*, the usual servers in a performance model. Scenarios are executed by *workloads*, being *open* when they are characterized by requests that arrive at a given rate or *closed* when they have a number of users which cycle executing the scenario.

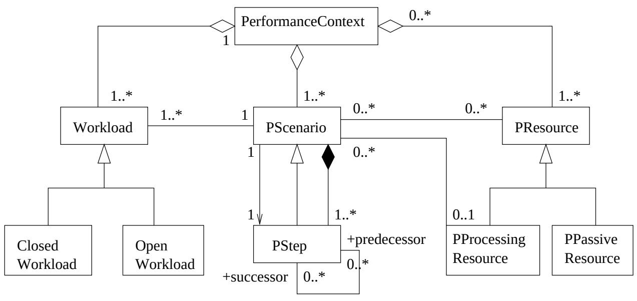


Figure 1. Performance analysis model from [13].

The *profile* maps these performance concepts into UML equivalents. Concretely, they are mapped into concepts of the collaborations and activity graphs packages. Obviously, a complete definition for the system performance requeriments needs complementary views. So, in sections 3 and 4, the possible performance roles for the use case and the statechart diagrams, are explored while trying to map these concepts into UML equivalents.

Previously to appear the *profile*, we worked in the definition of a language (pa-UML [9]) to specify performance requeriments in UML. Table 1 summarizes for each kind of diagram, the annotations proposed in pa-UML as well as the model element affected by the annotation. Figure 2 depicts how pa-UML uses the tagged values to annotate the system usage, the system load and the routing rates. Despite using tagged values, our proposal differs from the *profile* since we do not combine them with stereotypes and neither a “tagged values language” was proposed. On the other hand, our proposal allows to (semi)automatically generate a performance model in terms of Petri nets [4]. Then performance predictions can be computed.

3 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 [12].

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 [12].

In the use case diagram in Figure 3 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.

3.1 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 [11].

The existence of a use case diagram is not mandatory to obtain a performance model. In [10] was shown that a performance model for the whole system can be obtained from the statecharts that describe it.

It is important to recall the proposal in [2], 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 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

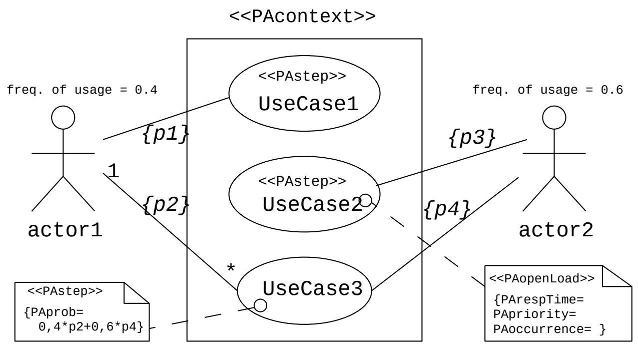


Figure 3. Use case diagram with performance annotations.

$\sum_{j=1}^{n} P_{ij} = 1$, the probability of a sequence diagram corresponding to the use case x to be executed is:

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

The previous formula, taken from [2], is important because it allows to assign a "weight" to each particular execution of the system. As an example, see in Figure 3 the annotations attached to UseCase3.

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

3.2 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. Therefore, the base classes for these stereotypes will be incremented with the class *UseCase*. 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 3). They are useful to calculate the probability for each *Step*, i.e. use case, using equation (1).

4 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 section 2-12 of [12]. Recent studies of their semantics can be found in [10, 5].

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 [12].

A composite state is decomposed into two or more concurrent substates (regions) or into mutually exclusive disjoint substates [12].

4.1 Role of the statechart diagram 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 modelling the statecharts for the system are sufficient enough to obtain a performance model [10]. 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 becames 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 [1].

In a statechart diagram the useful model elements

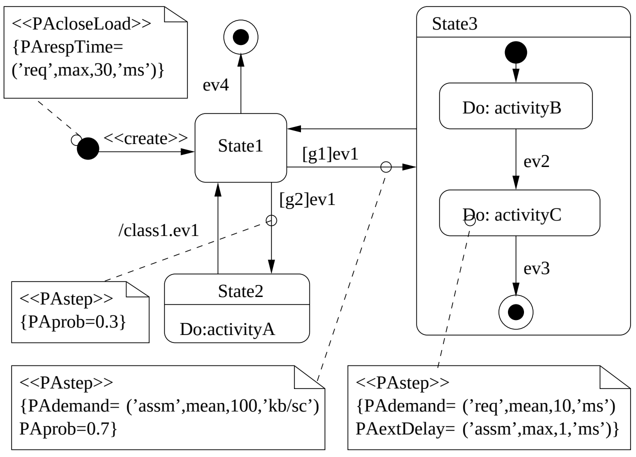


Figure 4. Statechart with performance annotations.

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 modelled. 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.

4.2 Performance annotations

As proposed in [13] for the activity-based approach, the open or closed workload (induced by the object in our case) is associated with the first step in the diagram, in this case the transition stereotyped ≪create≫, see Figure 4.

In pa-UML [9] 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 4. 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 [12], 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 indeterminism (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 4. 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 4.

5 Conclusions

In this work, we have approached to the use case diagram and the statechart diagram in order to study how they can be used by a software engineer to identify (model) performance requirements. Moreover, this approach keeps in mind the UML profile for schedulability, performance and time, which today becames a must for software performance engineers [3, 8]. The sequence and activity diagrams, addressed by the *profile*, will be subject of future research under the viewpoint followed in this work.

References

[1] S. Bernardi, S. Donatelli, and J. Merseguer, *From UML sequence diagrams and statecharts to analysable Petri net models*, in Inverardi et al. [6], ISBN 1-58113-563-7, pp. 35–45.

[2] V. Cortellessa and R. Mirandola, *Deriving a queueing network based performance model from UML diagrams*, in Woodside et al. [15], ISBN 1-58113-195-x, pp. 58–70.

[3] 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. [15], ISBN 1-58113-195-x, pp. 83–88.

[4] F. DiCesare, G. Harhalakis, J. M. Proth, M. Silva, and F.B. Vernadat, *Practice of Petri nets in manufacturing*, Chapman & Hall, London, 1993.

[5] E. Domínguez, A.L. Rubio, and M.A. Zapata, *Dynamic semantics of UML state machines: A metamodelling perspectiv*, Journal of Database Management **13** (2002), 20–38.

[6] 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, ISBN 1-58113-563-7.

[7] C. Juiz, R. Puigjaner, and K. Jackson, *Performance modelling of interaction protocols in soft real-time design architectures*, Performance Engineering. State of the Art and Current Trends, LNCS vol. 2047, Springer-Verlag, 2001, pp. 300–316.

[8] J.L. Medina, M. Gonzalez, and J.M. Drake, *MAST real-time view: A graphic UML tool for modeling object-oriented real-time systems*, Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001) (London, UK), IEEE Computer Society Press, December 2001, pp. 245–256.

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

[10] 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.

[11] J. Merseguer, J. Campos, and E. Mena, *Analysing internet software retrieval systems: Modeling and performance comparison*, Wireless Networks (WINET) **9** (2003), no. 3, 223–238.

[12] Object Management Group, http:/www.omg.org, *OMG Unified Modeling Language specification*, September 2001, version 1.4.

[13] Object Management Group, http:/www.omg.org, *UML Profile for Schedulabibity, Performance and Time Specification*, March 2002.

[14] 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, ISBN 1-58113-060-0.

[15] 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, ISBN 1-58113-195-x.